

# AI Summary

Yingxin Wu

2020 年 9 月 9 日

## 目录

<b>1 Chapter 1: 导引</b>	<b>4</b>
<b>2 Chapter 2: Agents</b>	<b>4</b>
<b>3 Chapter 3: 搜索</b>	<b>6</b>
3.1 基础定义 . . . . .	6
3.2 无信息搜索 . . . . .	9
3.3 有信息搜索 . . . . .	10
3.3.1 贪婪最佳优先搜索(Best-First) . . . . .	10
3.3.2 (重点)A*搜索(Best-First) . . . . .	10
<b>4 Chapter 4: 局部搜索</b>	<b>12</b>
4.1 (重点)爬山法(贪婪局部搜索) . . . . .	12
4.2 模拟退火搜索 . . . . .	13
4.3 局部束搜索 . . . . .	13
4.4 遗传算法 . . . . .	13
<b>5 Chapter 6: 约束满足问题</b>	<b>15</b>
5.1 约束的定义 . . . . .	15
5.2 约束问题形式化 . . . . .	15
5.3 约束的增量形式化 . . . . .	16
5.4 约束相容性 . . . . .	16
5.5 CSP 回溯搜索 . . . . .	16
5.6 回溯优化 . . . . .	17
5.6.1 变量顺序 . . . . .	17
5.6.2 取值顺序 . . . . .	17
5.6.3 失败预警 . . . . .	17
5.6.4 利用约束结构 . . . . .	18
5.7 局部搜索 . . . . .	19

<b>6 Chapter 5: 博弈</b>	<b>19</b>
6.1 博弈定义 . . . . .	19
6.2 博弈优化 . . . . .	20
6.3 截断 . . . . .	21
6.4 机会博弈 . . . . .	22
6.5 部分可观察博弈(不考) . . . . .	22
<b>7 Chapter 7: 逻辑Agent</b>	<b>23</b>
7.1 基本概念与 Wumpus World 案例 . . . . .	23
7.2 命题逻辑 . . . . .	24
7.3 前向链接 . . . . .	25
7.4 后向链接 . . . . .	26
7.5 归结算法 . . . . .	27
7.6 优缺点 . . . . .	29
<b>8 Chapter 8: 一阶逻辑</b>	<b>29</b>
<b>9 Chapter 9: 一阶逻辑推理</b>	<b>32</b>
9.1 Unification 合一 . . . . .	34
9.2 GMP 一般化分离规则 . . . . .	34
9.3 前向链接 . . . . .	35
9.4 后向链接 . . . . .	36
9.5 归结算法 . . . . .	37
<b>10 不确定度的量化与概率推理</b>	<b>37</b>
10.1 概率基础 . . . . .	37
10.2 贝叶斯网络 . . . . .	39
<b>11 Chapter 18: Learning</b>	<b>42</b>
11.1 决策树 . . . . .	42
11.2 K 近邻 . . . . .	43
11.3 线性预测 . . . . .	43
11.3.1 问题形式化 . . . . .	43
11.3.2 过拟合解决方案 . . . . .	44
<b>12 Chapter 20: SVM</b>	<b>45</b>
12.1 分类间距 . . . . .	45
12.2 问题形式化 . . . . .	46
12.3 软间隔 . . . . .	46
12.4 损失函数 . . . . .	47
12.5 非线性 SVM . . . . .	47
<b>13 Chapter 21: Logistic回归</b>	<b>48</b>

<b>14 Chapter 22:</b> 神经网络	<b>49</b>
14.1 CNN . . . . .	50
<b>15 无监督学习</b>	<b>51</b>
15.1 PCA . . . . .	51
15.2 聚类 . . . . .	53

# 1 Chapter 1: 导引

**Major Components:** knowledge, reasoning, language, understanding, learning

Thinking humanly    Thinking rationally  
Acting humanly    **Acting rationally**

**Thinking humanly:** cognitive revolution, brain knowledge

studying human subjects or neurological data, which is distinct from AI now

**Thinking rationally**

laws of thought

Logical deliberation can't be generalized to all intelligent behavior and tends to do the wrong thing in the presence of uncertainty.

**Acting humanly: Turing Test**

Turing test is not reproducible or amenable to mathematical analysis.

**Acting rationally: Maximize your expected utility**

Rational agent:

- doing the right thing (perceives and acts) which means to maximize goal achievement, given the available information.
- Entirely dependent on goals.
- Irrational ≠ insane, irrationality is sub-optimal action. Rational ≠ successful.

**Problems:** computation complexity, information complexity

# 2 Chapter 2: Agents

对于每个可能的感知序列，基于已知感知序列提供的信息，和智能体已有的先验知识，选择能使其性能最大的行为。

**Performance measure:** An objective criterion for success of an agent's behavior, evaluates the environment sequence.

**Environment types:**

Fully observable	Deterministic	Episodic(atomic)	Static (deliberating)	Discrete	Single agent
Partially observable	Stochastic	Sequential	Dynamic	Continuous	Multiagent

	Chess with a clock	Chess without a clock	Taxi driving
Fully observable	Yes	Yes	No
Deterministic	Strategic	Strategic	No
Episodic	No	No	No
Static	Semi	Yes	No
Discrete	Yes	Yes	No
Single agent	No	No	No

图 1: example for environment type

strategic: the environment is deterministic except for the actions of other agents.

semidynamic: the environment itself does not change with the passage of time but the agent's performance score does.

**PEAS Example:** Automated taxi Agents interact with environments through actuators and sensors.

performance measure	safety, right destination, comfort, legality, profits
Environment	street, traffic, pedestrians
Actuators	steering, accelerator, brake
Sensors	gauges, accelerometers, GPS

**Agent function:** maps from percept histories to actions:  $[f : \mathcal{P} \rightarrow \mathcal{A}]$ , describes what the agent does in all circumstances.

**Agent program:** runs on the physical architecture to produce f (implement (some) agent functions)

**table-lookup agent:** a huge table mapping from possible situations to actions, which has several drawbacks:

1. take a long time to build one, even with learning
2. huge, no autonomy.

### Agent types

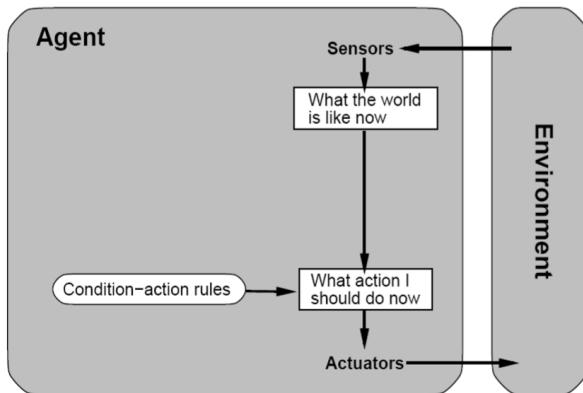


图 2: simple reflex agents

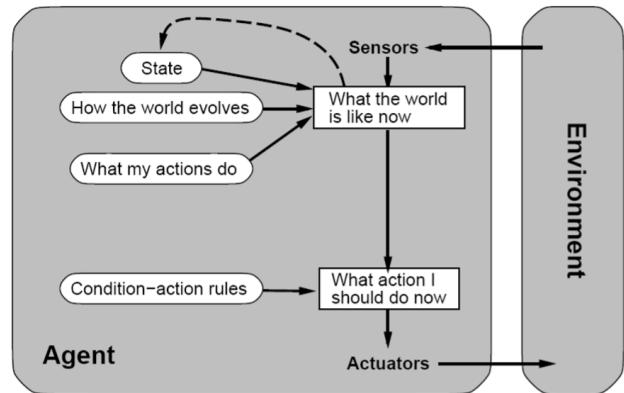


图 3: reflex agents with state

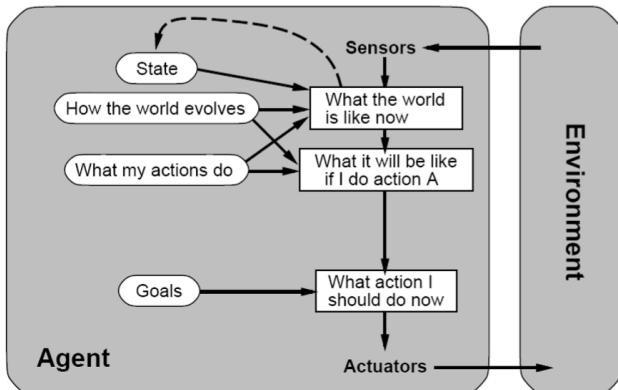


图 4: goal-based agents

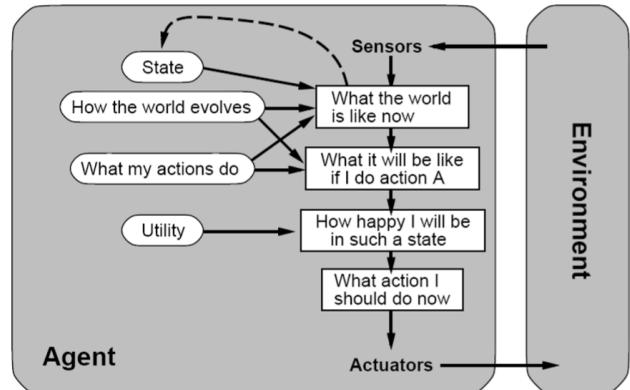


图 5: utility-based agents

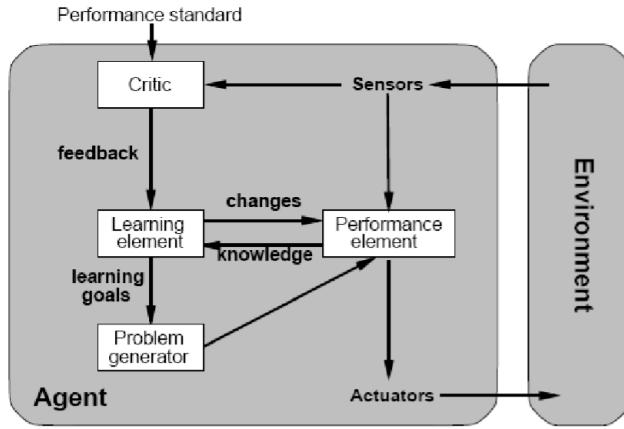


图 6: learning agents

### 3 Chapter 3: 搜索

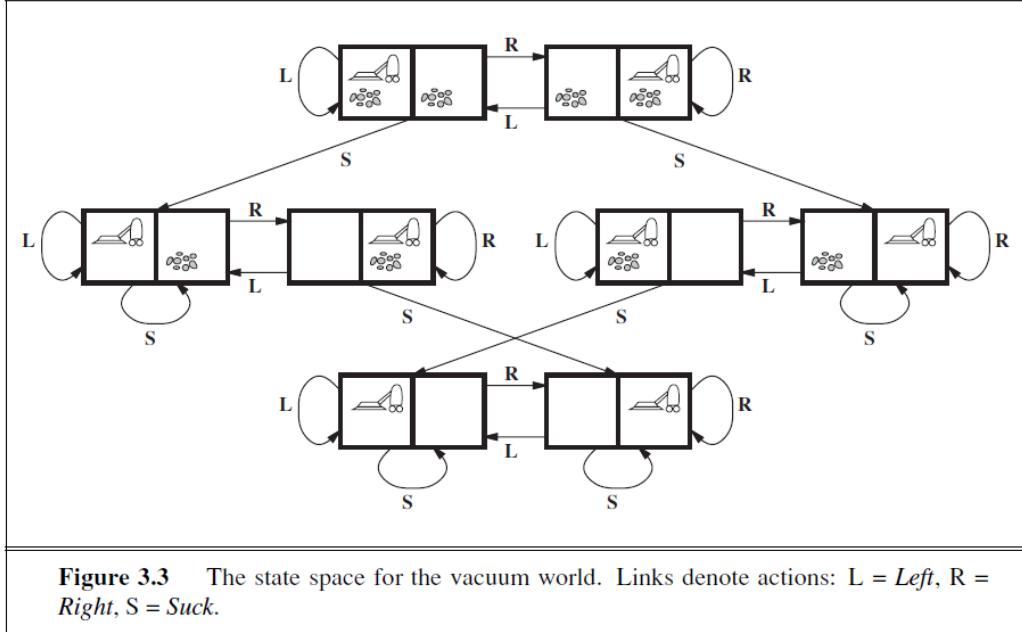
当问题的求解不能通过一步完成时, Agent需要找到一组行动序列达到目标, 该过程称为搜索。问题求解agent采用原子表示, 不关心问题的内部结构。本章只讨论确定性, 可观察, 静态和完全可知的情况。

#### 3.1 基础定义

**问题的形式化** 1) **抽象**: 保持有效抽象(抽象后的解可拓展为更细节世界中的解); 抽象后的问题容易完成。  
 2) **描述**: 只注重最终状态, 而不是怎么到这个状态的问题形式化(增量形式化与完整状态形式化), 无需考虑路径耗散。初始状态, 行动, 转移模型构成状态空间。形式化过程中应该使得状态空间尽量小。

状态	真空吸尘器 Agent与灰尘位置	八皇后 8个棋子分布	玩具问题 正整数	飞机航行 地点, 时间等
初始状态	/	/	4	出发的地点时间
行动	Left,Right,Suck,NoOp	增量形式化(放);状态式(移动)	!,pow,sqrt	当前时间后乘机起飞
转移模型	新环境及位置	新棋盘	由数学定义	( $p_1, t_1$ ) $\rightarrow$ ( $p_2, t_2$ )
目标测试	所有位置干净	8个皇后且不相互攻击	得到要求的正整数	到达目的地
路径耗散	移动的步数	/	/	金钱, 等待时间等

目标测试可能为可枚举的集合, 或某种特性的状态。路径耗散最小的解称为**最优解**。



```

1   function SIMPLE-PROBLEM-SOLVING-AGENT()  return  an  solution
2       persistent:seq = [] , an  action  sequence
3           state ,
4               goal = [] ,
5               problem  formation .
6
7       state = UPDATE-STATE(state , percept)
8       if  seq == []  then
9           goal = FORMULATION-GOAL
10          problem = FORMULATION-PROBLEM
11          seq = SEARCH(problem)
12          if  seq == failure  then  return  null
13          action = FIRST(seq)
14          seq = REST(seq)
15          return  action

```

## Search Function

1. 性能指标 完备性: 保证找到问题的解; 最优性: 找到代价最小的解; 时间复杂度, 空间复杂度.

复杂度常由 **b**: 分支因子, 即结点最大后继数; **d**: 目标节点所在最浅深度; **m**: 状态空间中任何路径的最大长度衡量。

## 2. 数据结构

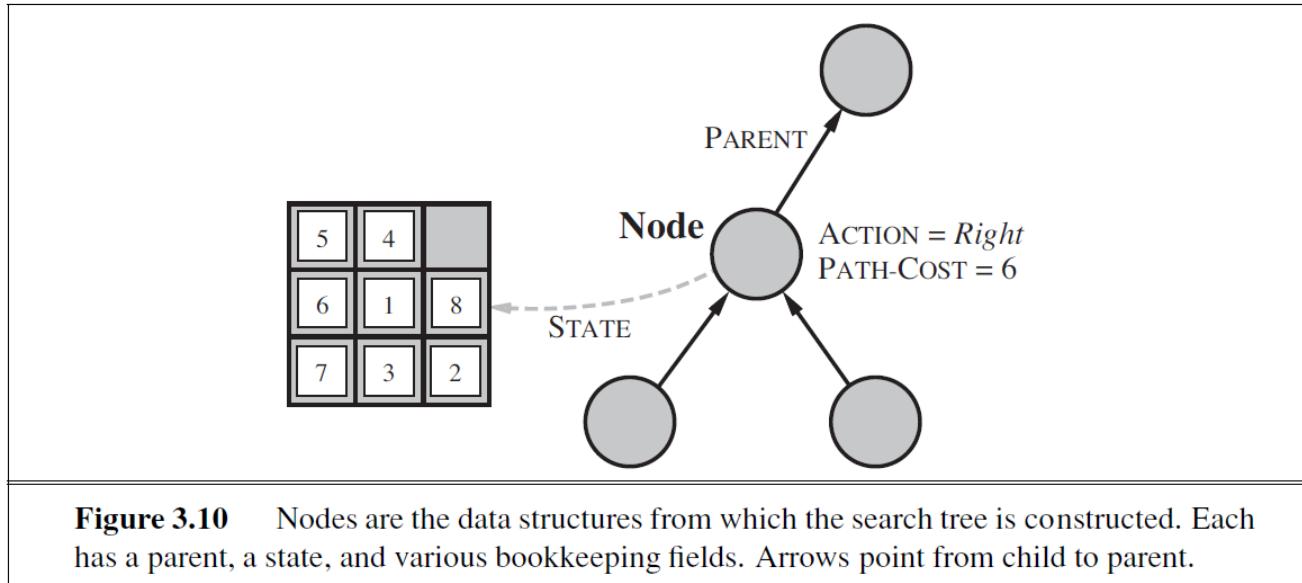
**n.STATE**: 对应空间中状态

**n.PARENT**: 父节点, 求得问题的解后用来找到解路径

**n.ACTION**: 父节点到该节点的行动

**n.PATH-COST**: 用  $g(n)$  表示, 到达此状态的路径消耗

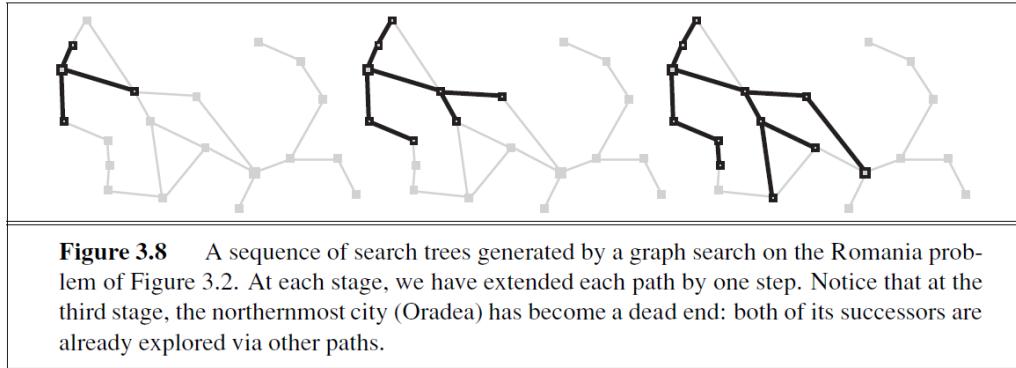
搜索树：Node 对应 State, Edge 对应 Action. 边缘：所有待扩展的叶节点的集合.



边缘队列中第一个结点为下一个要拓展的结点，反映了不同的搜索策略，如FIFO, LIFO，优先级队列。

```
1   function TREE-SEARCH(problem)  return a solution or failure
2       initialize(frontier)
3       loop do
4           frontier.sort(criteria)
5               if frontier == [] then return failure
6               leaf = frontier.pop()
7               if leaf == goal then return solution
8               frontier.append(expanding nodes)
9
10      function GRAPH-SEARCH(problem)  return a solution or failure
11          initialize(frontier)
12          explored_set = []
13          loop do
14              frontier.sort(criteria)
15              if frontier == [] then return failure
16              leaf = frontier.pop()
17              if leaf == goal then return solution
18              explored_set.append(leaf)
19              frontier.append(expanding nodes and not in frontier)
```

状态空间被分为探索区域与未探索区域，查重将极大缩小状态空间。



## 3.2 无信息搜索

除了问题定义与状态信息外没有其他附加信息。

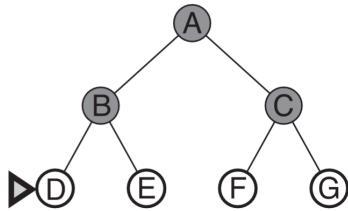


图 7: BFS

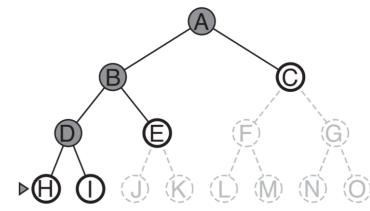


图 8: DFS

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes <sup>a</sup>	Yes <sup>a,b</sup>	No	No	Yes <sup>a</sup>	Yes <sup>a,d</sup>
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes <sup>c</sup>	Yes	No	No	Yes <sup>c</sup>	Yes <sup>c,d</sup>

**Figure 3.21** Evaluation of tree-search strategies.  $b$  is the branching factor;  $d$  is the depth of the shallowest solution;  $m$  is the maximum depth of the search tree;  $\ell$  is the depth limit. Superscript caveats are as follows: <sup>a</sup> complete if  $b$  is finite; <sup>b</sup> complete if step costs  $\geq \epsilon$  for positive  $\epsilon$ ; <sup>c</sup> optimal if step costs are all identical; <sup>d</sup> if both directions use breadth-first search.

### 1. BFS

**特点:** 边缘队列被组织为 FIFO; 在结点生成时就进行 GOAL-TEST(), 免得多测试一层。

**复杂度:** 时间复杂度(结点总数) $O(b^d) = b + b^2 + \dots + b^d$ ;

FIFO 队列中最多存储 $O(b^d - 1)$ 个节点在探索集中,  $O(b^d)$ 个节点在边缘节点集中, 所以空间复杂度由边缘节点机的大小决定, 即空间复杂度为 $O(b^d)$ .

**优点:** 宽度优先搜索是完备的, 能够找到目标节点, 且能保证最优(在所有行动的代价都相同的情况下)。

**缺点:** 在深度 d 比较大的时候, 时间和空间都会指数爆炸!

## 2. Uniform-cost search (不考)

**特点:**边缘队列被组织为按 $g(n)$ 优先级队列；在结点拓展时才进行GOAL-TEST(),可以防止路径的次优性。

**复杂度:**由路径代价来引导，故设 $C^*$ 为最优路径代价， $\varepsilon$ 为每个行动最低代价，时间与空间复杂度均为 $O(b^{1+\lfloor C^*/\varepsilon \rfloor})$ ,为生成的最多结点与边缘集最大数目。

**性能:** 是完备的，在代价为正的前提下是最优的，时空开销比BFS大。

## 3. DFS

**特点:**边缘队列为 LIFO 队列。

**复杂度:** 时间复杂度与整棵搜索树的深度有关，即需要遍历整棵搜索树，为 $O(b^m)$ ;

空间复杂度，树搜索下复杂度为 $O(bm)$ , 图搜索时为 $O(b^m)$ ，若使用回溯搜索则为 $O(m)$ .

**优点:** 树搜索时空间复杂度为线性复杂度。

**缺点:** 不是完备的(无限空间与树搜索，有限的空间里图搜索是完备的)。不是最优的。且时间开销很大。

## 4. Limited-DFS

**特点:**深度为  $l$  的结点视为无后继，即指定深度界限，最好为状态空间直径；深度优先可视为深度受限的特例；失败分 *failure* 和 *cutoff*.

**优点:** 解决了无限深度的问题；**缺点:**  $l < d$  时不完备， $l > d$  时完备但不最优。

## 5. IDS

**特点:** 增大深度限制，多次进行深度受限搜索。

**复杂度:** 时间复杂度 $(d+1)b^0 + db^1 + (d-1)b^2 + \dots + b^d = O(b^d)$ 。空间复杂度与深度最大时的深度受限搜索相同，为 $O(bd)$ .

**优点:** 完备(**b** 有限)且最优(代价非递减)。

## 6. 双向搜索

**特点:** 运行两个搜索，一个从初始往目标状态，一个从目标往初始状态；效率高因为 $b^{d/2} + b^{d/2} << b^d$ ；若目标状态明确则可行，若不明确则很难应用，如n皇后问题。总结：算法的不同在于扩展方法。需要复杂性权衡，但当状态空间很大时，性能很差(最坏情况或典型情况)

### 3.3 有信息搜索

结点基于评价函数  $f(n)$  排成优先级队列， $f(n)$  由启发函数  $h(n)$  构成，用于评价该 Node 是否理想，有 $h(Goal) = 0$ 。

好的启发式函数能直达目标，但坏的启发式函数就像一个被错误引导的 DFS。

#### 3.3.1 贪婪最佳优先搜索(Best-First)

**特点:** 边缘队列被组织为按  $f(n) = h(n)$  优先级队列。

**复杂度:** 时空复杂度均为 $O(b^m)$ —(worst DFS). 由于是图搜索，需要记录所有的结点。

**缺点:** 不完备(对无限空间以及树搜索(P83，会陷入死循环)，在有限的空间里图搜索是完备的)，不能保证最优。

#### 3.3.2 (重点) $A^*$ 搜索(Best-First)

**特点:** 边缘队列被组织为按 $f(n) = g(n) + h(n)$ 优先级队列。

**复杂度:** 时空复杂度均为 $O(b^{\varepsilon d})$ , 空间上仍然需要存储所有结点。

**优点:** 完备( $b, C^*$ 有穷)且最优( $h$ 若为可采纳的/乐观的, 树搜索是最优的; 若为一致的, 图搜索是最优的)。

注意到T,Z等被剪枝了, 即无需检验就直接被排除。对一致的启发式函数,  $A^*$ 为效率最优的, 即没有其他的最优算法能保证效率高于它。

**缺点:** 仍然需要指数量级的时间与存储空间, 在状态很多的情况下, 甚至不能探索完所有  $f(n) \leq f^*$  的结点。

记  $C^*$  为最优路径的实际损耗, 则

$A^*$  扩展满足  $f(n) < C^*$  的所有结点

$A^*$  扩展一些满足  $f(n) = C^*$  的结点

$A^*$  不扩展满足  $f(n) > C^*$  的任何结点

### (重点)启发式函数

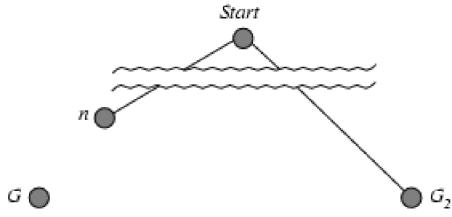


图 9: 可采纳的最优证明

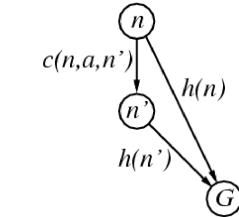


图 10: 一致性

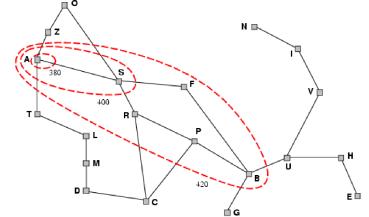


图 11: 一致性的最优证明

**性质1:** 可采纳  $h(n) \leq h^*(n)$ ,  $h^*(n)$  为真实估计值, 即保证估计是乐观的。

**Theorem1:** 如果  $h(n)$  是可采纳的, 那么使用树搜索的  $A^*$  是最优的。

**Proof** 假定有次优解  $G_2$  已产生并且在边缘队列中。设  $n$  为边缘队列中未被扩展的结点, 并且  $n$  在从出发点到达最优目标  $G$  的最短路径上。则有

```

1      f(G2) = g(G2),      since h(G2) = 0
2      g(G2) > g(G),      since G2 is suboptimal
3      f(G) = g(G),      since h(G) = 0
4      f(G2) > f(G),      from above
5      h(n) <= h*(n),      since h is admissible
6      g(n) + h(n) <= g(n) + h*(n) = g(G) = f(G),
7      since n is on a shortest path to an optimal goal G
8      f(n) <= f(G)
9      Hence f(G2) > f(n), and A* will never select G2 for expansion

```

**性质2:** 一致的  $h(n) \leq c(n, a, n') + h(n')$ , 进一步有

$$f(n') = g(n') + h(n') = g(n) + c(n, a, n') + h(n') \leq g(n) + h(n) = f(n)$$

即  $f(n)$  沿任何路径都是不递减的。

**Theorem2:** 如果  $h(n)$  是一致的, 那么使用图搜索的  $A^*$  是最优的。

**Proof**  $A^*$  按  $f$  值的递增顺序展开节点, 即按  $f$  等值线来扩展结点, 保证  $f$  值小的最先被拓展。

良好的启发式函数使有效分支因子  $b^*$  接近于1。

对任意结点  $h_2(n) \geq h_1(n)$ ,  $h_2(n)$  比  $h_1(n)$  占优势, 对应使用  $h_2(n)$  的  $A^*$  比使用  $h_1(n)$  的  $A^*$  扩展结点数更少。

一个松弛问题的最优解代价是原问题的可采纳的启发式，关键在于松弛问题的最优解成本不大于实际问题的最优解成本。

合成的启发式  $h(n) = \max\{h_1(n), \dots, h_m(n)\}$  也是可采纳的，且比任何一个成员更占优势。

\*\*如何设计

### Example

#### 构造松弛问题

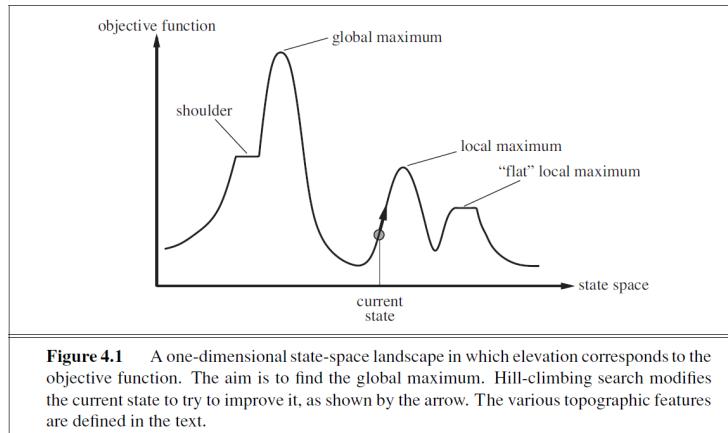
- ▣ 原问题：一个棋子可以从方格A移动到方格B，如果A与B水平或者垂直相邻而且B是空的
- ▣ 松弛1：一个棋子可以从方格A移动到方格B，如果A与B相邻 —  $h_2$
- ▣ 松弛2：一个棋子可以从方格A移动到方格B，如果B是空的
- ▣ 松弛3：一个棋子可以从方格A移动到方格B —  $h_1$

## 4 Chapter 4：局部搜索

一般不在乎怎么求解，只在乎解是什么。保持单一的“当前”状态，努力改善它，直到你无法让它变得更好。对于最优化问题，我们的目标是找到目标函数的全局最大值或最小化代价函数。

优点(1)不留搜索路径，故使用很少的内存；(2)对状态空间很大或无限的问题仍然适用。

缺点：不完备，不最优。



### 4.1 (重点)爬山法(贪婪局部搜索)

**最陡上升爬山法** 不断向最好的相邻的状态移动，如果没有这样的状态，那么算法停止。不维护搜索树，因此只需记录当前状态与目标函数值。往往会困在局部极大值，或困在造成一系列局部极大值的山脊，也可能在高原中迷路。有许多变形(P108)。

**随机重启爬山法** 多次运行爬山法，每次选择不同的初始位置，避免陷入局部最优。最后从返回  $k$  个局部最优值中选择最大(小)的一个。

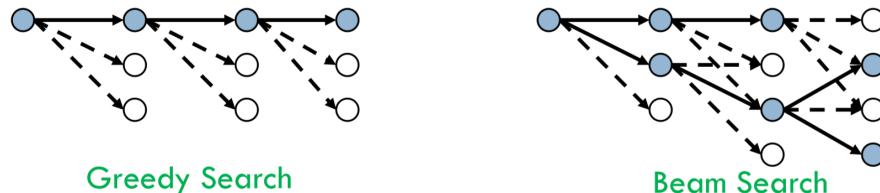
## 4.2 模拟退火搜索

有时为了避免局部最优，需要移动到一个较差的后继。通过允许一些“坏的”动作，但逐渐减少它们的频率，来逃避局部最大值。

优点：如果  $T$  下降得足够慢，那么模拟退火搜索将会找到一个概率接近 1 的全局最优解。

```
1     function SIMULATED-ANNEALING(problem , schedule)  return final state
2         input: problem ,
3                 schedule: temperature = schedule(time)
4         current = MAKE-NODE(problem .INITIAL-STATE)
5         for t = 1 to upbound
6             T = schedule(t)
7             if T == 0 then return current
8             next = a randomly selected successor of current
9             E = next.VALUE - current.VALUE
10            if E > 0 then current = next
11            else current = next.PROB(e^(E/T))
```

## 4.3 局部束搜索



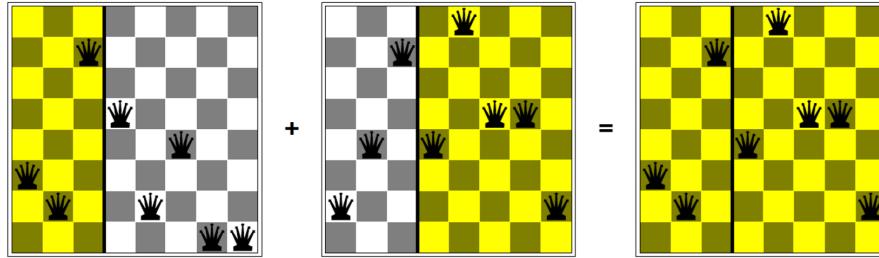
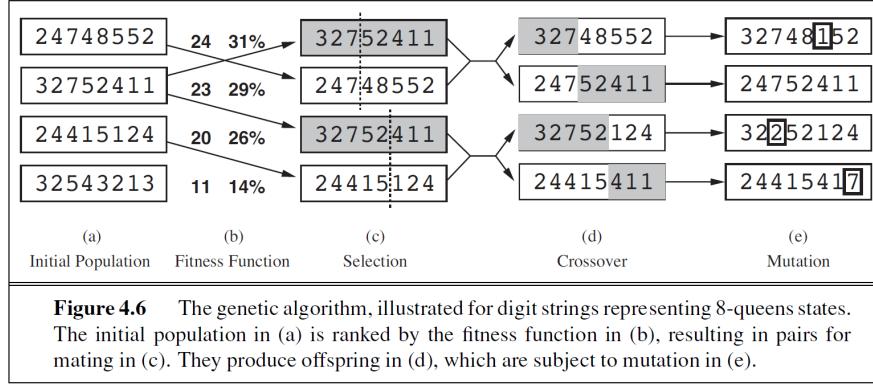
初始时选择  $k$  个状态，生成所有后继，从中选择最优的  $k$  个状态继续循环；若目标状态在生成的  $k$  个状态中，则停止。

缺点：很多情况下，所有  $k$  个状态都在同一个局部最优点上(或者相同的 hill)。

随机束搜索 不选择最优的  $k$  个状态，而是根据状态值对应的概率，保留随机性地选择  $k$  个状态，类似于自然选择。

## 4.4 遗传算法

随机束搜索的一个变形，结合了两个父结点的状态，同时考虑了变异的因素。



```

1   function GENETIC-ALGORITHM( population , FITNESS-FN)  return an individual
2     input: population , individuals
3           FITNESS-FN, a function measures the fitness of an individual
4
5     repeat
6       new_population = []
7       for i in range(1, population.length)
8         x = RANDOM_SELECTION( population , FITNESS_FN)
9         y = RANDOS_SELECTION( population , FITNESS_FN)
10        child = REPRODUCE(x, y)
11        child.MUTATE(random probability)
12        new_population.append(child)
13      population = new_population
14    until some individual is fit enough or enough time has elapsed
15    return the best individual in population
16
17  function REPRODUCE(x, y)  return an individual
18    input: parents
19    c = np.random.int(x.length)
20    return APPEND(x.SUBSTRING(1, c) + y.SUBSTRING(c+1, n))

```

总结：好的搜索策略应该

引起运动—避免原地踏步；系统—避免兜圈；运用启发函数—缓解组合爆炸

搜索树结点有重复，搜索图无重复(每次都需要查重)

## 5 Chapter 6: 约束满足问题

考虑变量之间的约束关系，利用这种关系可以进行剪枝，最后迅速消除大规模的搜索空间。

### 5.1 约束的定义

约束满足问题的三元组 $(X, D, C)$ :  $X = \{X_1, X_2, \dots, X_n\}$ 为变量集;  $D^{(i)} = \{D_1^{(i)}, D_2^{(i)}, \dots, D_d^{(i)}\}$ 为值域集,  $i = 1, 2, \dots, n$ ;  $C = < scope, rel >$ ,  $scope$ 为变量组而 $rel$ 定义变量之间的关系。

地图染色问题中,  $X$ 表示所有的待染色区域;  $D_i = \{\text{red}, \text{green}, \text{blue}\}$ 为取值情况; 约束关系为 $\{< (X_i, X_j), D^{(i)} \neq D^{(j)} \text{ if } X_i \in N(X_j) >\}$

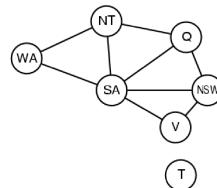
**离散变量**, 在有限区域中, 若有  $n$  个变量, 值域大小为  $d$ , 则共有  $O(n^d)$  种赋值方式。无限值域下(整数, 字符串), 需要用约束语言描述, 且线性约束可解, 非线性解不可知。

**连续变量**, 线性约束在多项式时间内可解。

**软约束**: 如红色比绿色好, 可以额外定义个体变量赋值的 cost。

**相容的**: 一个不违反任何约束条件的赋值。 **完整的**: 每个变量都已经赋值。 **约束问题的解**: 相容的, 完整的。

约束图为以结点为变量, 弧为约束关系的图, 利用图结构可以划分独立的子问题。



### 5.2 约束问题形式化



Variables  $WA, NT, Q, NSW, V, SA, T$

Domains  $D_i = \{\text{red}, \text{green}, \text{blue}\}$

Constraints: adjacent regions must have different colors

e.g.,  $WA \neq NT$ , or (if the language allows this), or  
 $(WA, NT) \in \{(red, green), (red, blue), (green, red), (green, blue), \dots\}$

$\begin{array}{r} T \quad W \quad O \\ + \quad T \quad W \quad O \\ \hline F \quad O \quad U \quad R \end{array}$   
 Variables:  $F \ T \ U \ W \ R \ O, X_1, X_2, X_3$   
 Domains:  $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$   
 Constraints:  
 $\text{alldiff}(F, T, U, W, R, O)$   
 $O + O = R + 10 \cdot X_1$   
 $X_1 + W + W = U + 10 \cdot X_2$   
 $X_2 + T + T = O + 10 \cdot X_3$   
 $X_3 = F, T \neq 0, F \neq 0$

图 12: Map-Coloring

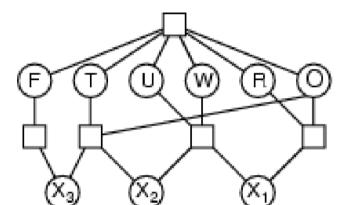


图 13: 密码算数

约束的种类: **一元约束** 限制变量本身的取值。**二元约束** 只与两个变量相关。**全局约束** 变量个数为任意值, e.g. 9宫格。任意有限值域的约束都可以通过引入足够的约束变量而转化为二元约束。可以采用对偶图的方式。

### 5.3 约束的增量形式化

初始状态: 空的赋值, { }

行动: 为未赋值且不与当前赋值冲突的变量赋值, 如果没有合法的赋值, 则失败。

目标测试: 当前的赋值是否完备。

每一个解所在的深度都是  $n$ , 可以使用 **DFS!** 时间复杂度为  $O(d^n)$ .

也可以用完整状态形式化, 但深度为 1 时叶子数目为  $b = (n - l)d$ , 因此共  $n!d^n$  个叶子。

### 5.4 约束相容性

结点相容 所有的结点均满足一元约束, 即取值范围限制。

弧相容 ( $X, Y$ )是一致的  $\Leftrightarrow$  对于  $X$  的每一个值  $x$  都有一些允许的  $y$ .

```

function AC-3(csp) returns the CSP, possibly with reduced domains
  inputs: csp, a binary CSP with variables  $\{X_1, X_2, \dots, X_n\}$ 
  local variables: queue, a queue of arcs, initially all the arcs in csp
  while queue is not empty do
     $(X_i, X_j) \leftarrow \text{REMOVE-FIRST}(\text{queue})$ 
    if REMOVE-INCONSISTENT-VALUES( $X_i, X_j$ ) then
      for each  $X_k$  in NEIGHBORS[ $X_i$ ] do
        add  $(X_k, X_i)$  to queue

function REMOVE-INCONSISTENT-VALUES( $X_i, X_j$ ) returns true iff succeeds
  removed  $\leftarrow$  false
  for each  $x$  in DOMAIN[ $X_i$ ] do
    if no value  $y$  in DOMAIN[ $X_j$ ] allows  $(x, y)$  to satisfy the constraint  $X_i \leftrightarrow X_j$ 
      then delete  $x$  from DOMAIN[ $X_i$ ]; removed  $\leftarrow$  true
  return removed

```

$\mathcal{O}(n^2d^3)$  (but detecting all is NP-hard)

在队列中弹出  $(X_i, X_j)$ ,  $X_i$  的值域中每个值与  $X_j$  相容, 若存在不相容的值则移除, 并将与  $X_i$  相关的弧重新加入到队列中。实质上为以相容性为标准进行的缩小状态空间的过程。

**路径相容** 多个变量的取值集合  $S$  相对于  $X_j$  的取值是相容的  $\Leftrightarrow$  对于  $S$  中的每一个子集,  $X_j$  都有相应的值使得  $(S, X_j)$  是相容的。

### 5.5 CSP 回溯搜索

变量赋值的时候不考虑顺序, 故CSP是可交换的。因此可以采用回溯算法依次给结点赋值, 一旦遇到失败, 则回溯到上一个结点继续尝试下一个值。

```

function BACKTRACKING-SEARCH(csp) returns solution/failure
    return RECURSIVE-BACKTRACKING({ }, csp)
function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
    if assignment is complete then return assignment
    var  $\leftarrow$  SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment given CONSTRAINTS[csp] then
            add {var = value} to assignment
            result  $\leftarrow$  RECURSIVE-BACKTRACKING(assignment, csp)
            if result  $\neq$  failure then return result
            remove {var = value} from assignment
    return failure

```

## 5.6 回溯优化

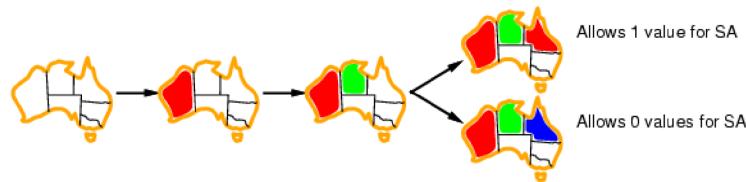
### 5.6.1 变量顺序

可以采用最小剩余值(**MRV**)启发式，即先对剩余值最少的(最受约束的)变量赋值，原理为“Fail-fast” ordering；若未被赋值的变量剩余值数目相同，可换用度启发式，即与该变量关联的约束数目越多，越早被赋值。

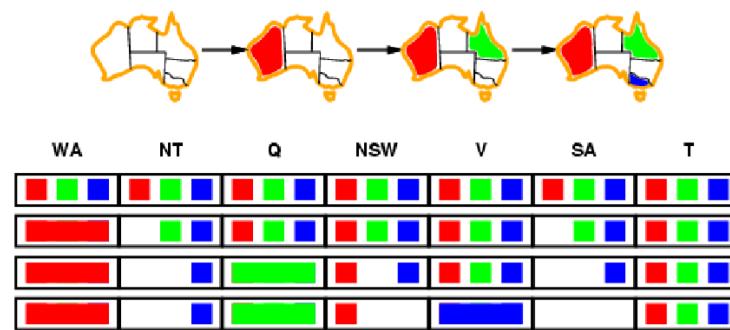
### 5.6.2 取值顺序

采用最小约束值启发式，使得给邻居变量留下更多的选择。

△ 使用MRV是为了减小搜索空间，对于取值顺序，应该选择最有可能的值才有意义。

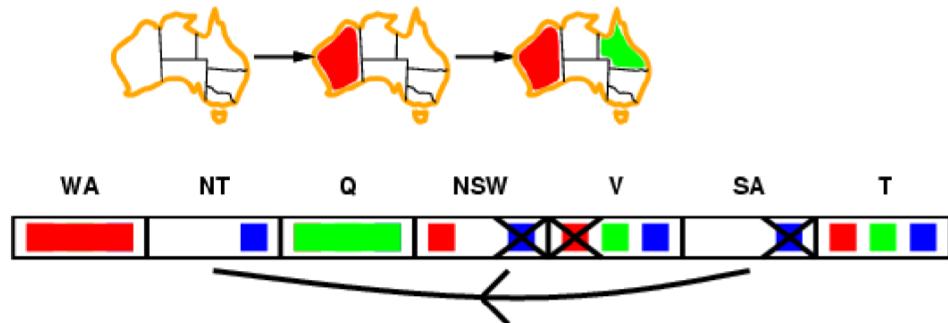


### 5.6.3 失败预警



**前向检验** 向前检验记录未分配变量的剩余合法值，当任何变量没有合法值时终止搜索(需要额外的计算)。  
向前检验向未分配值的变量传播关于已分配值的变量的信息，但不能对所有的失败预警。

**弧相容** 如果 X 丢失了一个值，则需要重新检查 X 的邻居。弧相容检测失败早于前向检验。可以对原约束进行预处理，或者在每次为某个变量赋值后运行。



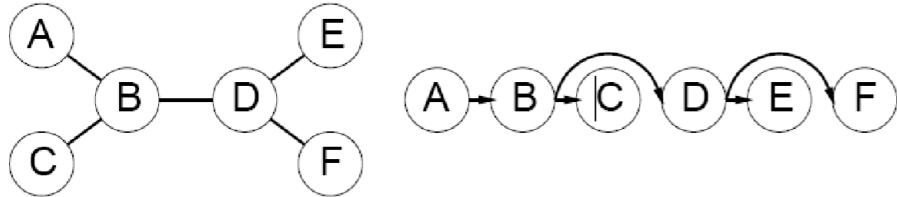
注意到由于只对赋值的变量作弧相容测试，(向前检验+弧相容)仍不能检测出所有的不相容。  
结合以上两种顺序和 AC-3 算法，可以综合对全变量及其值域空间作向前检验。

#### 5.6.4 利用约束结构

- 由约束图的结构，分解为多个独立连通子图。若每个子问题有 c 个变量，一共 n 个变量。则时间复杂度为  $O(\frac{n}{c} d^c) = O(n)$ .
- Theorem3:** 任何一个树状的 CSP 问题可以在  $O(nd^2)$  时间解决。

**Proof** 按照拓扑排序的顺序进行赋值，时间复杂度为  $O(n) * O(d^2) = O(nd^2)$ .

1. Choose a variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering



2. For  $j$  from  $n$  down to 2, apply REMOVEINCONSISTENT( $\text{Parent}(X_j), X_j$ )

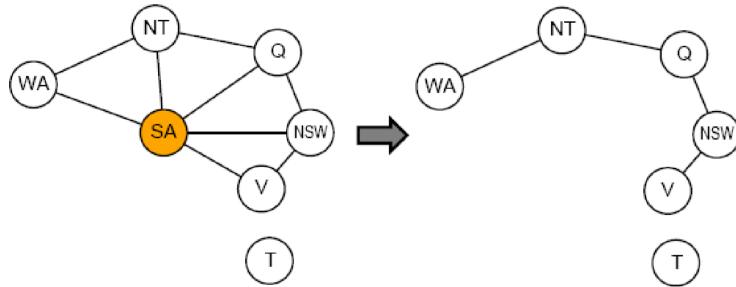
3. For  $j$  from 1 to  $n$ , assign  $X_j$  consistently with  $\text{Parent}(X_j)$

Complexity:  $O(n \cdot d^2)$

3. 作割集调整，选择最小割集 S 使得  $G/S$  为一棵树，对  $S$  与  $G/S$  在满足约束下分别地、独立地赋值，若各自有解且对于  $S$  的一组的解，存在在  $G/S$  中的解使得二者相容，则成功找到解。

简单地，可以实例化一个变量，并对邻居的值进行剪枝，在剩下的树结构上求解约束满足问题。

割集大小为 c 时，时间复杂度为  $O(d^c(n - c)d^2)$ ，c 小时效率很高。



## 5.7 局部搜索

采用局部搜索的思想解决 CSP 问题，启发式采用最小冲突启发式(hillclimb with  $h(n) = \text{total number of violated constraints}$ )。对于完整状态形式化问题，允许冲突的存在，每次迭代随机选取一个冲突的变量，并为它选择会造成与其他变量冲突最小的值。重复此过程，直到找到问题的解或达到 $max\ steps$ 。

在局部搜索中采用约束加权技术(minmax optimization)，即每个约束都有一个权重，随着当前状态的不同，使状态违反约束权重和最小，直到不能再减小；之后改变权重值来增大状态违反约束的权重和，直到最小化可以继续进行，重复此过程。

### Speedup 1: simulated annealing

Idea: escape local maxima by allowing some "bad" moves  
but **gradually decrease** their frequency

If 新状态比现有状态好，移动到新状态  
Else 以某个小于1的概率接受该移动  
□ 此概率随温度 “T” 降低而下降

### Speedup 2: minmax optimization

Put weights on constraints  
repeat  
    Primal search: update assignment to minimize weighted violation,  
                        until stuck  
    Dual step:     update weights to increase weighted violation,  
                        until unstuck  
until solution found, or bored

总结：约束满足问题的状态由一系列变量决定，目标测试是由变量间的约束关系来定义的  
回溯搜索 = 每次为一个变量赋值的深度优先搜索( $n$ 个变量对应最多  $n$ 层)  
变量顺序可以降低搜索空间，值顺序能够使算法尽快找到可行的解  
弧相容算法能帮助前向检验算法尽早否定不可能的解  
图约束能从拓扑结构角度简化问题，树形约束能在线性时间解决

## 6 Chapter 5: 博弈

竞争环境中，多 Agents 存在目标冲突，称之为“博弈”。指在有完整信息的、确定的、轮流行动的两个 Agents 的零和博弈，在游戏结束时效用值总是相等且符号方向相反。

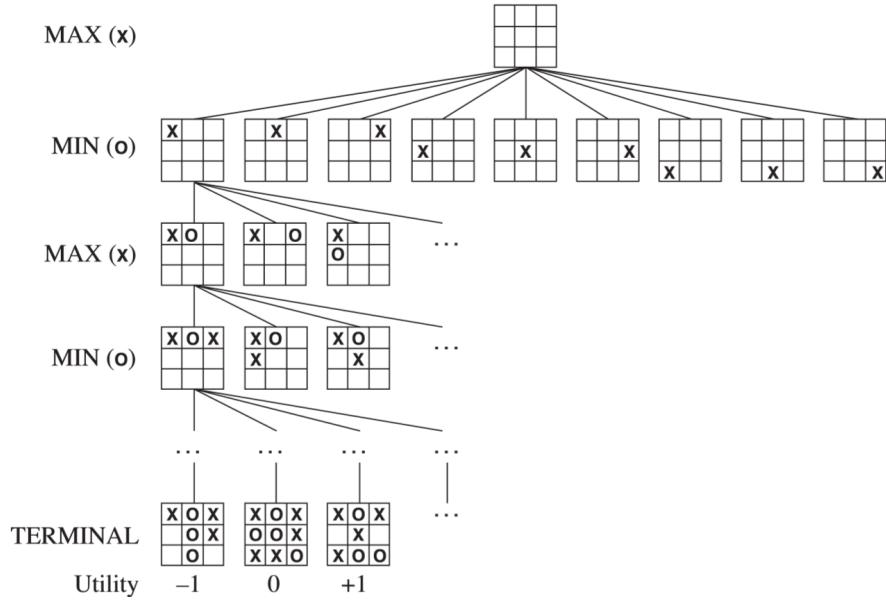
### 6.1 博弈定义

游戏形式化：

- $S_0$ , 初始状态

- PLAYER(s), 此轮的行动者
- ACTIONS(s), 此轮下的合法行动
- RESULT(s, a), 转移模型, 即行动的结果
- TERMINAL-TEST(s), 布尔类型, 测试游戏是否终止
- UTILITY(s, p), 定义player  $p$ 在终止状态  $s$ 下的效用值

由  $S_0$ , ACTIONS(s), RESULT(s, a) 定义了一棵博弈树



## 6.2 博弈优化

博弈中的最优解: MAX 针对 MIN 的行动, 并作出应急策略, 以此达到最优的末状态。

前提: MIN 也是有智能的。

策略: 假设 MIN 每次移动到极小值的状态, MAX 往极小极大值(极小中挑极大)方向行棋。

算法: 采用递归回溯的完整深度优先搜索, 基于结点类型(Terminal, Max, Min), 分别将(Utility, max MINIMAX, min MINIMAX)回传。此处只需将 MINIMAX 看成是一个结点属性, Max 与 Min 的区别在于对后继的属性取最大或最小。

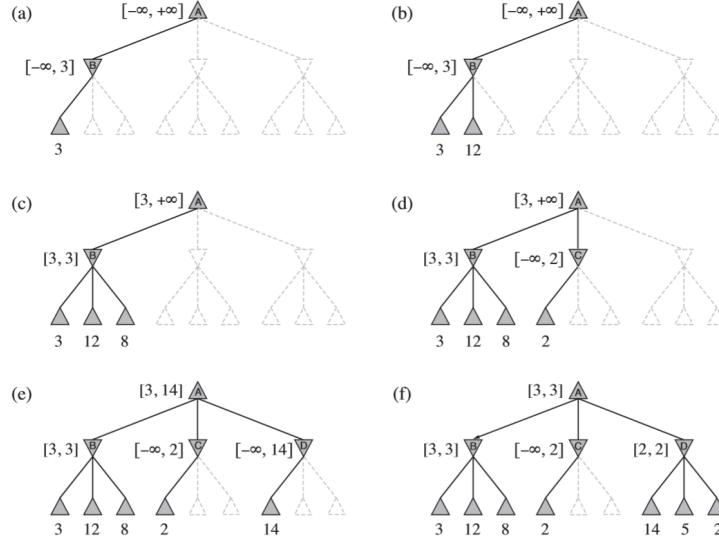
复杂度: 时间  $O(b^m)$ , 空间  $O(bm)$ .

优点: 完备的(在有限的树下), 最优的(对抗者也是最优的, 否则不最优)。

缺点: 游戏状态的数目随着博弈的进行呈指数级增长。

$$\text{MINMAX-VALUE}(n) = \begin{cases} \text{UTILITY}(n) & \text{当 } n \text{ 为终止状态} \\ \max_{s \in \text{Successors}(n)} \text{MINMAX-VALUE}(s) & \text{当 } n \text{ 为 MAX 节点} \\ \min_{s \in \text{Successors}(n)} \text{MINMAX-VALUE}(s) & \text{当 } n \text{ 为 MIN 节点} \end{cases}$$

不影响最终决策分支, 用剪枝尽可能消除极大极小搜索树上的部分决策树, 是对MINIMAX公式的优化, 即



$$\text{MINIMX}(\text{root}) = \max(\min(3, 12, 8), \min(2, x, y), \min(14, 5, 2)) = \max(3, z(\leq 2), 2) = 3$$

用  $\alpha$  记录所有 MAX 结点的最大值； $\beta$  记录所有 MIN 结点的最小值。当遇到某个结点  $\text{MAX}(\text{MIN})$  比当前的  $\beta(\alpha)$  还要大(小时)(套出的区间为空)，停止递归调用。

---

```

function ALPHA-BETA-SEARCH(state) returns an action
  v  $\leftarrow$  MAX-VALUE(state,  $-\infty, +\infty$ )
  return the action in ACTIONS(state) with value v

function MAX-VALUE(state,  $\alpha, \beta$ ) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v  $\leftarrow -\infty$ 
  for each a in ACTIONS(state) do
    v  $\leftarrow$  MAX(v, MIN-VALUE(RESULT(s,a),  $\alpha, \beta$ ))
    if v  $\geq \beta$  then return v
     $\alpha \leftarrow \text{MAX}(\alpha, v)$ 
  return v

function MIN-VALUE(state,  $\alpha, \beta$ ) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v  $\leftarrow +\infty$ 
  for each a in ACTIONS(state) do
    v  $\leftarrow$  MIN(v, MAX-VALUE(RESULT(s,a),  $\alpha, \beta$ ))
    if v  $\leq \alpha$  then return v
     $\beta \leftarrow \text{MIN}(\beta, v)$ 
  return v

```

---

$\alpha - \beta$  剪枝的时间复杂度为  $O(b^{m/2})$ ，有效分支因子为  $\sqrt{b}$ ，即若招法排序最优（始终优先搜索最佳招法），则需要评估的最大叶节点数目按层数奇偶性，分别约为  $O(b^1 * b^1 * b^1 * \dots * b)$ 。其意义为，对第一名玩家必须搜索全部招法找到最佳招式，但对于它们，只用将第二名玩家的最佳招法截断—— $\alpha - \beta$  确保无需考虑第二名玩家的其他招法。

### 6.3 截断

实际中仍然无法探索整棵博弈树，因此对  $\alpha - \beta$  算法进行修改，用启发式评估函数 EVAL 代替 Utility function。即在限制最大深度  $d$  下，将每个结点 (Terminal, CUT-OFF, MAX, MIN) 的属性改为 (Utility, EVAL, max H-MINIMAX, min H-

MINIMAX).

评估函数设计 考虑不同的特征参数，状态的合理评价是期望值，一般也可用加权线性函数

$$EVAL(s) = \sum_{i=1}^n \omega_i f_i(s)$$

其中基于特征独立的假设，确保更重要的特征有更大的权重，也可以采用非线性的特征组合。

EVAL 具体的数值不会改变决策的结果，重要的是其评估的次序。

**Example:** for chess,  $w_1 = 9$  with  $f_1(s) = (\text{number of white queens}) - (\text{number of black queens})$ .

实际中，为了保证在规定的时间内找到解，采用 IDS 的思想，随着深度限制的增加，运行 alpha-beta 搜索，当时间耗尽时，使用上次完成的 alpha-beta 搜索(即完成的最深的搜索)找到的解决方案。

## 6.4 机会博弈

Agents MAX在行动后，存在一定的几率结点，几率结点有一定的概率将目前的状态  $S$  改变为  $S_0, S_1$ ，之后再是MIN作决策。这时在博弈树中，将期望值作为几率结点的属性。

期望效用最大化原则:在已知条件下，Agents 应选择其期望效用最大化的行为

EXPECTIMINIMAX gives perfect play

Just like MINIMAX, except we must also handle chance nodes:

```
...
if state is a Max node then
    return the highest EXPECTIMINIMAX-VALUE of SUCCESSORS(state)
if state is a Min node then
    return the lowest EXPECTIMINIMAX-VALUE of SUCCESSORS(state)
if state is a chance node then
    return average of EXPECTIMINIMAX-VALUE of SUCCESSORS(state)
...
```

因为涉及到计算期望，因此 EVAL 具体的数值会改变决策的结果。

## 6.5 部分可观察博弈(不考)

在评价一个有未知牌的给定行动过程时，首先计算每副可能牌的出牌行动的极小极大值，然后用每副牌的概率来计算得到对所有发牌情况的期望值。注意在这种情形下，行动的价值是它在所有可能状态的平均是错的，还要考虑 Agent 所在的信度状态。这会导致 Agents 尝试去获取更多的信息，向 partner 发出信息以及减少自身的泄露，

总结：

MINIMAX 通过假设对手总是选择最好的一步来选择“最优”的一步

$\alpha - \beta$  可以避免搜索树的大部分内容，从而使搜索更加深入

$\alpha - \beta$  剪枝不改变游戏的结果

机会博弈只能使用期望效用，这使剪枝变得更困难

## 7 Chapter 7: 逻辑Agent

**State-based Models:** search problems, games

– Applications: routing finding, game playing, etc.

– Think in terms of **States, Actions, and Costs**

**Variable-based Models:** CSPs, Bayesian networks

– Applications: scheduling, medical diagnosis, etc.

– Think in terms of **Variables and Potentials**

**Logic-based models:** propositional logic, first-order logic

– Applications: theorem proving, verification, reasoning

– Think in terms of **Logical Formulas and Inference Rules**

问题求解 Agent 有局限性并且缺乏灵活性，下一步行动是预先编写入代码的，而无法应用推理。逻辑 Agent 建立在知识库  $KB$  之上的，通过 TELL 和 ASK 与  $KB$  进行交互。

### 7.1 基本概念与 Wumpus World 案例

- 语法: 定义语句; 语义: 定义语句在每个可能世界的真值
- 模型  $M(\alpha)$ : 所有使得  $\alpha$  为真的可能的世界，“ $m$  是  $\alpha$  的一个模型” 表示语句  $\alpha$  在模型  $m$  中为真。
- 蕴涵 entail:  $\alpha \models \beta \iff M(\alpha) \subseteq M(\beta)$ .

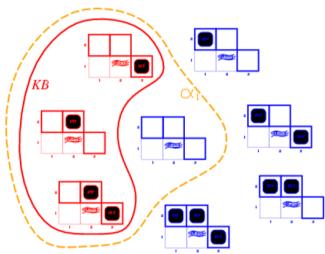


图 14: KB 蕴涵  $\alpha_1$

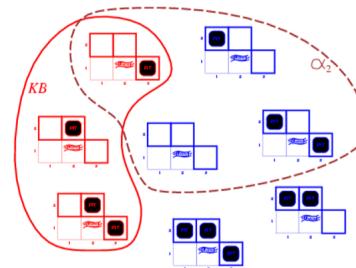


图 15: KB 不蕴涵  $\alpha_2$

- 模型检验: 枚举所有可能的  $M$  来检验  $KB$  与  $\alpha$  同为真，即  $M(KB) \subseteq M(\alpha)$ .

- 知识库  $KB =$  规范语句的集合

```
function KB-AGENT(percept) returns an action
    static: KB, a knowledge base
        t, a counter, initially 0, indicating time
    TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
    action  $\leftarrow$  ASK(KB, MAKE-ACTION-QUERY(t))
    TELL(KB, MAKE-ACTION-SENTENCE(action, t))
    t  $\leftarrow$  t + 1
    return action
```

TELL 与 ASK 需要应用推理机制。

- 推理算法  $i: KB \vdash_i \alpha$ : 推理算法  $i$  能根据  $KB$  导出  $\alpha$ , “ $i$  从  $KB$  导出  $\alpha$ ”

可靠性: 只导出语义蕴涵句, whenever  $KB \vdash_i \alpha, KB \vdash \alpha$ 。

完备性: 可以生成任一蕴涵句, whenever  $KB \vdash \alpha, KB \vdash_i \alpha$ 。

**KB:** 干草堆,  $\alpha$ : 针, 蕴涵: 针在干草堆中, 推理: 从干草堆中找出针。

#### Performance measure

gold +1000, death -1000  
-1 per step, -10 for using the arrow

#### Environment

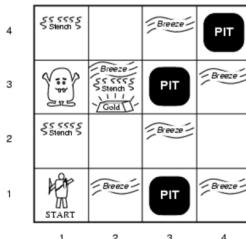
4×4网格

智能体初始在[1,1], 面向右方  
金子和wumpus在[1,1]之外随机均匀分布  
[1,1]之外的任意方格是陷阱的概率是0.2

#### Actuators Left turn, Right turn,

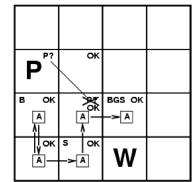
Forward, Grab, Shoot

- 智能体可向前、左转或右转
- 智能体如果进入一个有陷阱或者活着的wumpus的方格, 将死去。
- 如果智能体前方有一堵墙, 那么向前移动无效
- Grab: 捡起智能体所在方格中的一个物体
- Shoot: 向智能体所正对方向射箭 (只有一枝箭)



#### Sensors

- Smell: 在wumpus所在之处以及与之直接相邻的方格内, 智能体可以感知到臭气。
- Breeze: 在与陷阱直接相邻的方格内, 智能体可以感知到微风。
- Glitter(发光): 在金子所处的方格内, 智能体可以感知到闪闪金光。
- 当智能体撞到墙时, 它感受到撞击。
- 当wumpus被杀死时, 它发出洞穴内任何地方都可感知到的悲惨嚎叫。



以5个符号的列表形式将感知信息提供给智能体,  
例如(stench, breeze, none, none, none)。

特点: 非可观测的, 确定性的, 非原子的, 静态的, 离散的, 单智能体的。

## 7.2 命题逻辑

**BNF语法** 原子语句、复合句; 否定式、合取式、析取式、蕴含式、双向蕴含式。

$\text{Sentence} \rightarrow \text{AtomicSentence} \mid \text{ComplexSentence}$ $\text{AtomicSentence} \rightarrow \text{True} \mid \text{False} \mid P \mid Q \mid R \mid \dots$ $\text{ComplexSentence} \rightarrow (\text{Sentence}) \mid [\text{Sentence}]$ $\quad \mid \neg \text{Sentence}$ $\quad \mid \text{Sentence} \wedge \text{Sentence}$ $\quad \mid \text{Sentence} \vee \text{Sentence}$ $\quad \mid \text{Sentence} \Rightarrow \text{Sentence}$ $\quad \mid \text{Sentence} \Leftrightarrow \text{Sentence}$  <b>OPERATOR PRECEDENCE</b> : $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$
---

**Figure 7.7** A BNF (Backus–Naur Form) grammar of sentences in propositional logic, along with operator precedences, from highest to lowest.

$P$	$Q$	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	false	true	false	false	true	true
false	true	true	false	true	true	false
true	false	false	false	true	false	false
true	true	false	true	true	true	true

**Figure 7.8** Truth tables for the five logical connectives. To use the table to compute, for example, the value of  $P \vee Q$  when  $P$  is true and  $Q$  is false, first look on the left for the row where  $P$  is true and  $Q$  is false (the third row). Then look in that row under the  $P \vee Q$  column to see the result: true.

```

function TT-ENTAILS?(KB,  $\alpha$ ) returns true or false
  inputs: KB, the knowledge base, a sentence in propositional logic
   $\alpha$ , the query, a sentence in propositional logic
  symbols  $\leftarrow$  a list of the proposition symbols in KB and  $\alpha$ 
  return TT-CHECK-ALL(KB,  $\alpha$ , symbols, { })


---


function TT-CHECK-ALL(KB,  $\alpha$ , symbols, model) returns true or false
  if EMPTY?(symbols) then
    if PL-TRUE?(KB, model) then return PL-TRUE?( $\alpha$ , model)
    else return true // when KB is false, always return true
  else do
    P  $\leftarrow$  FIRST(symbols)
    rest  $\leftarrow$  REST(symbols)
    return (TT-CHECK-ALL(KB,  $\alpha$ , rest, model  $\cup$  {P = true})
           and
           TT-CHECK-ALL(KB,  $\alpha$ , rest, model  $\cup$  {P = false}))

```

**Figure 7.10** A truth-table enumeration algorithm for deciding propositional entailment. (TT stands for truth table.) PL-TRUE? returns true if a sentence holds within a model. The variable *model* represents a partial model—an assignment to some of the symbols. The keyword “**and**” is used here as a logical operation on its two arguments, returning true or false.

图 16: 枚举法判断是否蕴涵, 时间复杂度  $O(2^n)$ , 空间复杂度  $O(n)$ .

逻辑等价性  $\alpha \equiv \beta$ :  $\alpha, \beta$  为真的模型集合相同。

$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$	commutativity of $\wedge$	$\wedge$ 的可交换性
$(\alpha \vee \beta) \equiv (\beta \vee \alpha)$	commutativity of $\vee$	$\vee$ 的可交换性
$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$	associativity of $\wedge$	$\wedge$ 的结合律
$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$	associativity of $\vee$	$\vee$ 的结合律
$\neg(\neg\alpha) \equiv \alpha$	double-negation elimination	双重否定消去
$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha)$	contraposition	逆否命题
$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta)$	implication elimination	蕴涵消去
$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$	biconditional elimination	双向蕴涵消去
$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$	De Morgan	摩根律
$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$	De Morgan	摩根律
$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$	distributivity of $\wedge$ over $\vee$	$\wedge$ 对 $\vee$ 的分配率
$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$	distributivity of $\vee$ over $\wedge$	$\vee$ 对 $\wedge$ 的分配率

有效性(合法性):  $\alpha$  在所有的模型中均为真。

可满足性: 语句  $\alpha$  在部分模型中为真。

不可满足性: 在所有的模型中均不为真, e.g. 反证法, 假定  $\alpha$  为假, 推出与 KB 矛盾。

推理规则: 假言推理规则, 消去合取词规则, 归结规则。(P210)

### 7.3 前向链接

霍恩子句 = 命题符号 或 [命题符号的连接](一定要是正文字!!)  $\Rightarrow$  命题。KB 为霍恩子句的与构成的语句集.

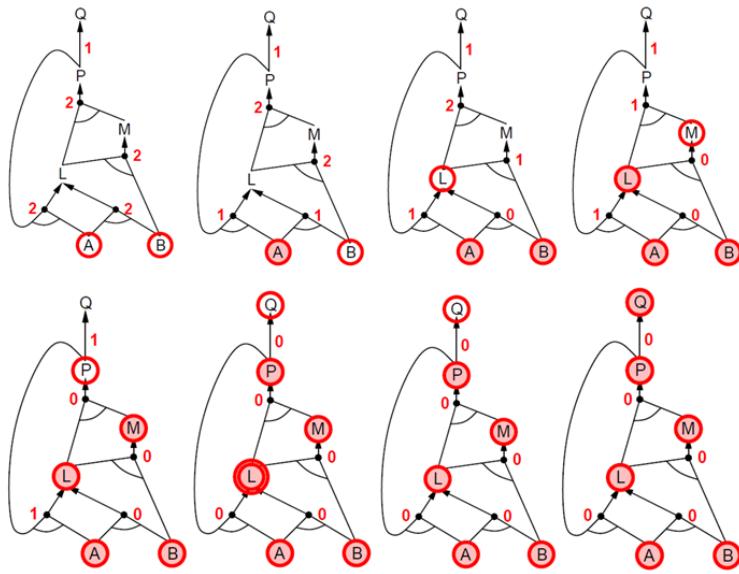
假言推理

$$\frac{\alpha_1, \dots, \alpha_n, \quad \alpha_1 \wedge \dots \wedge \alpha_n \Rightarrow \beta}{\beta}$$

对于霍恩子句是完备的, 可用于前向链接或后向链接。

想法: 如果规则的前提能被 KB 满足, 将其结论加到 KB, 直到得出所询问的命题。

复杂度:  $O(m)$



对于霍恩子句，前向链接算法是可靠(每个规则本质上是分离规则的应用)且完备的(每个被蕴涵的原子语句都将被生成)。

### 完备性 Proof

1. FC到达不动点以后，不可能再出现新的推理。
2. 考察 inferred 表的最终状态，参与推理过程的每个符号为 true，其它为 false。把该推理表看做一个逻辑模型  $m$
3. 原始 KB 中的每个确定子句在该模型  $m$  中都为真  
证明: 假设某个子句  $a_1 \wedge \dots \wedge a_k \Rightarrow b$  在  $m$  中为 false  
那么  $a_1 \wedge \dots \wedge a_k$  在  $m$  中为 true,  
 $b$  在  $m$  中为 false 与算法已经到达一个不动点相矛盾  
(前提条件都是真，这个子句又在模型里，那么结论也是真，说明  $b$  需要被更新)
4.  $m$  是 KB 的一个模型  
如果  $KB \models q$ ,  $q$  在 KB 的所有模型中必须为真，包括  $m$
6.  $q$  在  $m$  中为真  $\Rightarrow$  在 inferred 表中为真  $\Rightarrow$  被 FC 算法推断出来

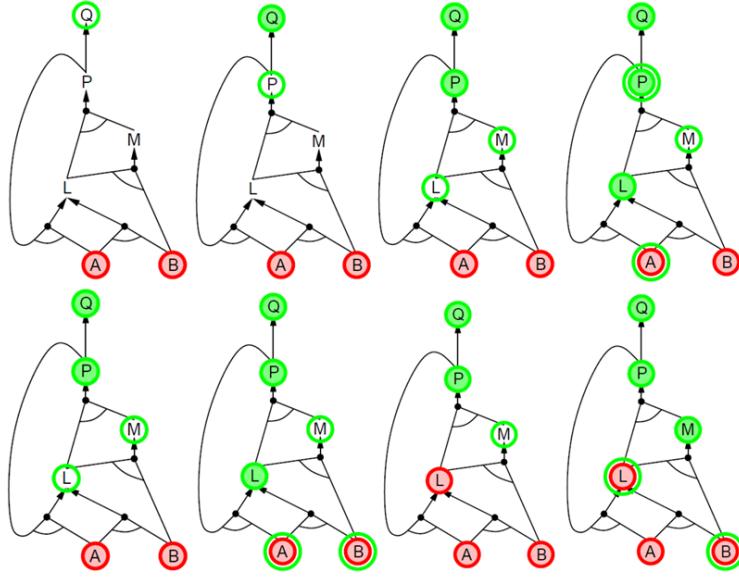
## 7.4 后向链接

想法: 从查询点  $q$  反向进行，检查  $q$  是否为真，或者寻找以  $q$  为结论的蕴涵，证明其中一个蕴涵的前提条件为真。

复杂度: 低于前向链接，比  $O(m)$  小很多。

避免循环: 检查新的子目标是否已经在目标堆栈中。

避免重复工作: 检查新的子目标 1)是否已经被证明是正确的，或者 2)是否已经失败.



## 7.5 归结算法

适用于任何的子句，需要先转化为合取范式。

对于合取范式，E.g.,  $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$ ，可以利用归结规则消去互补文字。

想法：

$$\frac{l_i \vee \dots \vee l_k, \quad m_1 \vee \dots \vee m_n}{l_i \vee \dots \vee l_{i-1} \vee l_{i+1} \vee \dots \vee l_k \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n}$$

图 17:  $l_i$  与  $m_i$  为互补文字

同时，利用反证法

$$KB| = \alpha \Leftrightarrow KB \wedge \alpha \text{ 不满足}$$

复杂度： $O(e^n)$ ，每次应用规则都添加了带有许多命题符号的子句。

优点：命题逻辑中，归结是可靠和完备的(P214)。

**step1：将语句转化为合取范式，Example**

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

1. Eliminate  $\Leftrightarrow$ , replacing  $\alpha \Leftrightarrow \beta$  with  $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$ .

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

2. Eliminate  $\Rightarrow$ , replacing  $\alpha \Rightarrow \beta$  with  $\neg \alpha \vee \beta$ ..

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

3. Move  $\neg$  inwards using de Morgan's rules and double-negation:

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

4. Apply distributivity law ( $\vee$  over  $\wedge$ ) and flatten:

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

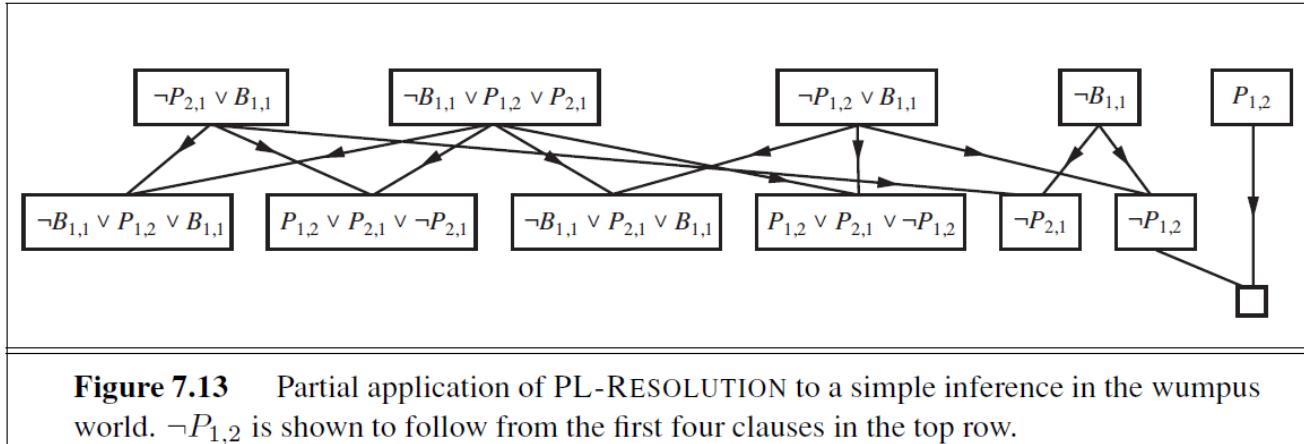
**step2** : 多次使用归结算法

**step3** : 返回不满足 *iff* 归结失败

```

function PL-RESOLUTION(KB, α) returns true or false
    inputs: KB, the knowledge base, a sentence in propositional logic
            α, the query, a sentence in propositional logic
    clauses  $\leftarrow$  the set of clauses in the CNF representation of KB  $\wedge \neg\alpha$ 
    new  $\leftarrow \{ \}$ 
    loop do
        for each Ci, Cj in clauses do
            resolvents  $\leftarrow$  PL-RESOLVE(Ci, Cj)
            if resolvents contains the empty clause then return true
            new  $\leftarrow$  new  $\cup$  resolvents
        if new  $\subseteq$  clauses then return false
        clauses  $\leftarrow$  clauses  $\cup$  new

```



**Figure 7.13** Partial application of PL-RESOLUTION to a simple inference in the wumpus world.  $\neg P_{1,2}$  is shown to follow from the first four clauses in the top row.

## 7.6 优缺点

Pros (优点) of propositional logic	Cons (缺点) of propositional logic
<span style="color: green;">😊</span> Propositional logic is <b>declarative</b> (陈述性的) <ul style="list-style-type: none"> <li>□ 知识和推理分开，而且推理完全不依赖于领域</li> <li>□ 对比：程序设计语言——过程性语言               <ul style="list-style-type: none"> <li>■ 缺乏从其它事实派生出事实的通用机制</li> <li>■ 对数据结构的更新通过一个领域特定的过程来完成</li> </ul> </li> </ul>	<span style="color: green;">😊</span> Propositional logic has very limited expressive power <ul style="list-style-type: none"> <li>□ (unlike natural language)</li> <li>□ E.g., cannot say "pits cause breezes in adjacent squares" except by writing one sentence for each square               <math display="block">B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})</math> </li> </ul>
<span style="color: green;">😊</span> Propositional logic allows partial (不完全) /disjunctive (分离的) /negated information <ul style="list-style-type: none"> <li>□ (unlike most data structures and databases)</li> </ul>	
<span style="color: green;">😊</span> Propositional logic is <b>compositional</b> (合成性的) : <ul style="list-style-type: none"> <li>□ meaning of <math>B_{1,1} \wedge P_{1,2}</math> is derived from meaning of <math>B_{1,1}</math> and of <math>P_{1,2}</math> (语句的含义是它的各部分含义的一个函数)</li> </ul>	
<span style="color: green;">😊</span> Meaning in propositional logic is <b>context-independent</b> <ul style="list-style-type: none"> <li>□ (unlike natural language, where meaning depends on context)</li> </ul>	

## 8 Chapter 8: 一阶逻辑

采用对象, 关系, 函数, 并借用自然语言的思想, 表达能力强。

基本元素:

Constants/常量	KingJohn, 2, USTC, ...
Predicates/谓词	Brother, >, ...
Functions/函数	Sqrt, LeftLegOf, ...
Variables/变量	x, y, a, b, ...
Connectives/连接词	$\neg, \Rightarrow, \wedge, \vee, \Leftrightarrow$
Equality/等词	=
Quantifiers/量词	$\forall, \exists$

由于元素的多样性, 通过枚举来检验模型是否蕴涵某语句是不可行的。

项: function(项)、常量、变量

原子语句: predicate( $term_1, \dots, term_n$ ) 或等式  $term_1 = term_2$ .

复合语句: 原子语句与连接词组合使用。

量词:  $\forall$  与  $\Rightarrow$  搭配使用;  $\exists$  与  $\wedge$  搭配使用。

使用量词 Example:

$\forall x At(x, USTC) \wedge Smart(x)$   
 means "Everyone is at USTC and everyone is smart"  
 $\exists x At(x, USTC) \Rightarrow Smart(x)$   
 is true if there is anyone who is not at USTC!

等式: 若指代的对象相同则为真。

使用等式 Example: definition of Sibling in terms of Parent:

$$\begin{aligned} \forall x, y \text{ Sibling}(x, y) \Leftrightarrow & [\neg(x = y) \wedge \exists m, f \neg(m = f) \wedge \text{Parent}(m, x) \\ & \wedge \text{Parent}(f, x) \wedge \text{Parent}(m, y) \wedge \text{Parent}(f, y)] \end{aligned}$$

## 一阶逻辑规则 Example:

### The kinship domain:

Brothers are siblings

$$\forall x, y \text{ Brother}(x, y) \Rightarrow \text{sib ling}(x, y)$$

"Sibling" is symmetric

$$\forall x, y \text{ Sibling}(x, y) \Leftrightarrow \text{ sibling}(y, x)$$

One's mother is one's female parent

$$\forall x, y \text{ Mother}(x, y) \Leftrightarrow (\text{ Female}(x) \wedge \text{ Parent}(x, y))$$

A cousin is a child of a parent's sibling

$$\forall x, y \text{ cousin}(x, y) \Leftrightarrow \exists p, ps \text{ Parent}(p, x) \wedge \text{Sibling}(ps, p) \wedge \text{Parent}(ps, y)$$

### The set (集合) domain:

集合就是空集或通过将一些元素添加到一个集合而构成  $\forall s \text{ Set}(s) \Leftrightarrow (s = \{\}) \vee (\exists x, s_2 \text{ Set}(s_2) \wedge s = \{x \mid s_2\})$

空集没有任何元素，也就是说，空集无法再分解为更小的集合和元素

$$\neg \exists x, s \quad \{x \mid s\} = \{\}$$

将已经存在于集合中的元素添加到该集合，无任何变化

$$\forall x, s \quad x \in s \Leftrightarrow s = \{x \mid s\}$$

集合的元素仅是那些被添加到集合中的元素

$$\forall x, s \quad x \in s \Leftrightarrow [\exists y, s_2 (s = \{y \mid s_2\} \wedge (x = y \vee x \in s_2))]$$

一个集合是另一个集合的子集，当且仅当第一个集合的所有元素都是第二个集合的元素

$$\forall s_1, s_2 \quad s_1 \subseteq s_2 \Leftrightarrow (\forall x \quad x \in s_1 \Rightarrow x \in s_2)$$

两个集合是相同的，当且仅当它们互为子集

$$\forall s_1, s_2 \quad (s_1 = s_2) \Leftrightarrow (s_1 \subseteq s_2 \wedge s_2 \subseteq s_1)$$

一个对象是两个集合的交集的元素，当且仅当它同时是这两个集合的元素

$$\forall x, s_1, s_2 \quad x \in (s_1 \cap s_2) \Leftrightarrow (x \in s_1 \wedge x \in s_2)$$

一个对象是两个集合的并集的元素，当且仅当它是其中某一集合的元素

$$\forall x, s_1, s_2 \quad x \in (s_1 \cup s_2) \Leftrightarrow (x \in s_1 \vee x \in s_2)$$

与 KB 进行互动采用 TELL, ASK。ASK 后 KB 返回一个绑定表，把变量与常量绑定在作为回答。

Ask(KB, S) returns some / all  $\sigma$  such that  $\text{KB} \models S\sigma$

## 使用绑定表 Example:

Tell(KB,Percept([Smell,Breeze,None],5))

Ask(KB,  $\exists a \text{ BestAction}(a, 5)$ )

I.e., does the KB entail some best action at  $t = 5$  ? Answer: Yes, {a/ Shoot }

$\leftarrow$  substitution (binding list 绑定表)

## Wumpus 世界的知识库:

Perception (感知)

$$\forall t, s, b \quad \text{Percept}([s, b, G \mid \text{itter}], t) \Rightarrow \text{Glitter}(t)$$

Reflex

$$\forall t \quad \text{Glitter}(t) \Rightarrow \text{BestAction(Grab,t)}$$

Reflex with internal state: do we have the gold already?

$$\forall t \text{ AtGold}(t) \wedge \neg \text{Holding}(\text{Gold}, t) \Rightarrow \text{BestAction}(\text{Grab}, t)$$

$\text{Holding}(\text{Gold}, t)$  cannot be observed  $\Rightarrow$  keeping track of change is essential

### 推导隐藏属性

Definition of adjacent squares

$$\forall x, y, a, b \quad \text{Adjacent}([x, y], [a, b]) \Leftrightarrow [a, b] \in \{[x+1, y], [x-1, y], [x, y+1], [x, y-1]\}$$

Properties of squares:

$$\forall s, t \quad \text{At}(\text{Agent}, s, t) \wedge \text{Breeze}(t) \Rightarrow \text{Breezy}(s)$$

Squares are breezy near a pit: Diagnostic rule (诊断规则) — infer cause from effect

$$\forall s \quad \text{Breezy}(s) \Rightarrow \exists r \text{ Adjacent}(r, s) \wedge \text{Pit}(r)$$

Causal rule (因果规则) - infer effect from cause

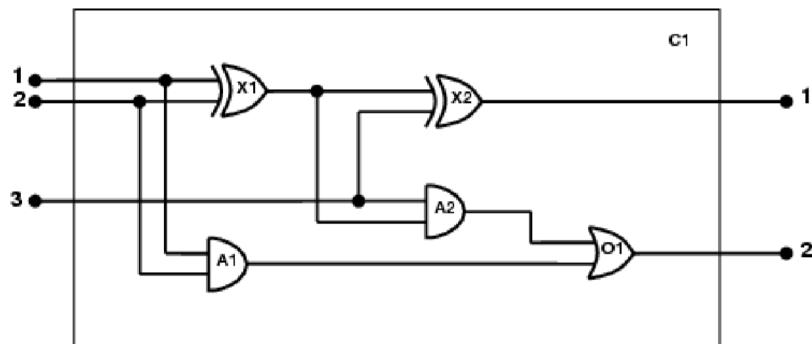
$$\forall r, s \quad \text{Adjacent}(r, s) \wedge \text{Pit}(r) \Rightarrow \text{Breezy}(s)$$

Neither of these is complete e.g., the causal rule doesn't say whether squares far away from pits can be breezy

Definition for the Breezy predicate:

$$\forall s \quad \text{Breezy}(s) \Leftrightarrow \exists r \quad \text{Adjacent}(r, s) \wedge \text{Pit}(r)$$

一阶逻辑中的知识工程:



1. Identify the task  
 Does the circuit actually add properly? (circuit verification)

2. Assemble the relevant knowledge  
 Composed of wires (导线) and gates (门); Types of gates (AND, OR, XOR, NOT)  
 Irrelevant: size, shape, color, cost of gates

3. Decide on a vocabulary (词汇表)  
 Alternatives:  
 $Type(X_1) = \text{XOR}$   
 $Type(X_1, \text{XOR})$   
 $\text{XOR}(X_1)$

4. Encode (编码) general knowledge of the domain  
(1) 如果两个接线端是相连的, 那么它们具有相同的信号  
 $\forall t_1, t_2 \text{ Connected}(t_1, t_2) \Rightarrow Signal(t_1) = Signal(t_2)$   
(2) 每个接线端的信号不是1就是0 (不可能两者都是)  
 $\forall t \ Signal(t) = 1 \vee Signal(t) = 0$   
 $t \neq 0$   
(3) Connected是一个可交换谓词  
 $\forall t_1, t_2 \text{ Connected}(t_1, t_2) \Leftrightarrow \text{Connected}(t_2, t_1)$   
(4) 或门的输出为1, 当且仅当它的某一个输入为1  
 $\forall g \ Type(g) = \text{OR} \Rightarrow$   
 $Signal(Out(1,g)) = 1 \Leftrightarrow \exists n \ Signal(In(n,g)) = 1$   
(5) 与门的输出为0, 当且仅当它的某一个输入为0  
 $\forall g \ Type(g) = \text{AND} \Rightarrow$   
 $Signal(Out(1,g)) = 0 \Leftrightarrow \exists n \ Signal(Out(n,g)) = 0$   
(6) 异或门的输出为1, 当且仅当它的输入是不相同的  
 $\forall g \ Type(g) = \text{XOR} \Rightarrow$   
 $Signal(Out(1,g)) = 1 \Leftrightarrow Signal(Out(1,g)) \neq Signal(Out(2,g))$   
(7) 非门的输出与它的输入相反  
 $\forall g \ Type(g) = \text{NOT} \Rightarrow Signal(Out(1,g)) \neq Signal(Out(1,g))$

5. Encode the specific problem instance

首先对门加以分类  
 $Type(X_1) = \text{XOR}$        $Type(X_2) = \text{XOR}$   
 $Type(A_1) = \text{AND}$      $Type(A_2) = \text{AND}$   
 $Type(O_1) = \text{OR}$

其次说明门与门之间的连接  
 $Connected(Out(1, X_1), In(1, X_2))$   
 $Connected(Out(1, X_1), In(2, A_2))$   
 $Connected(Out(1, A_2), In(1, O_1))$   
 $Connected(Out(1, A_1), In(2, O_1))$   
 $Connected(Out(1, X_2), Out(1, C_1))$   
 $Connected(Out(1, O_1), Out(2, C_1))$

$Connected(In(1, C_1), In(1, X_1))$   
 $Connected(In(1, C_1), In(1, A_1))$   
 $Connected(In(2, C_1), In(2, X_1))$   
 $Connected(In(2, C_1), In(2, A_1))$   
 $Connected(In(3, C_1), In(1, X_2))$   
 $Connected(In(3, C_1), In(1, A_2))$

6. Pose queries to the inference procedure—把查询提交给推理过程

What are the possible sets of values of all the terminals for the adder circuit?  
对于1位全加器有哪些可能的输入与输出组合?

$\exists i_1, i_2, i_3, o_1, o_2 \ Signal(Out(1, C_1)) = i_1 \wedge Signal(Out(2, C_1)) = i_2 \wedge Signal(Out(3, C_1)) = i_3 \wedge$   
 $Signal(Out(1, C_1)) = o_1 \wedge Signal(Out(2, C_1)) = o_2$

7. Debug the knowledge base

May have omitted assertions like  $1 \neq 0$

对异或门(XOR)尤其重要:

$Signal(Out(1, X_1)) = 1 \Leftrightarrow Signal(Out(1, X_1)) \neq Signal(Out(2, X_1))$

注意到在写复合语句时不要遗漏基本的知识:  $0 \neq 1$  等。

## 9 Chapter 9: 一阶逻辑推理

全称量词实例化可以多次应用从而获得许多不同的结果。

# Universal instantiation (UI)

## 全称实例化

Every instantiation of a universally quantified sentence is entailed by it:  
全称量化语句蕴含它的所有实例

$$\frac{\forall v \alpha}{\text{Subst}(\{v/g\}, \alpha)}$$

for any variable (变量)  $v$  and ground term (基项)  $g$

E.g.,  $\forall x King(x) \wedge greedy(x) \Rightarrow Evil(x)$  yields

$King(John) \wedge Greedy(John) \Rightarrow Evil(John)$

$King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard)$

$King(Father(John)) \wedge Greedy(Father(John)) \Rightarrow Evil(Father(John))$

⋮

存在量词实例化可以应用一次，然后用新的常量符号取代存在量化语句。

# Existential instantiation (EI)

## 存在实例化

For any sentence , variable  $v$ , and new constant symbol  $k$   
that does not appear elsewhere in the knowledge base:

$$\frac{\exists v \alpha}{\text{Subst}(\{v/k\}, \alpha)}$$

E.g.,  $\exists x Crown(x) \wedge OnHead(x, John)$  yields

$Crown(C_1) \wedge OnHead(C_1, John)$

provided  $C_1$  is a new constant symbol, called a Skolem constant (斯科伦常数)

Another example: from  $\exists x d(x^y) / dy = x^y$  we obtain

$d(e^y / dy) = e^y$

provided  $e$  is a new constant symbol

**Skolem** 常数代表一个新的常量符号，用于存在实例化。注意新知识库逻辑上并不等价于旧知识库，但只有在原始知识库可满足时，新的知识库才是可满足的。

存在的问题: 函数的存在造成无限多个基项。

**Theorem:**

如果某个语句被原始的一阶知识库蕴含，则存在一个只涉及命题化知识库的有限子集的证明:

Idea: For  $n = 0$  to  $\infty$  do

create a propositional KB by instantiating with depth-n terms see if  $\alpha$  is entailed by this KB

一阶逻辑的蕴涵是半可判定的。半可判定指的是若 KB 蕴涵  $f$  则算法在有限时间能够证明；若不蕴涵，则没有算法能在有限时间内证明出来。而且在推理的过程中，可能会产生很多无关的子句。对于  $p$  个  $k$  元谓词和  $n$  常量，有  $pn^k$  种实例化。

## 9.1 Unification 合一

采用置换  $\theta$  使得蕴涵的前提和 KB 中已有的语句完全相同，那么应用  $\theta$  后即可以断言对应蕴涵的结论。  
采取标准化分离避免变量名的重复使用(没有共享变量)，应该采用更加一般的合一(对变量的取值限制最少)。  
存在一个唯一的最一般合一者 MGU，不考虑变量的重新命名时，置换唯一。

p	q	$\theta$
Knows(John,x)	Knows(John,Jane)	{x/Jane}
Knows(John,x)	Knows(y,OJ)	{x/OJ,y/John}
Knows(John,x)	Knows(y,Mother(y))	{y/John,x/Mother(John)}
Knows(John,x)	Knows(x,OJ)	{fail}

Standardizing apart (标准化分离) eliminates overlap of variables, e.g.  
 $\text{Knows}(z_{17}, \text{OJ})$

## 9.2 GMP 一般化分离规则

GMP (一般化分离规则) :

$$\frac{p_1', p_2', \dots, p_n', (p_1 \wedge p_2 \wedge \dots \wedge p_n \Rightarrow q)}{q\theta} \text{ where } p_i'\theta = p_i\theta \text{ for all } i$$

$$\begin{array}{ll} p_1' \text{ is } \text{King}(John) & p_1 \text{ is } \text{King}(x) \\ p_2' \text{ is } \text{Greedy}(y) & p_2 \text{ is } \text{Greedy}(x) \\ \theta \text{ is } \{x/John, y/John\} & q \text{ is } \text{Evil}(x) \\ q\theta \text{ is } \text{Evil}(John) & \end{array}$$

优点：可靠的。

Proof:

Need to show that

$$p_1', \dots, p_n', (p_1 \wedge \dots \wedge p_n \Rightarrow q) \models q\theta$$

provided that  $p_i'\theta = p_i\theta$  for all  $i$

Lemma: For any sentence  $p$ , we have  $p \models p\theta$

1.  $(p_1 \wedge \dots \wedge p_n \Rightarrow q) \models (p_1 \wedge \dots \wedge p_n \Rightarrow q)\theta = (p_1\theta \wedge \dots \wedge p_n\theta \Rightarrow q\theta)$
2.  $p_1', \dots, p_n' \models p_1' \wedge \dots \wedge p_n' \models p_1'\theta \wedge \dots \wedge p_n'\theta$
3. From 1 and 2,  $q\theta$  follows by ordinary Modus Ponens

缺点：GMP 对 FOL 是不完备的：不是所有的语句都能被转换为霍恩子句。只有对有限子句的 FOL KB 是完备的。GMP 需要与 KB 中的确定子句(只有一个正文字)配合使用。

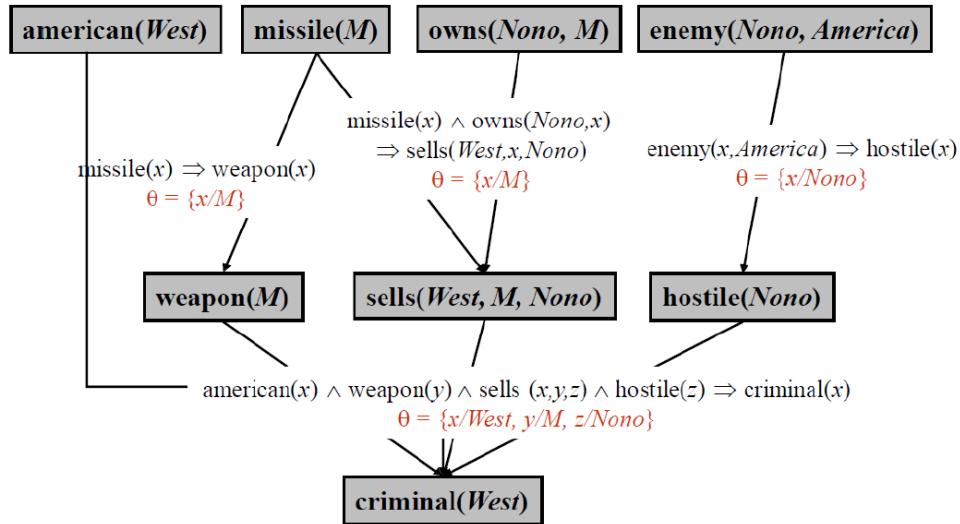
### 9.3 前向链接

```

function FOL-FC-Ask(KB, α) returns a substitution or false
    repeat until new is empty
        new  $\leftarrow \{ \}$ 
        for each sentence r in KB do
            (p1  $\wedge \dots \wedge$  pn  $\Rightarrow$  q)  $\leftarrow$  STANDARDIZE-A-PART(r)
            for each  $\theta$  such that (p1  $\wedge \dots \wedge$  pn) $\theta$  = (p'1  $\wedge \dots \wedge$  p'n) $\theta$ 
                for some p'1, ..., p'n in KB
                    q'  $\leftarrow$  SUBST( $\theta, q$ )
                    if q' is not a renaming of a sentence already in KB or new then do
                        add q' to new
                         $\phi \leftarrow$  UNIFY(q', α)
                        if  $\phi$  is not fail then return  $\phi$ 
                    add new to KB
    return false

```

## Forward chaining proof



**优点：**前向链接算法对一阶有限子句是可靠完备的。数据日志是有限的一阶逻辑子句且无函数(函数)。对于数据日志而言，不重复的基本事实不会多于  $pn^k$ (*p*为谓词的数量，*n*为常量符号的数量，*k*为谓词的(元数)最大参数个数)

**缺点：**当 KB 不蕴涵目标子句  $\alpha$  时算法可能无法终止。

**效率提升：**若一个规则的前提在 *k-1* 次迭代没有被加入 *new*, 则不需要对该条规则进行匹配。

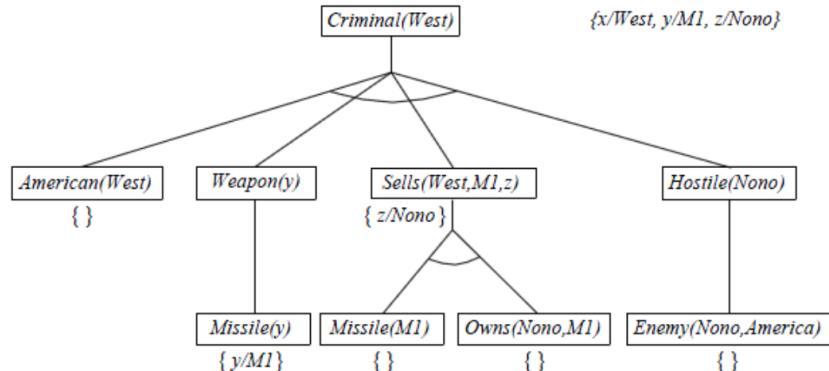
## 9.4 后向链接

```

function FOL-BC-ASK(KB, goals, θ) returns a set of substitutions
  inputs: KB, a knowledge base
    goals, a list of conjuncts forming a query (θ already applied)
    θ, the current substitution, initially the empty substitution { }
  local variables: answers, a set of substitutions, initially empty
  if goals is empty then return {θ}
  q' ← SUBST(θ, FIRST(goals))
  for each sentence r in KB
    where STANDARDIZE-APART(r) = (p1  $\wedge$  ...  $\wedge$  pn  $\Rightarrow$  q)
    and θ' ← UNIFY(q, q') succeeds
    new_goals ← [p1, ..., pn]REST(goals)
    answers ← FOL-BC-Ask(KB, new_goals, COMPOSE(θ', θ)) ∪ answers
  return answers

```

## Backward chaining example



**优点：**可以采用深度优先搜索，空间复杂度  $O(n)$ 。

仿照命题逻辑后向链接避免循环与重复工作。

```

PhD(x)   $\Rightarrow$  HighlyQualified(x)
 $\neg$ PhD(x)   $\Rightarrow$  EarlyEarnings(x)
HighlyQualified(x)   $\Rightarrow$  Rich(x)
EarlyEarnings(x)   $\Rightarrow$  Rich(x)
Query: Rich(Me)

```

算法只对一阶确定子句(霍恩子句)是完备的, 对一般的一阶逻辑知识库不完备。

## 9.5 归结算法

归结的收敛性前提是能转化为合取范式

Full first-order version:

$$\frac{l_1 \vee \dots \vee l_k, \quad m_1 \vee \dots \vee m_n}{(l_1 \vee \dots \vee l_{i-1} \vee l_{i+1} \vee \dots \vee l_k \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n)\theta}$$

where  $\text{Unify}(l_i, \neg m_j) = \theta$ .

Steps:

Everyone who loves all animals is loved by someone:  
 $\forall x [\forall y \text{Animal}(y) \Rightarrow \text{Loves}(x,y)] \Rightarrow [\exists y \text{Loves}(y,x)]$

1. Eliminate biconditionals and implications—消除蕴涵  
 $\forall x [\neg \forall y \neg \text{Animal}(y) \vee \text{Loves}(x,y)] \vee [\exists y \text{Loves}(y,x)]$

2. Move  $\neg$  inwards —将 $\neg$ 内移:  $\neg \forall x p \equiv \exists x \neg p, \neg \exists x p \equiv \forall x \neg p$   
 $\forall x [\exists y \neg (\neg \text{Animal}(y) \vee \text{Loves}(x,y))] \vee [\exists y \text{Loves}(y,x)]$   
 $\forall x [\exists y \neg \neg \text{Animal}(y) \wedge \neg \text{Loves}(x,y)] \vee [\exists y \text{Loves}(y,x)]$   
 $\forall x [\exists y \text{Animal}(y) \wedge \neg \text{Loves}(x,y)] \vee [\exists y \text{Loves}(y,x)]$

3. Standardize variables—变量标准化: each quantifier should use a different one

$\forall x [\exists y \text{Animal}(y) \wedge \neg \text{Loves}(x,y)] \vee [\exists z \text{Loves}(z,x)]$

4. Skolemize: a more general form of existential instantiation.

Each existential variable is replaced by a **Skolem function** (斯科伦函数) of the enclosing universally quantified variables:

$\forall x [\text{Animal}(F(x)) \wedge \neg \text{Loves}(x,F(x))] \vee \text{Loves}(G(x),x)$

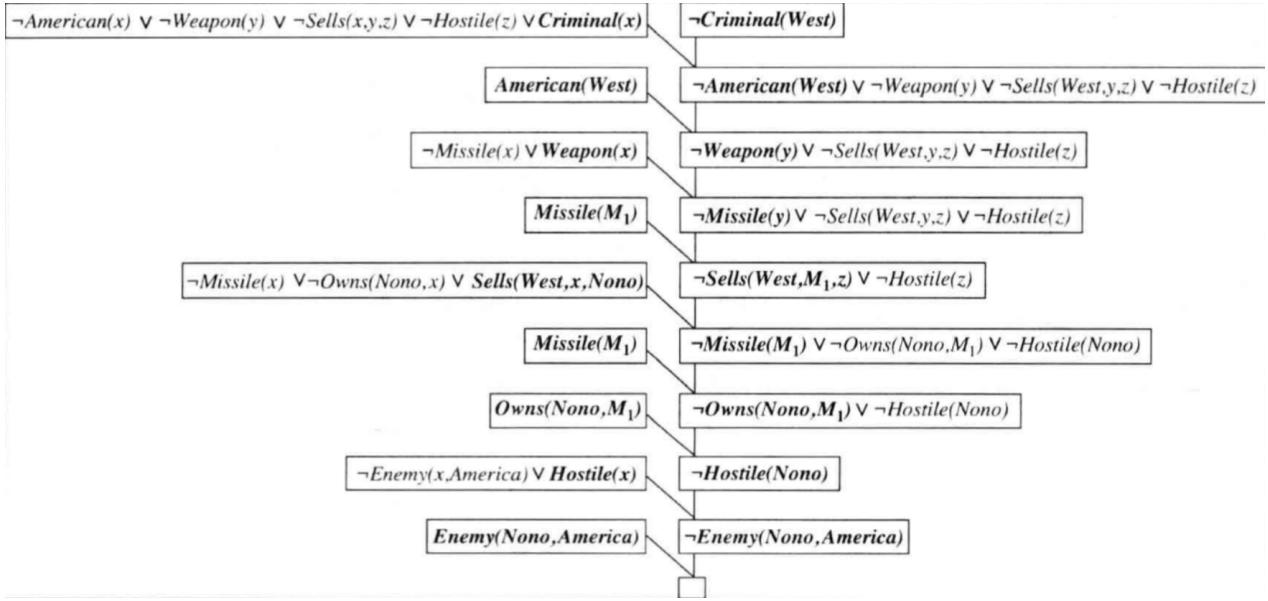
5. Drop universal quantifiers—去除全称量词:

$[\text{Animal}(F(x)) \wedge \neg \text{Loves}(x,F(x))] \vee \text{Loves}(G(x),x)$

6. Distribute  $\vee$  over  $\wedge$ —将 $\vee$ 分配到 $\wedge$ 中:

$[\text{Animal}(F(x)) \vee \text{Loves}(G(x),x)] \wedge [\neg \text{Loves}(x,F(x)) \vee \text{Loves}(G(x),x)]$

Example: P308,309



## 10 不确定度的量化与概率推理

### 10.1 概率基础

概率的定义, 连续/离散随机变量, 概率公理, 先验概率, 边缘分布, 联合分布, 全概率公式, 独立性, 条件独立性,

贝叶斯法则，朴素贝叶斯模型,极大似然估计。

归一化技巧 记  $Y$ : 查询变量,  $E$ : 证据变量,  $H = X - Y - E$ : 未观测变量

$$P(Y | E = e) = \alpha P(Y, E = e) = \alpha \sum_h P(Y, E = e, H = h)$$

	toothache		$\neg$ toothache	
	catch	$\neg$ catch	catch	$\neg$ catch
cavity	.108	.012	.072	.008
$\neg$ cavity	.016	.064	.144	.576

Denominator (分母) can be viewed as a **normalization constant  $\alpha$**

$$\begin{aligned} \mathbf{P}(\text{Cavity} | \text{toothache}) &= \alpha \mathbf{P}(\text{Cavity}, \text{toothache}) \\ &= \alpha [\mathbf{P}(\text{Cavity}, \text{toothache}, \text{catch}) + \mathbf{P}(\text{Cavity}, \text{toothache}, \neg \text{catch})] \\ &= \alpha [<0.108, 0.016> + <0.012, 0.064>] \\ &= \alpha <0.12, 0.08> = <0.6, 0.4> \end{aligned}$$

General idea: compute distribution on query variable by fixing evidence variables (证据变量) and summing over hidden variables (未观测变量)

缺点: 时间复杂度  $O(d^n)$

条件独立性 Random variables can be dependent, but conditionally independent.

$$P(\text{MaryCall} | \text{JohnCall}) \neq P(\text{MaryCall})$$

$$P(\text{MaryCall} | \text{Alarm}, \text{JohnCall}) = P(\text{MaryCall} | \text{Alarm})$$

In general, "A and B are conditionally independent given C" means:

$$\begin{aligned} P(A | B, C) &= P(A | C) \\ P(B | A, C) &= P(B | C) \\ P(A, B | C) &= P(A | C)P(B | C) \end{aligned}$$

$\mathbf{P}(\text{Toothache}, \text{Catch}, \text{Cavity})$

$$\begin{aligned} &= \mathbf{P}(\text{Toothache} | \text{Catch}, \text{Cavity}) \mathbf{P}(\text{Catch}, \text{Cavity}) \\ &= \mathbf{P}(\text{Toothache} | \text{Catch}, \text{Cavity}) \mathbf{P}(\text{Catch} — \text{Cavity}) \mathbf{P}(\text{Cavity}) \\ &= \mathbf{P}(\text{Toothache} | \text{Cavity}) \mathbf{P}(\text{Catch} | \text{Cavity}) \mathbf{P}(\text{Cavity}) \end{aligned}$$

I.e.,  $2 + 2 + 1 = 5$  independent numbers

在大多数情况下, 使用条件独立性能将全联合概率的表示由  $n$  的指数关系减为  $n$  的线性关系。

贝叶斯法则

$$\mathbf{P}(Y | X) = \frac{\mathbf{P}(X | Y)\mathbf{P}(Y)}{\mathbf{P}(X)} = \alpha \mathbf{P}(X | Y)\mathbf{P}(Y)$$

朴素贝叶斯模型

$$P(C, E_1, E_2, \dots, E_n) = P(C) \prod_{i=1}^n P(E_i | C) \quad (1)$$

## 极大似然估计

从 Data 中进行概率估计。

$$\hat{P}(x) = \frac{\text{count}(x)}{\text{total samples}}$$

## 10.2 贝叶斯网络

每个 Node 代表一个随机变量(或一组随机变量), Edge 表示变量之间的概率关系, 构成有向无环图。拓扑结构反映变量之间的因果、独立(条件独立)性质。

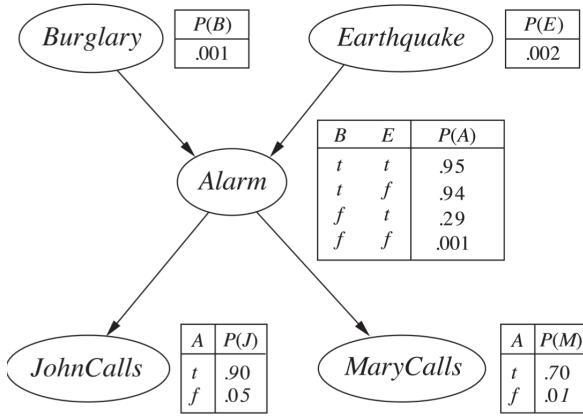


图 18: For burglary net,  $1 + 1 + 4 + 2 + 2 = 10$  numbers (vs.  $25 - 1 = 31$ )

Burglary 是影响 Alarm 的原因。

JohnCalls 与 MaryCalls 在给定 Alarm 下是独立的。

**紧致性:** 一个具有  $k$  个布尔父结点的布尔变量的条件概率表中有  $2^k$  个独立的可指定概率。若每个变量的父节点数小于  $k$ , 则存储网络的概率分布需要  $O(n2^k)$  的空间, 与完全联合分布时的  $O(2^n)$  相比, 空间复杂度随  $n$  线性增长。

**网络性质:**

1). 全联合概率可以表示成贝叶斯网络中条件概率分布的乘积。

$$\begin{aligned}
 P(x_1, \dots, x_n) &= \prod_{i=1}^n P(x_i | \text{parents}(X_i)) \\
 \text{e.g., } P(j \wedge m \wedge a \wedge \neg b \wedge \neg e) &= P(j | a)P(m | a)P(a | \neg b, \neg e)P(\neg b)P(\neg e) \\
 &= 0.9 \times 0.7 \times 0.001 \times 0.999 \times 0.998 \\
 &\approx 0.00063
 \end{aligned}$$

2). 给定父节点, 一个 Node 与它的非后代结点是条件独立的。

*Theorem : 1  $\Leftrightarrow$  2*

证明思路:

$2 \Rightarrow 1$ : 对有向无环图, 按拓扑排序的结构, 依次对图进行拆分。每次提出一个 Parent, 都可以使得被分割出来的子图数目 + 1, 直到不能再作分割, 即可将全联合分布拆成各个条件分布的乘积。

$1 \Rightarrow 2$ : 递归来看, 只需考虑类似上图 Alarm, JohnCalls, MaryCalls 的三角结构。由  $P(j|a)P(m|a)P(a) = P(j, m, a) \Rightarrow P(j|a)P(m|a) = \frac{P(j, m, a)}{P(a)} \Rightarrow 2$  因果链:

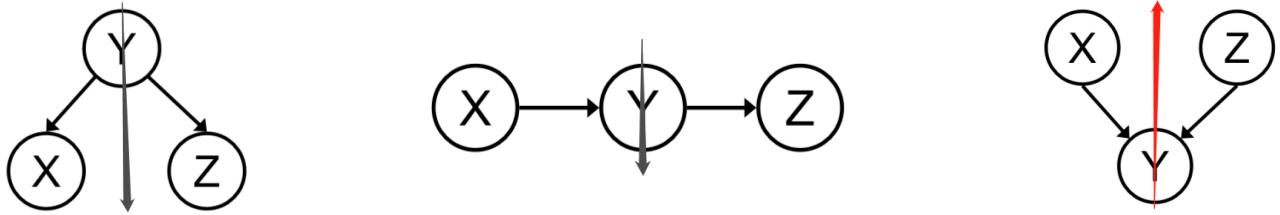


图 19: 灰色箭头: X, Z 给定 Y 后独立; 红色箭头: X, Z 独立, 给定 Y 后不独立。观察原因会阻止结果之间的影响, 路径上的证据阻止更远的影响, 观察结果可以在原因之间产生影响。

## 构造网络

1. Choose an ordering of variables  $X_1, \dots, X_n$

2. For  $i = 1$  to  $n$

add  $X_i$  to the network

select parents from  $X_1, \dots, X_{i-1}$  such that

$$\mathbf{P}(X_i | \text{Parents}(X_i)) = \mathbf{P}(X_i | X_1, \dots, X_{i-1})$$

要求网络的拓扑结构反映合适的父结点集对每个变量的那些直接影响。

添加结点的正确次序是首先添加“根本原因”结点, 然后加入受它们直接影响的变量, 以此类推。

## 推理

简单后验概率计算, 联合查询, 最优决策

Simple query on the burglary network:

$$\begin{aligned} \mathbf{P}(B | j, m) &= \mathbf{P}(B, j, m) / P(j, m) \\ &= \alpha \mathbf{P}(B, j, m) \\ &= \alpha \sum_e \sum_a \mathbf{P}(B, e, a, j, m) \\ &= \alpha \sum_e \sum_a \mathbf{P}(B) P(e) \mathbf{P}(a | B, e) P(j | a) P(m | a) \end{aligned}$$

Recursive depth-first enumeration:  $O(n)$  space,  $O(d^n)$  time

$$\begin{aligned}
& \mathbf{P}(B \mid j, m) \\
&= \alpha \underbrace{\mathbf{P}(B)}_{B} \sum_e \underbrace{P(e)}_{E} \sum_a \underbrace{\mathbf{P}(a \mid B, e)}_{A} \underbrace{P(j \mid a)}_{J} \underbrace{P(m \mid a)}_{M} \\
&= \alpha \mathbf{P}(B) \sum_e P(e) \sum_a \mathbf{P}(a \mid B, e) P(j \mid a) f_M(a) \\
&= \alpha \mathbf{P}(B) \sum_e P(e) \sum_a \mathbf{P}(a \mid B, e) f_J(a) f_M(a) \\
&= \alpha \mathbf{P}(B) \sum_e P(e) \sum_a f_A(a, b, e) f_J(a) f_M(a) \\
&= \alpha \mathbf{P}(B) \sum_e P(e) f_{\bar{A}JM}(b, e) (\text{sum out } A) \\
&= \alpha \mathbf{P}(B) f_{\bar{E}\bar{A}JM}(b) (\text{sum out } E) \\
&= \alpha f_B(b) \times f_{\bar{E}\bar{A}JM}(b)
\end{aligned}$$

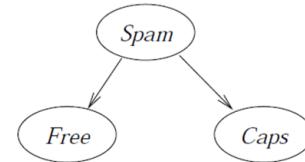
Exact inference by variable elimination:

- polytime on polytrees, NP-hard on general graphs
- space = time, very sensitive to topology

贝叶斯网络应用

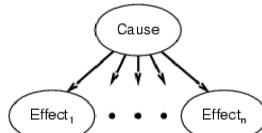
## Naïve Bayes model

Example: Spam detection



$$\mathbf{P}(\text{Cause}, \text{Effect}_1, \dots, \text{Effect}_n) = \mathbf{P}(\text{Cause}) \prod_i \mathbf{P}(\text{Effect}_i \mid \text{Cause})$$

$$\begin{aligned}
\mathbf{P}(\text{Cause} | \text{Effect}_1, \dots, \text{Effect}_n) &= \mathbf{P}(\text{Effects}, \text{Cause}) / \mathbf{P}(\text{Effects}) \\
&= \alpha \mathbf{P}(\text{Cause}, \text{Effects}) = \alpha \mathbf{P}(\text{Cause}) \prod_i \mathbf{P}(\text{Effect}_i \mid \text{Cause})
\end{aligned}$$



Total number of parameters (参数) is linear in  $n$

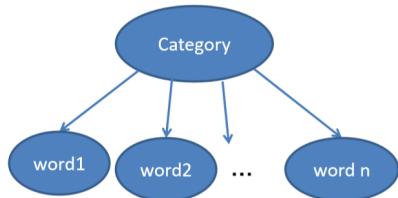
$$\mathbf{P}(\text{Free}, \text{Caps}, \text{Spam}) = \mathbf{P}(\text{Spam}) \mathbf{P}(\text{Caps} \mid \text{Spam}) \mathbf{P}(\text{Free} \mid \text{Spam})$$

Free	Caps	Spam	# messages
Y	Y	Y	20
Y	Y	N	1
Y	N	Y	5
Y	N	N	0
N	Y	Y	20
N	Y	N	3
N	N	Y	2
N	N	N	49
Total:			100

Spam	P(Spam)
Y	$\frac{100}{20+5+20+2} = 0.47$
N	$\frac{100}{1+0+3+49} = 0.53$

Caps	Spam	P(Caps Spam)	Free	Spam	P(Free Spam)
Y	Y	$\frac{20+20}{20+5+20+2} \approx 0.8511$	Y	Y	$\frac{20+5}{20+5+20+2} \approx 0.5319$
Y	N	$\frac{1+0+3+49}{1+0+3+49} \approx 0.0755$	Y	N	$\frac{1+0}{1+0+3+49} \approx 0.0189$
N	Y	$\frac{20+5+20+2}{20+5+20+2} \approx 0.1489$	N	Y	$\frac{20+5+20+2}{20+5+20+2} \approx 0.4681$
N	N	$\frac{0+49}{1+0+3+49} \approx 0.9245$	N	N	$\frac{0+49}{1+0+3+49} \approx 0.9811$

## Example: Learning to classify text documents



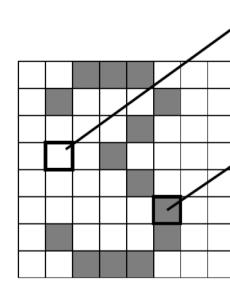
The model consists of the prior probability  $P(\text{Category})$  and the conditional probabilities  $P(\text{word } i | \text{Category})$

- $P(\text{Category} = c)$  is estimated as the fraction of all documents that are of category  $c$
- $P(\text{word } i = \text{true} | \text{Category} = c)$  is estimated as the fraction of documents of category  $c$  that contain word  $i$

## Examples: CPTs

$P(Y)$

	$P(Y)$
1	0.1
2	0.1
3	0.1
4	0.1
5	0.1
6	0.1
7	0.1
8	0.1
9	0.1
0	0.1



	$P(F_{3,1} = \text{on} Y)$
1	0.01
2	0.05
3	0.05
4	0.30
5	0.80
6	0.90
7	0.05
8	0.60
9	0.50
0	0.80

	$P(F_{5,5} = \text{on} Y)$
1	0.05
2	0.01
3	0.90
4	0.80
5	0.90
6	0.90
7	0.25
8	0.85
9	0.60
0	0.80

## 11 Chapter 18: Learning

定义: Tom Mitchell (1998) Well-posed Learning Problem: A computer program is said to learn from experience  $E$  with respect to some task  $T$  and some performance measure  $P$ , if its performance on  $T$ , as measured by  $P$ , improves with experience  $E$ .

Keypoints: 监督/无监督学习, 特征向量表示, Given data  $D$ , we learn the model parameters , from which we can predict new data points.

### 11.1 决策树

对于每个例子都有一条到叶的路径, 任何训练集都有一个一致的决策树 (除非f在x中不确定), 对于测试集则不一定。我们希望选择更紧凑的决策树。

目的: 找到与训练例子一致的小树.

想法: 递归地选择“最重要”属性作为子树的根.

优点: 造价低廉, 解释性强, 分类速度快, 对于许多简单的数据集准确性很高

```

function DECISION-TREE-LEARNING(examples, attributes, parent-examples) returns
  a tree
  if examples is empty then return PLURALITY-VALUE(parent-examples)
  else if all examples have the same classification then return the classification
  else if attributes is empty then return PLURALITY-VALUE(examples)
  else
     $A \leftarrow \arg\max_{a \in \text{attributes}} \text{IMPORTANCE}(a, \text{examples})$ 
    tree  $\leftarrow$  a new decision tree with root test  $A$ 
    for each value  $v_k$  of  $A$  do
       $exs \leftarrow \{e : e \in \text{examples} \text{ and } e.A = v_k\}$ 
      subtree  $\leftarrow$  DECISION-TREE-LEARNING(exs, attributes -  $A$ , examples)
      add a branch to tree with label ( $A = v_k$ ) and subtree subtree
  return tree
  
```

对于包含 p 个正例和 n 个负例的训练集:

$$I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) = -\frac{p}{p+n} \log_2 \frac{p}{p+n} - \frac{n}{p+n} \log_2 \frac{n}{p+n}$$

$$\text{remainder}(A) = \sum_{i=1}^v \frac{p_i + n_i}{p+n} I\left(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i}\right)$$

$$IG(A) = I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) - \text{remainder}(A)$$

**Example:**

For the training set,  $p = n = 6$ ,  $I(6/12, 6/12) = 1$  bit

Consider the attributes *Patrons* and *Type* (and others too):

$$IG(\text{Patrons}) = 1 - \left[ \frac{2}{12} I(0,1) + \frac{4}{12} I(1,0) + \frac{6}{12} I\left(\frac{2}{6}, \frac{4}{6}\right) \right] = .541 \text{ bits}$$

$$IG(\text{Type}) = 1 - \left[ \frac{2}{12} I\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{2}{12} I\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{4}{12} I\left(\frac{2}{4}, \frac{2}{4}\right) + \frac{4}{12} I\left(\frac{2}{4}, \frac{2}{4}\right) \right] = 0 \text{ bits}$$

*Patrons* has the highest IG of all attributes and so is chosen by the DTL algorithm as the root

通过训练集得出的树比“真正的”树简单得多——一个更复杂的假设是无法通过少量数据证明的。

## 11.2 K 近邻

在训练数据集中选取 k 个离自己最近的数据点，取其均值等作为对测试集数据的估计。

## 11.3 线性预测

分类=从有限离散标签的数据中学习。二值分类可以看作是在特征空间中进行类的分离。

### 11.3.1 问题形式化

理想情况是找到一个分类器  $h(\mathbf{x}) = \text{sign}(\mathbf{w}^\top \mathbf{x} + b)$  来减小 0/1 损失

$$\min_{\mathbf{w}, b} \sum_i L_{0/1}(h(\mathbf{x}_i), y_i)$$

由于不易优化，考虑采用其它损失函数

$$L_2(h(\mathbf{x}), y) = (y - \mathbf{w}^\top \mathbf{x} - b)^2 = (1 - y(\mathbf{w}^\top \mathbf{x} + b))^2$$

$$L_1(h(\mathbf{x}), y) = |y - \mathbf{w}^\top \mathbf{x} - b| = |1 - y(\mathbf{w}^\top \mathbf{x} + b)|$$

$$L_{\text{hinge}}(h(\mathbf{x}), y) = (1 - y(\mathbf{w}^\top \mathbf{x} + b))_+$$

$L_2$  损失最常使用:

$$\begin{aligned}\frac{\partial Loss}{\partial \mathbf{w}} &= 2(\mathbf{X}\mathbf{w} - \mathbf{y})^\top \mathbf{X} = 0 \\ \mathbf{X}^\top \mathbf{X}\mathbf{w} - \mathbf{X}^\top \mathbf{y} &= 0 \\ \mathbf{w}^* &= (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}\end{aligned}$$

得到最终预测结果:

$$\hat{y} = \text{sign} \left( \mathbf{w}^{*\top} \begin{bmatrix} 1 \\ \mathbf{x}_0 \end{bmatrix} \right) = \text{sign} \left( \mathbf{y}^\top \mathbf{X}^{+\top} \begin{bmatrix} 1 \\ \mathbf{x}_0 \end{bmatrix} \right)$$

事实上，可以将任意基函数作为拟合的变量:

$$f(\mathbf{x}, \mathbf{w}) = b + w_1\phi_1(\mathbf{x}) + w_2\phi_2(\mathbf{x}) + \cdots + w_m\phi_m(\mathbf{x})$$

回归问题中，既需要避免欠拟合，也要避免过拟合。

**训练/测试误差:** 在训练/测试集上发生的误差

**泛化误差:** 未知记录上的期望误差

**欠拟合:** 当模型过于简单时，训练和测试误差都很大

**过拟合:** 当模型过于复杂时，训练误差较小，测试误差较大

**奥卡姆剃刀原则:** 优先选择与数据一致的最简单假设，一个复杂的模型更有可能因数据错误而被过分拟合。

### 11.3.2 过拟合解决方案

规范化

$$\text{L2 regularization } \mathbf{w}^* = \arg \min_{\mathbf{w}} \text{Loss} + \lambda \|\mathbf{w}\|^2$$

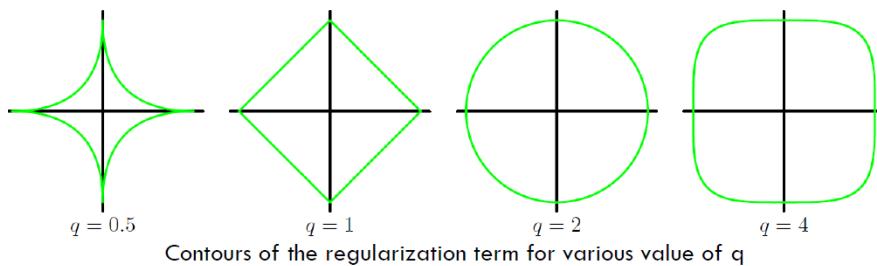
$$\text{LT regularization } \mathbf{w}^* = \arg \min_{\text{Loss}} + \lambda |\mathbf{w}|$$

$$\begin{aligned}\mathbf{w}^* &= \arg \min_{\mathbf{w}} \text{Loss} + \lambda \cdot \text{penalty}(\mathbf{w}) \\ &= \arg \min_{\mathbf{w}} \text{Loss} + \lambda R_q\end{aligned}$$

$$R_q = \sum_i |w_i|^q$$

When  $\lambda$  sufficiently large, equivalent to:

$$\min_{\mathbf{w}} \text{Loss} \text{ subject to } \sum_i |w_i|^q \leq \eta$$



$L_2$  更易于优化， $L_1$  的解更稀疏。 $L_2$  正则相当于用圆去逼近目标，而  $L_1$  正则相当于用菱形去逼近目标，所以更容易引起交点在坐标轴上即得到稀疏解。从导数角度看，设原损失函数在  $w = 0$  处的导数为  $d_0$ ，加了  $L_2$  的损失函数的导数在  $w = 0$  时不变，而加了  $L_1$  的损失函数的导数在  $w = 0$  时有突变，从  $d_0 + \lambda$  到  $d_0 - \lambda$ ，若  $d_0 + \lambda$  和  $d_0 - \lambda$  异号，则在 0 处会是一个极小值点。因此，优化时，很可能优化到该极小值点上，即  $w = 0$  处。

## More than two classes

### Learn:

- parameters  $W (d \times k)$  for a model  $f_W : X \rightarrow Y$
- Objective  $\min_W \text{tr}((XW - Y)(XW - Y)^\top)$

■ A convex quadratic, so just solve for a critical point:

$$\frac{d}{dW} = 2X^\top(XW - Y) = 0$$

---

$$\begin{aligned}\text{Thus } X^\top XW &= X^\top Y \\ W &= (X^\top X)^{-1}X^\top Y = X^\dagger Y\end{aligned}$$

最小二乘对于分类不是最好的方法，但其易于训练且有闭式解。

交叉验证

## Cross-validation

The improved holdout method: *k-fold cross-validation*

- Partition data into  $k$  roughly equal parts;
- Train on all but  $j$ -th part, test on  $j$ -th part



## 12 Chapter 20: SVM

### 12.1 分类间距

根据判别函数

$$\hat{y}(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$$

$\hat{y} > 0$  时即为正例。注意到对于几何间隔，当  $\mathbf{w}, b$  成倍时时不变的，而函数间隔也会成倍。

$$\mathbf{w} \rightarrow a\mathbf{w}, b \rightarrow a \cdot b$$

一个向量  $x_i$  到分界面的距离定义为

$$r = \frac{|\mathbf{w}^\top \mathbf{x}_i + b|}{\|\mathbf{w}\|}$$

## 12.2 问题形式化

**支持向量:** 接近于超平面的样本。

最大化几何间隔表示只有支持向量是重要的;其他的训练例子可以被忽略。

$$\max_{\mathbf{w}, b} \frac{1}{\|\mathbf{w}\|} \quad \text{subject to} \quad y_i (\mathbf{w}^\top \mathbf{x}_i + b) \geq 1, \forall i$$

$$m = \frac{1}{\|\mathbf{w}\|}$$

**解法一:** 原问题引出的二次规划

$$\min_{\mathbf{w}, b} \mathbf{w}^\top \mathbf{w} \quad \text{subject to} \quad y_i (\mathbf{w}^\top \mathbf{x}_i + b) \geq 1, \forall i$$

**解法二:** 对偶问题

$$\begin{aligned} \max_{\alpha} & \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j (\mathbf{x}_i^\top \mathbf{x}_j) \quad \text{subject to} \quad \alpha_i \geq 0, \forall i \\ & \sum_{i=1}^n \alpha_i y_i = 0 \end{aligned}$$

由于

$$\max_{\alpha_i \geq 0} \alpha_i [1 - y_i (\mathbf{w}^\top \mathbf{x}_i + b)] = 0 \quad y_i (\mathbf{w}^\top \mathbf{x}_i + b) \geq 1$$

因此有  $\alpha^* = 0$ ;  $\mathbf{x}_i$  非支持向量。进一步解出

$$\begin{aligned} \mathbf{w} &= \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i = \sum_{i \in SV} \alpha_i y_i \mathbf{x}_i \\ b &= y_i - \mathbf{w}^\top \mathbf{x}_i \quad \text{for any } i \text{ that } \alpha_i \neq 0 \end{aligned}$$

对测试集数据  $\mathbf{x}'$  做出预测时

$$\text{Compute } \mathbf{w}^\top \mathbf{x}' + b = \sum_{i \in SV} \alpha_i y_i (\mathbf{x}_i^\top \mathbf{x}') + b$$

and classify  $\mathbf{x}'$  as class 1 if the sum is positive, and class 2 otherwise

\*\* w 不需要显示地算出来

## 12.3 软间隔

处理非线性可分的情况, 本质上还是线性 SVM。

QP:

$$\begin{aligned} \min_{\mathbf{w}, b} & \frac{1}{2} \mathbf{w}^\top \mathbf{w} + C \sum_i \xi_i \quad \text{subject to} \quad y_i (\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i, \forall i \\ & \xi_i \geq 0, \forall i \end{aligned}$$

$\xi_i$  代表了误差的上界。C 控制软硬程度, 权衡边际最大化和拟合训练数据的相对重要性。C 越大, 对误差的惩罚越重。

对偶问题:

$$\begin{aligned} \max_{\alpha} & \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j (\mathbf{x}_i^\top \mathbf{x}_j) \quad \text{subject to} \quad 0 \leq \alpha_i \leq C, \forall i \\ & \sum_{i=1}^n \alpha_i y_i = 0 \end{aligned}$$

## 12.4 损失函数

### Loss in SVM

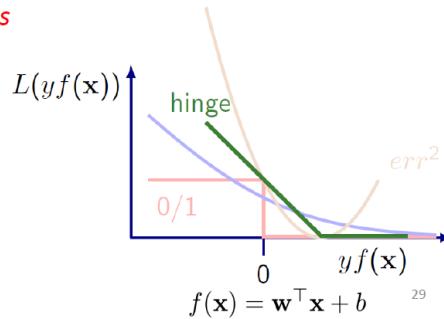
$$\min_{\mathbf{w}, b, \xi} \frac{1}{2} \mathbf{w}^\top \mathbf{w} + C \sum_i \xi_i \quad \text{subject to} \quad y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i, \forall i \\ \xi_i \geq 0, \forall i$$

Loss is measured as

$$\xi_i = \max(0, 1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b)) = [1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b)]_+$$

This loss is known as *hinge loss*

$$\boxed{\min_{\mathbf{w}, b} \frac{1}{2C} \mathbf{w}^\top \mathbf{w} + \sum_i \text{hingeloss}_i}$$



线性 SVM 总结：

分类器是一个分离超平面。

最“重要”的训练点是支持向量；它们定义了超平面。

二次优化算法可以识别哪些训练点  $\mathbf{x}_i$  是非零拉格朗日乘数  $\alpha_i$  的支持向量。

无论是在问题的对偶公式中，还是在求解过程中，训练点只出现在内积中。

## 12.5 非线性 SVM

### Linear in what?

Prediction driven by score:  $\mathbf{w} \cdot \phi(\mathbf{x})$

- Linear in  $\mathbf{w}$ ? Yes
- Linear in  $\phi(\mathbf{x})$ ? Yes
- Linear in  $\mathbf{x}$ ? No!

**Key idea: non-linearity**

- Predictors  $f_w(\mathbf{x})$  can be expressive non-linear functions and decision boundaries of  $\mathbf{x}$ .
- Score  $\mathbf{w} \cdot \phi(\mathbf{x})$  is linear function of  $\mathbf{w}$  and  $\phi(\mathbf{x})$

思想：原始特征空间总是可以映射到某个训练集可分离的高维特征空间。

优点：挖掘高维特征，构建特征之间的交互，

核技巧: (衡量两个输入变量的相似性)

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

核 Example:

## An Example for feature mapping and kernels

- Consider an input  $\mathbf{x} = [x_1, x_2]$
- Suppose  $\phi(\cdot)$  is given as follows

$$\phi\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = 1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2$$

- An inner product in the feature space is

$$\left\langle \phi\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \phi\begin{pmatrix} x'_1 \\ x'_2 \end{pmatrix} \right\rangle =$$

- So, if we define the **kernel function** as follows, there is no need to carry out  $\phi(\cdot)$  explicitly

$$K(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x}^T \mathbf{x}')^2$$

核矩阵:  $K_{i,j} = \varphi(\mathbf{x}_i)^T \varphi(\mathbf{x}_j)$

需满足的条件:

- Symmetry  $K = K^T$
- Positive-semidefinite (半正定)  $\mathbf{z}^T K \mathbf{z} \geq 0, \forall \mathbf{z} \in R^n$

总结:

SVM training: build a kernel matrix K using training data

- Linear:  $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$
- Gaussian (radial-basis function 径向基函数):  $K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\sigma^2}}$

solve the following quadratic program

$$\begin{aligned} & \max_{\alpha} \alpha^T \mathbf{e} - \frac{1}{2} \alpha^T (\mathbf{y} \mathbf{y}^T \circ K) \alpha \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C, \forall i \\ & \sum_{i=1}^n \alpha_i y_i = 0 \end{aligned}$$

SVM testing: now with  $\alpha_i$ , recover b,

$$b = y_i - \sum_{j=1}^n \alpha_j y_j K(\mathbf{x}_i, \mathbf{x}_j) \quad \text{for any } i \text{ that } \alpha_i \neq 0$$

we can predict new data points by:

$$y^* = \text{sign} \left( \sum_{i \in SV} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}') + b \right)$$

## 13 Chapter 21: Logistic回归

类别: 参数学习模型; 线性分类模型; 判别模型: 直接估计条件似然  $p(y|x)$

原理:

$$\begin{aligned} p(y=1 | \mathbf{x}) &= \sigma(\mathbf{w}^T \mathbf{x}) = \frac{1}{1+\exp(-\mathbf{w}^T \mathbf{x})} = \frac{\exp(\mathbf{w}^T \mathbf{x})}{1+\exp(\mathbf{w}^T \mathbf{x})} \\ p(y=0 | \mathbf{x}) &= 1 - p(y=1 | \mathbf{x}) = \frac{1}{1+\exp(\mathbf{w}^T \mathbf{x})} \end{aligned}$$

几率(Odds):  $\log \frac{p(y=1|\mathbf{x})}{1-p(y=1|\mathbf{x})} = \mathbf{w}^\top \mathbf{x}$

分界面:  $p(y=1 | \mathbf{x}) = 0.5 \Leftrightarrow \mathbf{w}^\top \mathbf{x} = 0$

训练过程(最大化似然函数):

$$\begin{aligned} \max_{\mathbf{w}} \ell(\mathbf{w}) &= \max_{\mathbf{w}} \sum_{i=1}^N \log p(y_i | \mathbf{x}_i; \mathbf{w}) \\ \text{Or } \min_{\mathbf{w}} J(\mathbf{w}) &= \min_{\mathbf{w}} -\ell(\mathbf{w}) \\ &= \min_{\mathbf{w}} - \left[ \sum_{i=1}^N y_i \log \sigma(\mathbf{w}^\top \mathbf{x}_i) + (1 - y_i) \log (1 - \sigma(\mathbf{w}^\top \mathbf{x}_i)) \right] \end{aligned}$$

梯度下降: 重复 {

$$w_j := w_j - \alpha \frac{\partial}{\partial w_j} J(\mathbf{w})$$

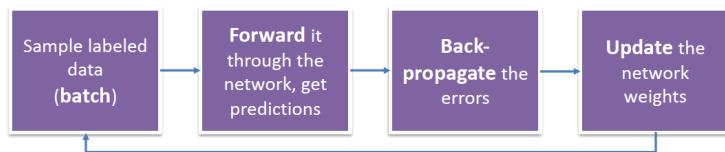
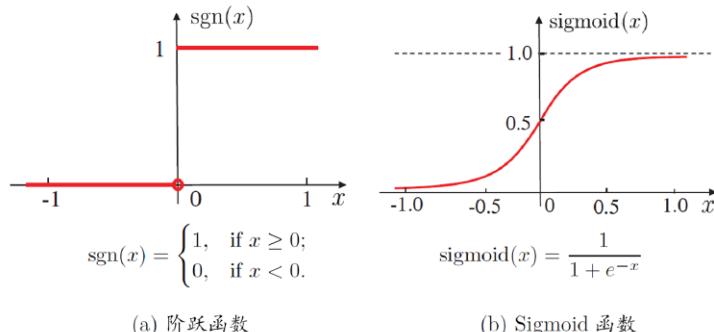
} (同步更新所有的  $w_j$ )

公式推导

$$\begin{aligned} \frac{\partial}{\partial z} \sigma(z) &= \sigma(z)(1 - \sigma(z)) \\ \frac{\partial}{\partial w_j} J(\mathbf{w}) &= \sum_{i=1}^N x_i (y_i - \sigma(\mathbf{w}^\top \mathbf{x}_i)) \end{aligned}$$

## 14 Chapter 22: 神经网络

激活函数:



# Optimization

➤ Backward

$$\frac{\partial \text{loss}}{\partial x^{(L)}} = g'(x^{(L)})$$

$$x^{(L+1)} = f(w^{(L)}x^{(L)})$$

$$\frac{\partial \text{loss}}{\partial x^{(L-1)}} = W^{(L-1)} \cdot \frac{\partial \text{loss}}{\partial x^{(L)}} \odot f'(W^{(L-1)}x^{(L-1)})$$

$$\frac{\partial \text{loss}}{\partial w^{(L-1)}} = \frac{\partial \text{loss}}{\partial x^{(L)}} \odot f'(W^{(L-1)}x^{(L-1)}) \cdot (x^{(L-1)})^T$$

## 多层前馈网络表示能力

只需要一个包含足够多神经元的隐层，多层前馈神经网络就能以任意精度逼近任意复杂度的连续函数

[Hornik et al., 1989]

## 多层前馈网络局限

- 神经网络由于强大的表示能力，经常遭遇过拟合。表现为：训练误差持续降低，但测试误差却可能上升
- 如何设置隐层神经元的个数仍然是个未决问题。实际应用中通常使用“试错法”调整

## 缓解过拟合的策略

- 早停**：在训练过程中，若训练误差降低，但验证误差升高，则停止训练
- 正则化**：在误差目标函数中增加一项描述网络复杂程度的部分，例如连接权值与阈值的平方和

## 14.1 CNN

采用共享变量的方式，使得需要学习的参数大大减少。

- Convolution (weighted sum)
  - Input: data to be convolved
  - Filter: convolution kernel, also the weight
  - Feature map: the output after the data being convolved.

### Example

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

\*

1	0	1
0	1	0
1	0	1

=

4	3	4
2	4	3
2	3	4

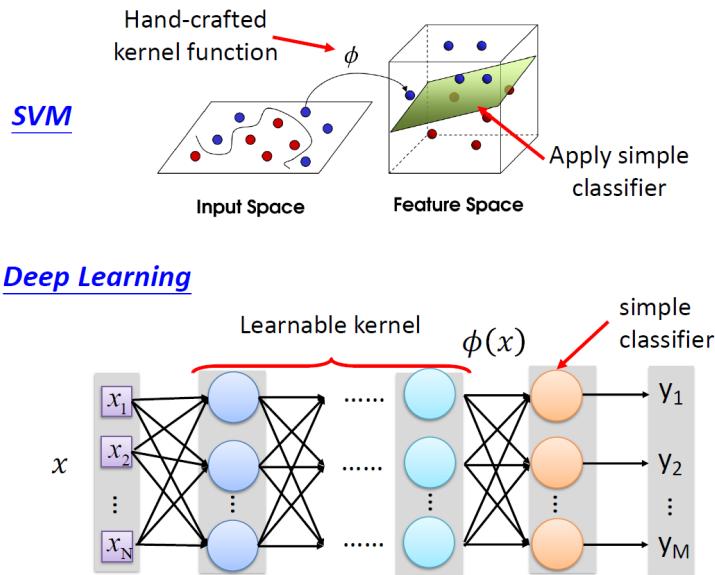
1 <sub>x1</sub>	1 <sub>x0</sub>	1 <sub>x1</sub>	0	0
0 <sub>x0</sub>	1 <sub>x1</sub>	1 <sub>x0</sub>	1	0
0 <sub>x1</sub>	0 <sub>x0</sub>	1 <sub>x1</sub>	1	1
0	0	1	1	0
0	1	1	0	0

Image
Convolved Feature

**Pooling**: 采样像素不会改变图片的形状等，我们可以对像素进行采样，使图像更小，因此能用更少的参数来表征图像。进一步降低复杂度。

**Flattening**: 将多层 channel 压平后，作为神经网络的输入。

**SVM** 与神经网络的对比：



## 15 无监督学习

### 15.1 PCA

适用情况:  $y$ 为连续值。

降维是指将原始高维数据映射到低维空间。

降维标准可以根据不同的问题设置而有所不同,如无监督下是最小化信息丢失; 监督学习下最大化类别差异。

**降维原因:** 解决维度灾难; 随着维数的增加, 查询的准确性和效率会迅速降低; 本征维数(能够真实反映特征的维度)可能很小; 数据压缩; 去噪; 可视化。

**问题形式化:** 给定数据 $\{x_1, x_2, \dots, x_n\}$ , 计算线性映射

$$P \in R^{d \times m} : \mathbf{x} \in R^d \rightarrow \mathbf{y} = P^\top \mathbf{x} \in R^m (m \ll d)$$

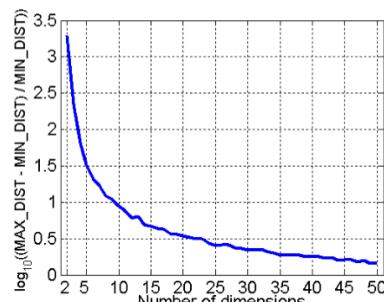
**Curse of Dimensionality :**

When dimensionality increases, data becomes increasingly sparse in the space that it occupies

Definitions of density and distance between points, which is critical for clustering and outlier detection, become less meaningful

If  $N_1 = 100$  represents a dense sample for a single input problem, then  $N_{10} = 100^{10}$  is the sample size required for the same sampling density with dimension 10.

The proportion of a hypersphere with radius  $r$  and dimension  $d$ , to that of a hypercube with sides of length  $2r$  and dimension  $d$  converges to 0 as  $d$  goes to infinity — nearly all of the high-dimensional space is “far away” from the center



- Randomly generate 500 points

- Compute difference between max and min distance between any pair of points

**思想：**选择一条符合数据的线，使得点沿着线的方向展开，即最小化到直线距离的平方和。最小化到这条直线的距离平方和等于最大化这条直线上投影的平方和。

**适用情况：**y为离散标签。

**步骤：**

- Calculate the covariance matrix  $S$

$$S = \frac{1}{n} \sum_{i=1}^n (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^\top$$

or first center the data:  $\{x'_1, x'_2, \dots, x'_n\}$  and  $\bar{x}' = 0$  let  $X = [\mathbf{x}'_1, \mathbf{x}'_2, \dots, \mathbf{x}'_n] \in R^{d \times n}$ ;

then  $S = \frac{1}{n} X X^\top$

- Find the first  $m$  eigenvectors

$$\{\mathbf{u}_i\}_{i=1}^m$$

- Form the projection matrix  $P = [\mathbf{u}_1 \mathbf{u}_2 \dots \mathbf{u}_m] \in R^{d \times m}$

- A new test point can be projected as:

$$\mathbf{x}_{\text{new}} \in R^d \rightarrow P^\top \mathbf{x}_{\text{new}} \in R^m$$

**数据还原：** If  $P$  is a square matrix, we can recover  $x$  by

$$\mathbf{x} = (P^\top)^{-1} \mathbf{y} = P \mathbf{y} = P P^\top \mathbf{x}$$

Here  $P$  is not full, but we can still recover  $x$  by  $x = P \mathbf{y} = P P^\top x$ , and lose some information

**结果：**PCA 保留了样本的大部分信息。被称为主成分(PCs)的新变量是不相关的，并按每个变量所保留的总信息的比例排序。事实上，PCA 投影使所有结果维度为  $m$  的线性投影的重构误差最小。

$$\begin{aligned}
 \arg \min_{P \in R^{d \times m}} \|X - X'\|^2 &= \arg \min_{P \in R^{d \times m}} \|X - P P^\top X\|^2 \\
 &\stackrel{\substack{\uparrow \\ \text{Reconstruction error}}}{=} \arg \max_{P \in R^{d \times m}} \text{trace}(X^\top P P^\top X) \\
 &= \arg \max_{P \in R^{d \times m}} \text{trace}(P^\top X X^\top P) \\
 &= \arg \max_{P \in R^{d \times m}} \text{trace}(P^\top S P) \\
 \text{subject to} \quad P^\top P &= I_m
 \end{aligned}$$

**Kernel PCA:**

$$S\mathbf{u} = \frac{1}{n} \sum_i \mathbf{x}_i \mathbf{x}_i^T \mathbf{u} = \lambda \mathbf{u} \Rightarrow \mathbf{u} = \frac{1}{n\lambda} \sum_i (\mathbf{x}_i^T \mathbf{u}) \mathbf{x}_i$$

The covariance matrix can be written in matrix form:

$$S = \frac{1}{n} XX^T, \text{ where } X = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n].$$

$$\mathbf{u} = \sum_i \alpha_i \mathbf{x}_i = X\mathbf{a} \quad S\mathbf{u} = \frac{1}{n} XX^T X\mathbf{a} = \lambda X\mathbf{a}$$

$$\frac{1}{n} (X^T X)(X^T X)\mathbf{a} = \lambda (X^T X)\mathbf{a}$$

**Any benefits?**

$$\xrightarrow{\hspace{1cm}} \boxed{\frac{1}{n} (X^T X)\mathbf{a} = \lambda \mathbf{a}} \xrightarrow{\hspace{1cm}} \boxed{\frac{1}{n} K\mathbf{a} = \lambda \mathbf{a}}$$

$$\mathbf{u}^T \phi(\mathbf{x}') = \sum_i \alpha_i \phi(\mathbf{x}_i^T) \phi(\mathbf{x}') = \sum_i \alpha_i K(\mathbf{x}_i, \mathbf{x}')$$

36

优势：PCA（作为降维技术）试图找到数据所局限于的低维线性子空间。但是可能是数据被限制在低维非线性子空间中。那会发生什么呢？考虑一个二维的类似于抛物线的曲线，数据点大部分位于2D曲线上，但PCA不能将维数从2减小到1，因为这些点不在直线上。但是，数据仍然“显然”位于一维非线性曲线周围。因此，当PCA失败时，必须有另一种方法！实际上，内核PCA可以找到该非线性流形并发现数据实际上几乎是一维的。“内核技巧”的本质是，实际上并不需要明确考虑高维空间，因此，这种潜在的令人迷惑的维数跃迁完全是秘密执行的。

## 15.2 聚类

思想：降低类间相似性，提高类内相似性。

步骤：

优化损失函数  $\text{Loss}(\mu, C)$

$$\min_{\mu} \min_C \sum_{j=1}^n \|\mu_{C(j)} - \mathbf{x}_j\|^2$$

(1) Fix  $\mu$ , optimize  $C$

$$\min_{C(1), C(2), \dots, C(n)} \sum_{j=1}^{-n} \|\mu_{C(j)} - \mathbf{x}_j\|^2$$

Assign each point to the nearest cluster center

(2) Fix  $C$ , optimize  $\mu$

$$\min_{\mu(1), \mu(2), \dots, \mu(k)} \sum_{j=1}^n \|\mu_{C(j)} - \mathbf{x}_j\|^2$$

Solution: average of points in cluster  $i$ , exactly second step (re-center)

关于KMeans的一些证明：

定理一、设  $x_1, x_2 \dots x_n$  是欧式空间的  $n$  个向量，则  $\sum_{i=1}^n (x_i - x)'(x_i - x)$  取到最小值当且仅当  $x$  是这  $n$  个向量的中心位置，即  $x = \frac{1}{n} \sum_{i=1}^n x_i$

证明、

$$\frac{\partial \sum_{i=1}^n (x_i - x)'(x_i - x)}{\partial x} = -2(x_i - x) = 0$$

$x = \frac{1}{n} \sum_{i=1}^n x_i$  是该函数的一个驻点。显然该目标函数是严格凸的，所以  $x = \frac{1}{n} \sum_{i=1}^n x_i$  是目标

函数的唯一最小值点。这就说明，只要步骤4中有某个中心位置发生了变化，那么目标函数值严格较小。综上所述，从步骤2到步骤4，经过一次迭代后，目标函数不增，即

$$\rho(\omega'_i) = \sum_{i=1}^n (x_i - x^{**})'(x_i - x^{**}) \leq \sum_{i=1}^n (x_i - x')'(x_i - x') \leq \sum_{i=1}^n (x_i - x^*)'(x_i - x^*) = \rho(\omega_i)$$

经过这样一次处理后，目标函数的值是递减的，且只要在该过程中中心位置发生了变化，则

$$\rho(\omega'_i) < \rho(\omega_i)$$

即不等号严格成立，目标函数严格减小。

## 二、k-means聚类算法的收敛性证明

定理二、对于任意给定的迭代聚类中心初值（或者任意给定的一种划分方式）， $k-means$  算法的目标函数一定会收敛。

证明、将目标函数记为  $f(T)$ ，其中  $T$  是对给定数据集的一种划分方式，例如划分  $T_1$  是将数据集划分成  $\{\omega_1, \omega_2 \dots \omega_k\}$  这  $k$  个互不相交的集合，则

$$f(T_1) = \sum_{i=1}^k \rho(\omega_i) = \sum_{j=1}^n (x - x^*)'(x - x^*)$$

显然对于任何划分方式  $T$ ， $f(T) \geq 0$ 。对于从任意一个初始方式开始，不断进行迭代，就会得到对数据集的一列划分： $T_1, T_2, T_3 \dots$ ，同时对应地得到一列目标函数值  $f(T_1), f(T_2), f(T_3) \dots$ 。由前文所述，该数列  $\{f(T_n)\}$  单调递减且有下界 0，由单调有界数列的收敛定理知， $\{f(T_n)\}$  收敛，即

$$\lim_{n \rightarrow \infty} f(T_n)$$

存在，这就意味着，随着算法迭代次数增加，目标函数一定会收敛。

定理三、随着迭代次数趋向无穷， $k$  个中心点必然会收敛，即对于每种确定的初值选取方式，聚类结果是唯一确定的。

若将第  $k$  次迭代得到的中心点的位置记为  $c_n^1, c_n^2 \dots c_n^k$ 。现要证明对于任意给定的迭代聚类中心初值（或者任意给定的一种初始划分方式），对任意固定的  $p$  ( $p \in N^+, 1 \leq p \leq k$ )

$$\lim_{n \rightarrow \infty} c_n^p$$

证明、对于一个给定的数据集来说，元素个数是有限的，所以总的划分方式是有限的，所以  $f(T_n)$  只有有限个不同的取值。所以，随着  $n$  的增大， $f(T_n)$  不可能一直严格递减，即严格递减过程必然要中止，即存在  $N$ ，当  $n > N$  时， $f(T_n)$  恒为常数。由定理一可知，当  $n > N$  时， $c_n^1, c_n^2 \dots c_n^k$  恒为常数，不会再改变（因为中心点只要有一个改变，那么  $f(T_n)$  必然严格递减，与  $f(T_n)$  恒为常数矛盾）。所以对任意固定的  $p$  ( $p \in N^+, 1 \leq p \leq k$ )，  
 $\lim_{n \rightarrow \infty} c_n^p$  存在。证毕。

推论、从证明过程中可以看出，不仅仅可以证明对于任意一个初值给定方式，目标函数  $f(T_n)$  与中心点  $c_n^1, c_n^2 \dots c_n^k$  都会收敛，而且可以得到更强的结论：经过有限次迭代后，目标函数值与中心点位置都会恒为常量，即该算法迭代过程不是一直无限逼近极小值点的过程，而是经过有限步逼近后，必然会严格等于极小值点，此后再进行迭代，划分方法、中心点、目标函数等都不会再改变。由于对于给定的初值，每一步的过程是完全确定的，不含随机因素。所以对于给定初值，聚类结果是唯一确定的。

注、虽然对于给定的初值，算法可以保证收敛，但是对于不同的初值选取情况，算法收敛到的结果可能是不一样的。显然，对于不同的类别数  $k$ ，聚类结果必然不同。所以初始中心位置（或初始划分方式）与类别数  $k$  是该算法需要调节的超参数。

## 如何快速收敛超大的 KMeans? (来自网络)

1. 第一次迭代的时候，正常进行，选取  $K$  个初始点，然后计算所有节点到这些  $K$  的距离，再分到不同的组，计算新的质心；

2. 后续迭代的时候，在第  $m$  次开始，每次不再计算每个点到所有  $K$  个质心的距离，仅仅计算上一次迭代中离这个节点最近的某几个（2 到 3）个质心的距离，决定分组的归属。对于其他的质心，因为距离实在太远，所以归属到那些组的可能性会非常非常小，所以不用再重复计算距离了。

3. 最后，还是用正常的迭代终止方法，结束迭代。

这个方法中，有几个地方需要仔细定义的。

第一，如何选择  $m$  次？过早的话，后面的那个归属到远距离组的可能性会增加；过晚，则收敛的速度不够。

第二，如何选择最后要比较的那几个质心点数？数量过多则收敛的速度提高不明显，过少则还是有可能出现分组错误。

这两个问题应该都没有标准答案，就如同  $K$  值的选取。我自己思考的基本思路可以是：

1. 从第三次开始就开始比较每次每个质心的偏移量，亦即对于收敛的标准可以划分两个阈值，接近优化的阈值（比如偏移范围在 20%）和结束收敛的阈值（比如偏移范围在 10% 以内）。 $m$  次的选择可以从达到接近优化的阈值开始。

2. 选择比较的质心点数可以设定一个阈值，比较一个点到  $K$  个质心的距离，排序这些距离，或者固定选取一个数值，比如 3 个最近的点，或者按最近的 20% 那些质心点。