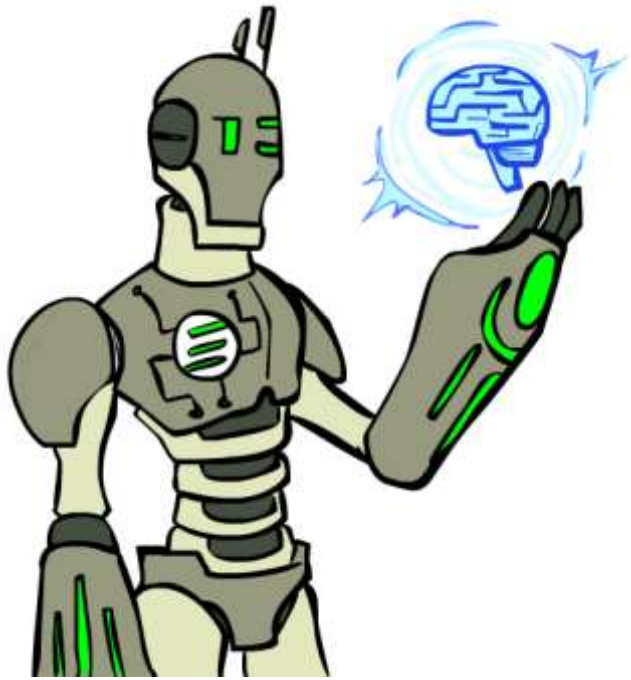
人工智能 I

本节课安排

- 上节内容回顾

- 约束满足问题
(Constraints Satisfaction Problems, CSP)

- CSP的解决方案
  - 回溯搜索
  - 改进
    - 预处理
    - 排序
    - 推理
    - 结构

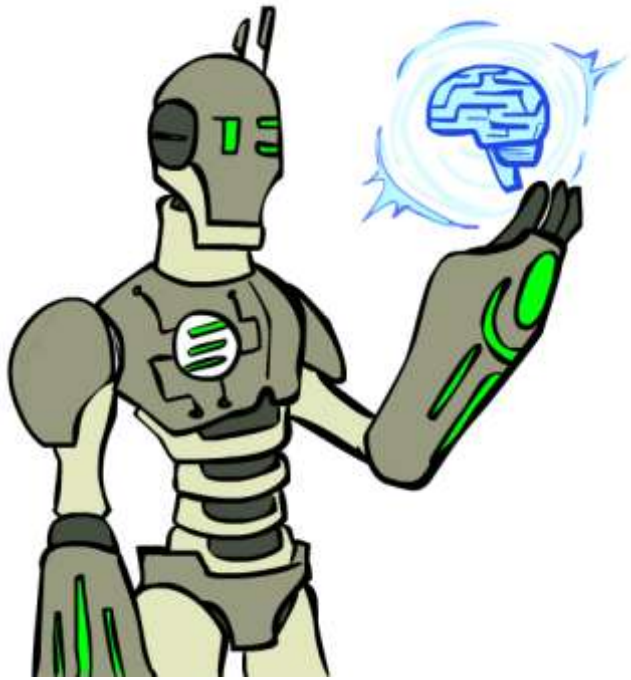


本节课安排

- 上节内容回顾

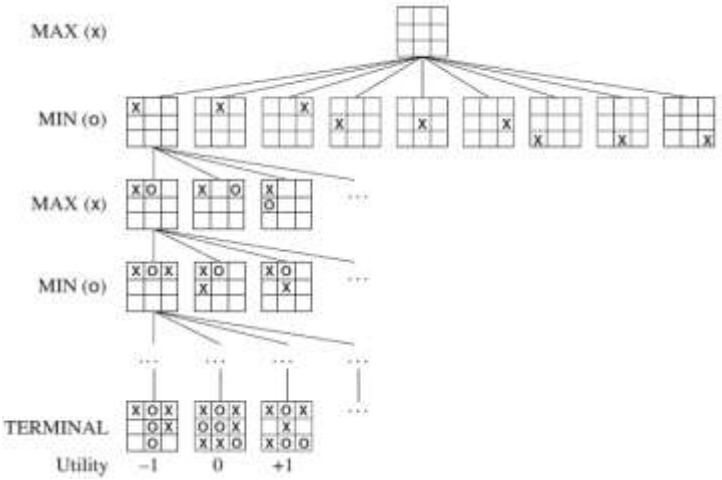
- 约束满足问题
(Constraints Satisfaction Problems, CSP)

- CSP的解决方案
  - 回溯搜索
  - 改进
    - 预处理
    - 排序
    - 推理
    - 结构



要点

- 零和博弈（两玩家）问题的建模
  - 比搜索问题多出：players(s), utility(s)
- 博弈树
  - 根节点：初始状态，行为决策节点
  - 叶子节点：终止状态（终局）
  - 与搜索树类似，主要不同点：
    - 博弈树：有MAX与MIN两位玩家，深度有限



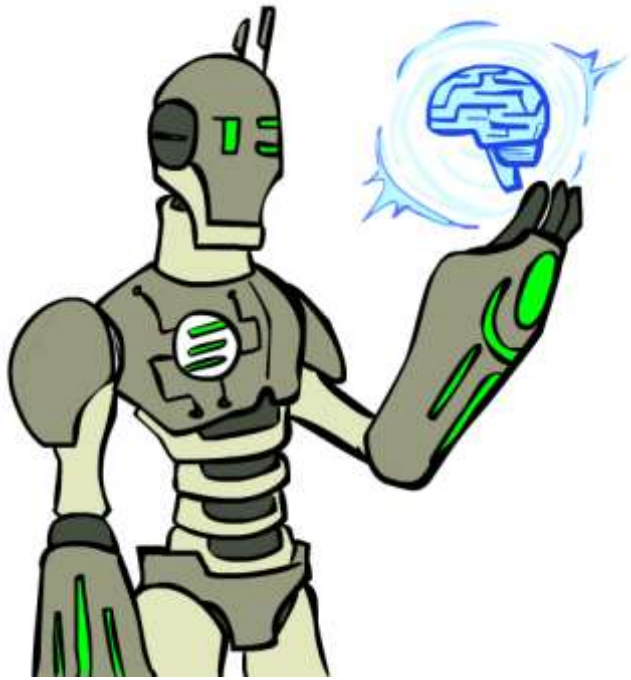
- 极小极大搜索、α-β 剪枝

本节课安排

- 上节内容回顾

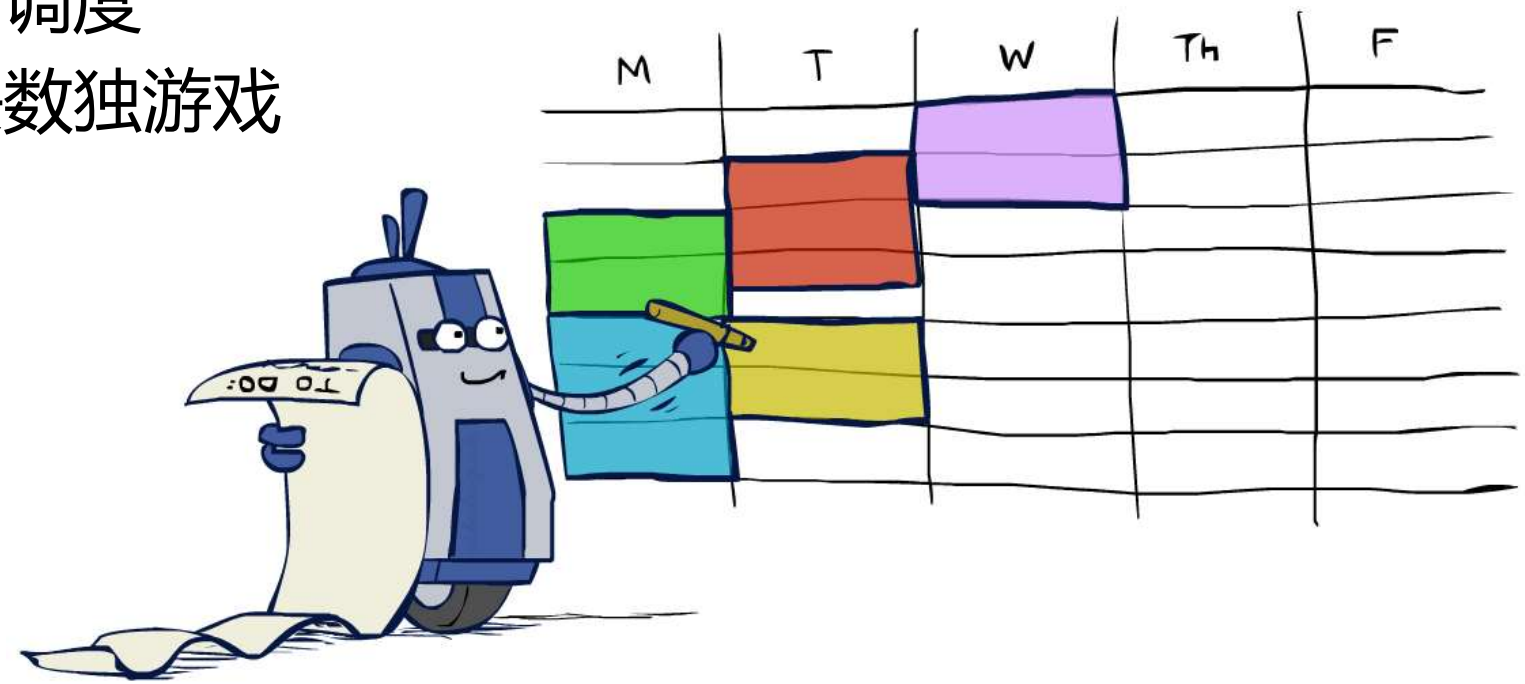
- 约束满足问题
(Constraints Satisfaction Problems, CSP)

- CSP的解决方案
  - 回溯搜索
  - 改进
    - 预处理
    - 排序
    - 推理
    - 结构



生活中的约束满足问题

- 安排会议 e.g.: 班会?
- 时间表安排问题 e.g.: 何时何地开设哪个课程?
- 规划旅行行程
- 交通调度
- 工厂调度
- 解决数独游戏

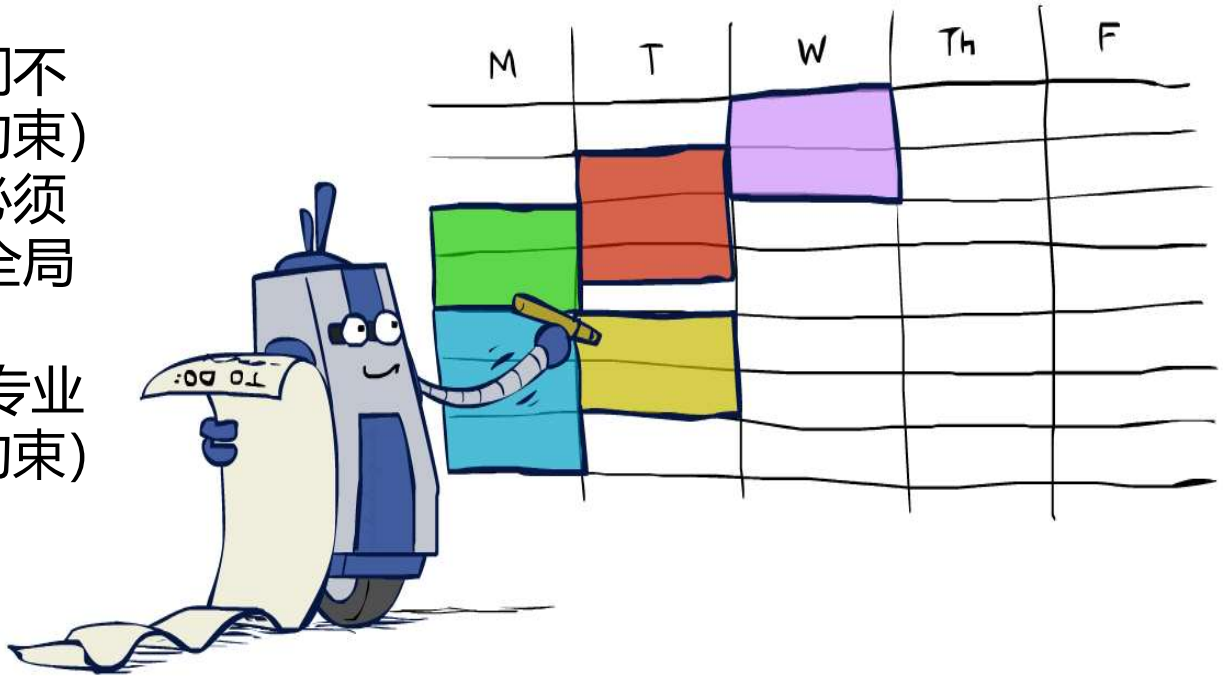


生活中的约束满足问题

学生选课

- 小明的任务就是为CourseA，CourseB，CourseC这三个变量从值域中分别赋值（即选择具体的课程），使得所有约束同时得到满足。
- 变量: CourseA, CourseB, CourseC
- 值域: 所有本学期开设的课程集合
- 约束:

① 三门课的上课时间不能冲突。（全局约束）
② 三门课的总学分必须达到10学分。（全局约束）
③ CourseA 必须是专业必修课。（一元约束）



约束满足问题(CSP)的描述

- $\mathrm{CSP} = \{\mathbf{X}, \mathbf{D}, \mathbf{C}\}$
- 变量集：$\mathbf{X} = \{X_1, X_2, \cdots\cdots, X_n\}$
- 值域集：$\mathbf{D} = \{D_1, D_2, \cdots\cdots, D_n\}$, D_i为变量X_i的取值范围
- 约束集：$\mathbf{C} = \{C_1, C_2, \cdots\cdots, C_k\}$

地图着色问题

$\mathbf{X} = \{WA, NT, SA, Q, NSW, V, T\}$

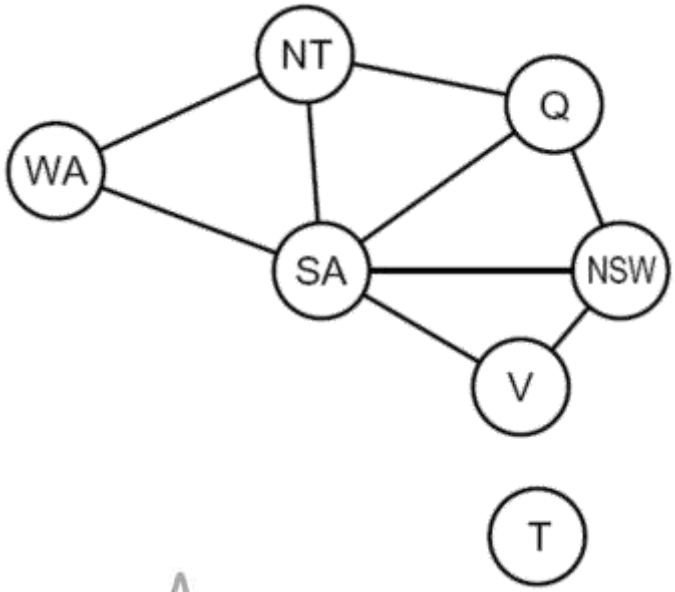
$\mathbf{D} = \{\mathrm{r}, \mathrm{g}, \mathrm{b}\}$

$\mathbf{C} = \{SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V, WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V\}$

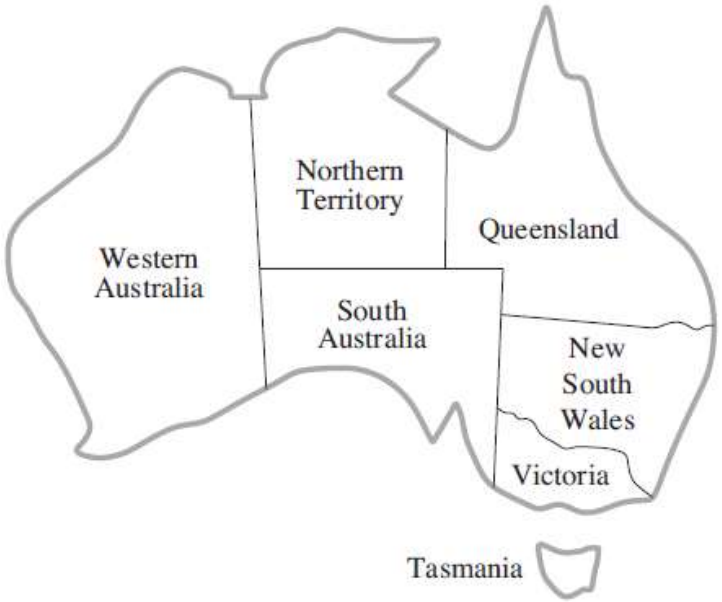
$C_1: \langle (SA, WA), SA \neq WA \rangle$
简写$SA \neq WA$

约束满足问题(CSP)的描述

- 约束图



- 节点：变量
- 边：两个变量之间存在约束关系
  - ✓ 约束条件需额外说明



$$\mathbf{X} = \{WA, NT, SA, Q, NSW, V, T\}$$

$$\mathbf{D} = \{\mathrm{r}, \mathrm{g}, \mathrm{b}\}$$

$$\mathbf{C} = \{SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V, WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V\}$$

约束图

- 变量

  $F\ T\ U\ W\ R\ O\ X_1\ X_2\ X_3$

- 值域

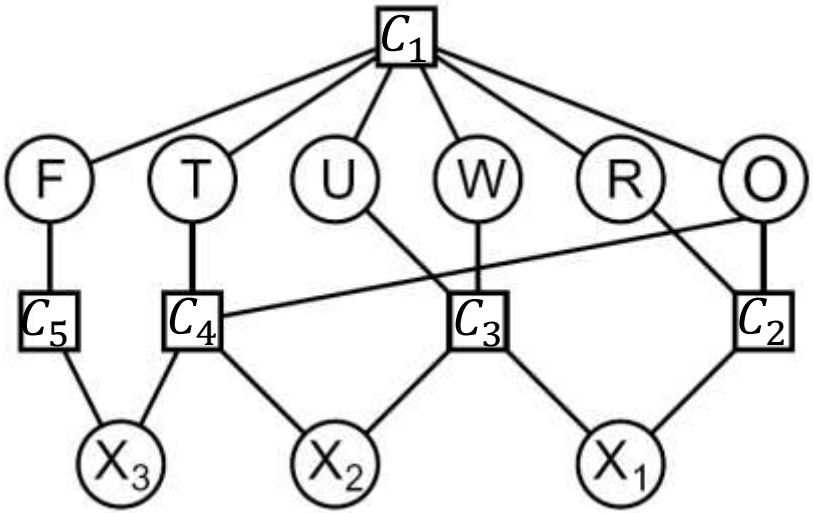
  $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

- 约束

C_1: alldiff(F, T, U, W, R, O)
C_2: $O + O = R + 10X_1$
C_3: $X_1 + W + W = U + 10X_2$
C_4: $X_2 + T + T = O + 10X_3$
C_5: $\mathrm{F} = X_3$

算术密码

```
  T W O
+ T W O
-------
F O U R
```



约束图

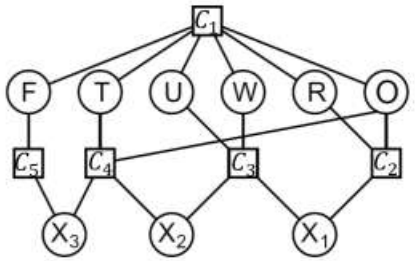
- 变量

  $F\ T\ U\ W\ R\ O\ X_1\ X_2\ X_3$

- 值域

  $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

- 约束



C_1: alldiff(F, T, U, W, R, O)
C_2: $O + O = R + 10X_1$
C_3: $X_1 + W + W = U + 10X_2$
C_4: $X_2 + T + T = O + 10X_3$
C_5: $\mathrm{F} = X_3$

算术密码

```
  T W O
+ T W O
-------
F O U R
```

```
  7 3 4
+ 7 3 4
-------
1 4 6 8
```

约束满足问题(CSP)的解

- 一组不违反任何约束的对（全部或部分）变量的赋值称为相容赋值（或合法赋值）

$$\{\mathrm{WA} = \mathrm{r}, \mathrm{NT} = \mathrm{g}\}$$

- 对全部变量的赋值称为完全赋值

$$\{\mathrm{WA} = \mathrm{r}, \mathrm{NT} = \mathrm{r}, \mathrm{Q} = \mathrm{r},$$
$$\mathrm{NSW} = \mathrm{r}, \mathrm{V} = \mathrm{r}, \mathrm{SA} = \mathrm{r}, \mathrm{T} = \mathrm{r}\}$$

- 一组既是相容赋值又是完全赋值的赋值成为CSP的一个解

$$\{\mathrm{WA} = \mathrm{red}, \mathrm{NT} = \mathrm{green}, \mathrm{Q} = \mathrm{red},$$
$$\mathrm{NSW} = \mathrm{green}, \mathrm{V} = \mathrm{red}, \mathrm{SA} = \mathrm{blue}, \mathrm{T} = \mathrm{green}\}$$

CSP的种类

- 离散值域
  - 有限值域

    如：$D = \{red, green, blue\}$
  - 无限值域（整数、字符串等）
    - 无法用枚举描述值域，只能用约束语言

    如：$T_1 + d_1 \leq T_2$

    线性约束可解

    非线性约束无通用解法

- 连续值域
  - 线性约束可使用线性规划方法求解
  - 最优化课程

约束的种类

- 一元(Unary)约束
  - 约束只限制单个变量的取值，e.g.: $F \neq 0$
- 二元(Binary)约束
  - 约束与两个变量有关，e.g.: $F = X_3$
- 高阶(Higher-order)约束
  - 约束三个及以上变量，e.g:

$$\text{alldiff}(F, T, U, W, R, O)$$

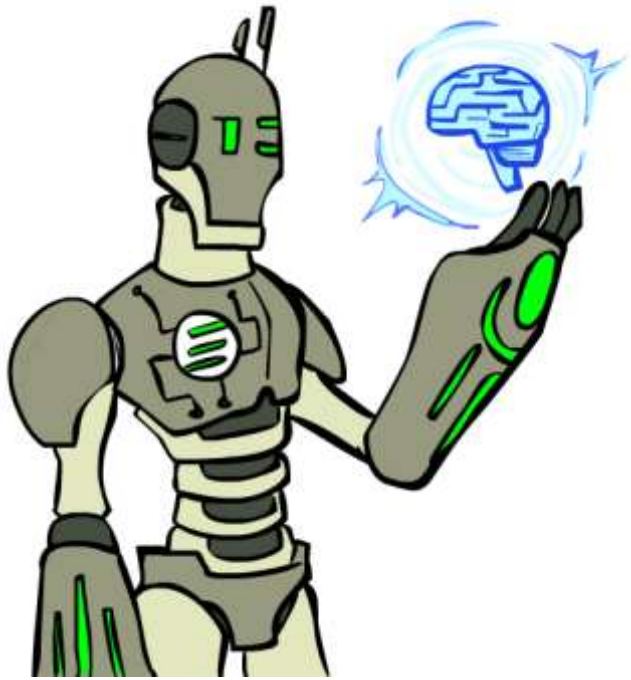
  - 可转化为二元约束
- 软(Soft)约束
  - 尽可能多地满足软约束，e.g.，红色比绿色更好
  - 通常用每个变量赋值的代价来表示
  - 约束优化问题

本节课安排

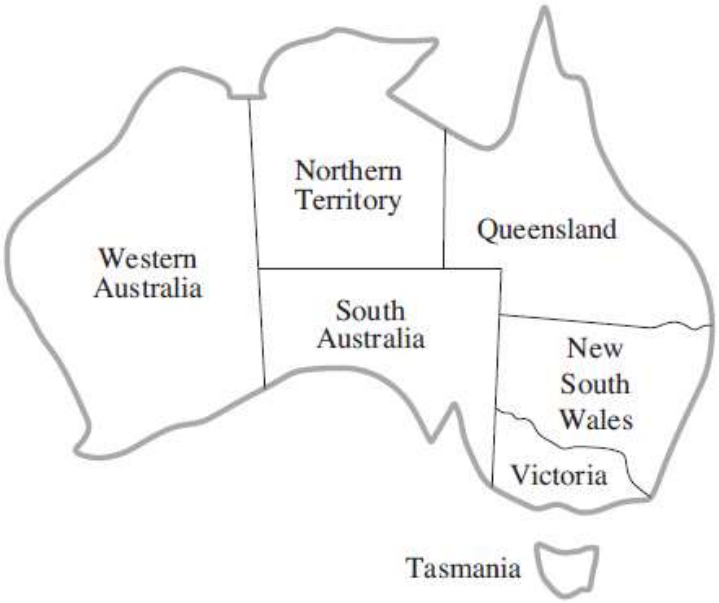
- 上节内容回顾

- 约束满足问题
(Constraints Satisfaction Problems, CSP)

- CSP的解决方案
  - 回溯搜索
  - 改进
    - 预处理
    - 排序
    - 推理
    - 结构



如何求解CSP?



$$X = \{WA, NT, SA, Q, NSW, V, T\}$$

$$D = \{\mathrm{r}, \mathrm{g}, \mathrm{b}\}$$

$$C = \{SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V, WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V\}$$

- 枚举法：尝试所有组合
  - 上例中，需尝试$3^7 = 2178$种组合
  - 若有n个变量，每个变量可取值数不大于d

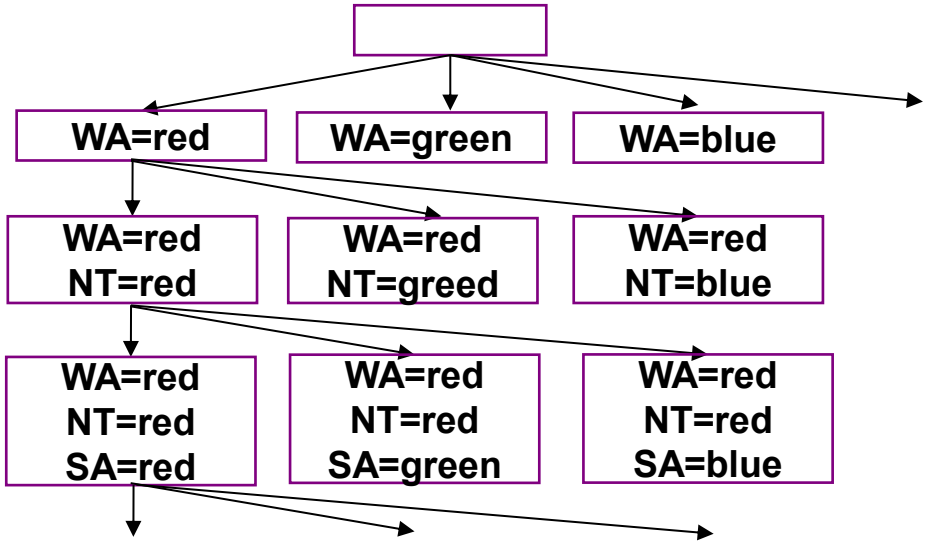
    计算复杂度：$\mathcal{O}(d^n)$

CSP是NP完全问题，但仍可设计出实际计算效率较高的算法

基于搜索的求解方法

- 把CSP看作路径规划问题
  - 初始状态：空赋值集合，所有的变量都是未赋值的
  - 后继函数：给未赋值的变量一个赋值，进入另一状态
  - 目标测试：测试完全赋值（叶节点）是否相容
  - 步骤成本：每步成本均为常数（1）

DFS　　BFS　　UCS　　greedy　　A*



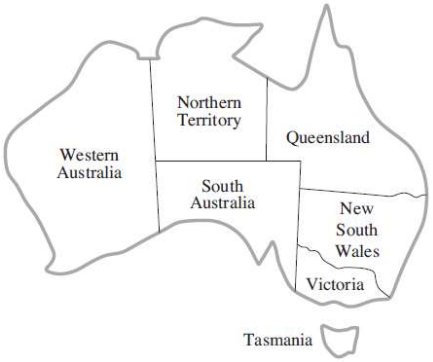
本课程只讲授基于DFS的改进方法

回溯搜索

- 方法
  - 在搜索过程中，一旦发现赋值是不相容的，就退回一步重新赋值（回溯），继续向前搜索，如此反复进行，直至得到解或证明无解。
- 1：每次只考虑单个变量的赋值，即在搜索树的每个结点上只考虑单个变量的合法赋值
  - 变量赋值具有可交换性，e.g.:

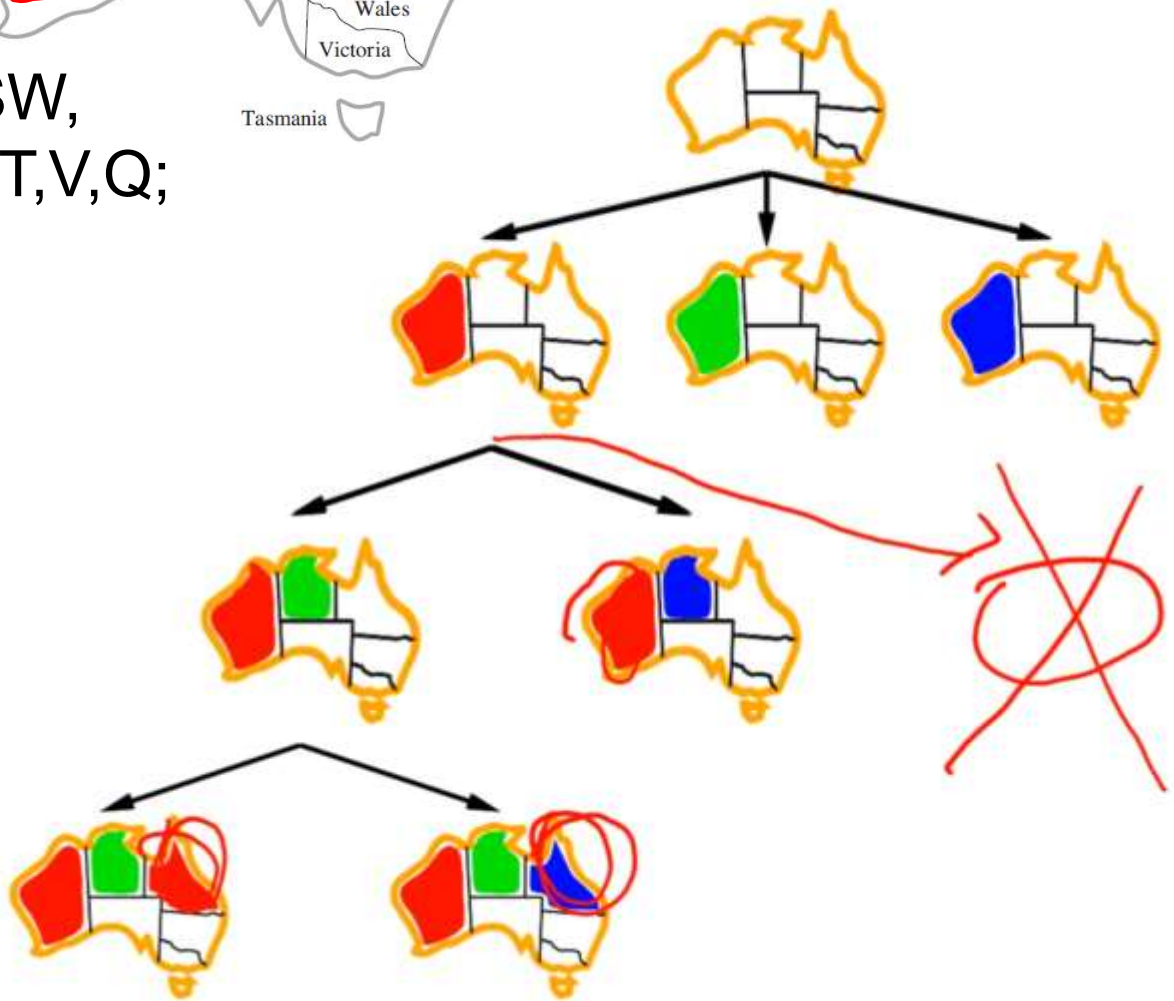
  {WA=r，NT=g}等价于{NT=g, WA=r}
- 2：提前检查约束
  - 检查与之前赋值是否相容
  - 对非叶子结点做“排除性”目标测试

WA=red
NT=red



回溯搜索

WA,NSW,
SA,NT,T,V,Q;
r, g, b



回溯搜索

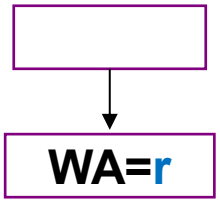
```
function BACKTRACKING-SEARCH(csp) returns solution/failure
    return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
    if assignment is complete then return assignment
①   var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
②   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
③           result ← RECURSIVE-BACKTRACKING(assignment, csp)
            if result ≠ failure then return result
④           remove {var = value} from assignment
    return failure
```

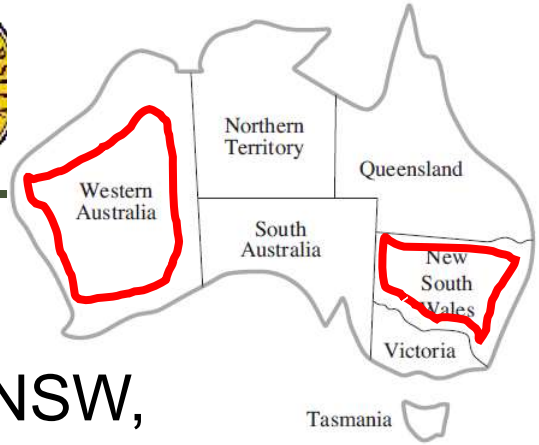
① 未赋值变量的选择顺序能影响搜索效率
② 值的排序能影响搜索效率
③ 递归调用（深度优先搜索）
④ 删除当前赋值，下次迭代将测试值域中的下一个值。当该变量所有可选值都不相容时，发生“回溯”（回到上一层）

回溯搜索

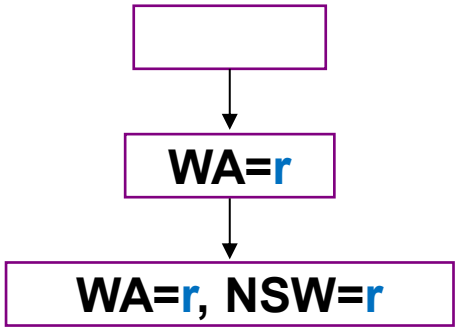
WA,NSW,
SA,NT,T,V,Q;
r, g, b



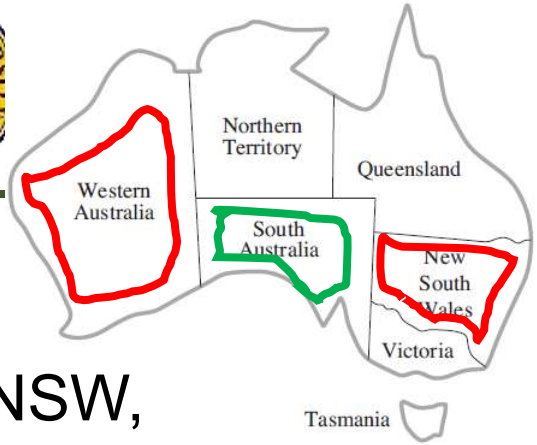
回溯搜索



WA,NSW,
SA,NT,T,V,Q;
r, g, b

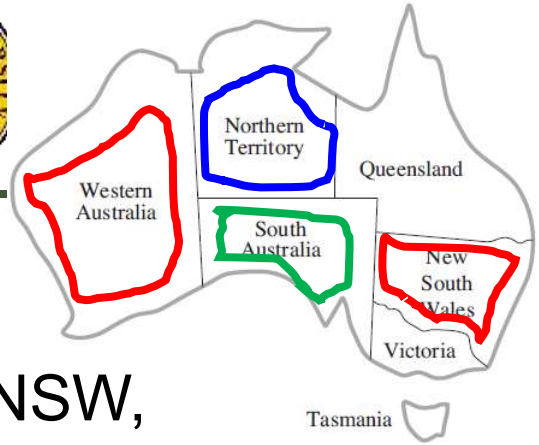


回溯搜索



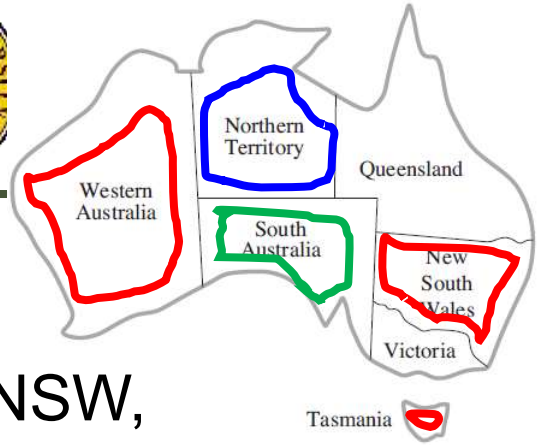
WA,NSW,
SA,NT,T,V,Q;
r, g, b

回溯搜索



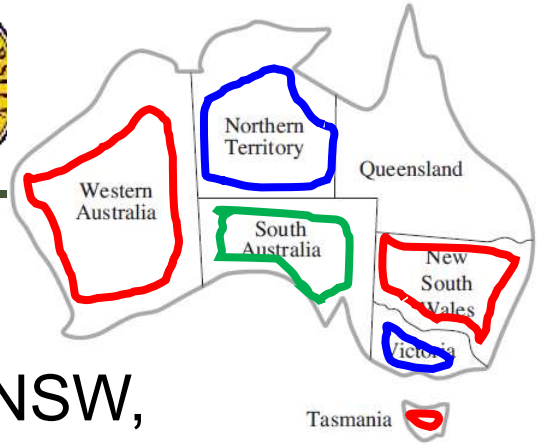
WA,NSW,
SA,NT,T,V,Q;
r, g, b

回溯搜索



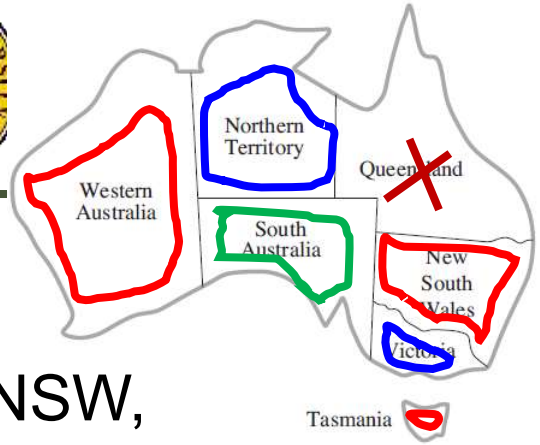
WA,NSW,
SA,NT,T,V,Q;
r, g, b

回溯搜索



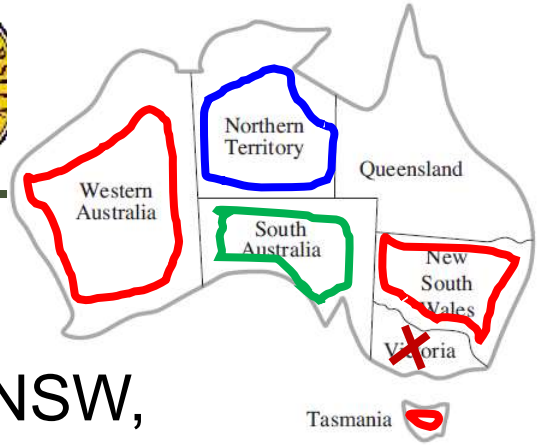
WA,NSW,
SA,NT,T,V,Q;
r, g, b

回溯搜索



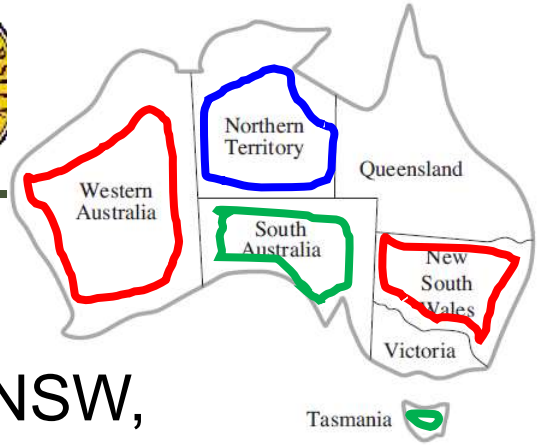
WA,NSW,
SA,NT,T,V,Q;
r, g, b

回溯搜索



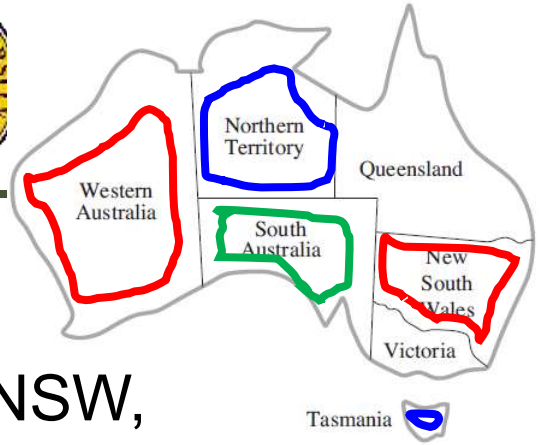
WA,NSW,
SA,NT,T,V,Q;
r, g, b

回溯搜索



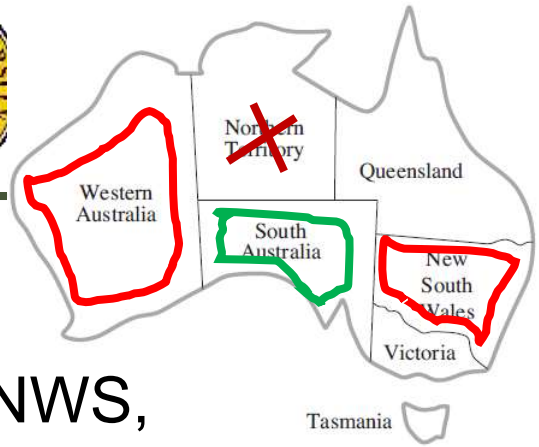
WA,NSW,
SA,NT,T,V,Q;
r, g, b

回溯搜索



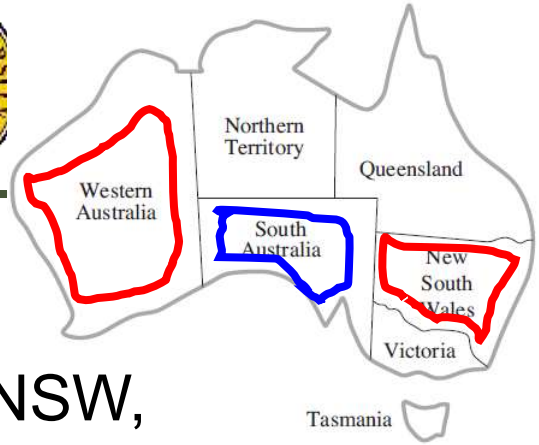
WA,NSW,
SA,NT,T,V,Q;
r, g, b

回溯搜索



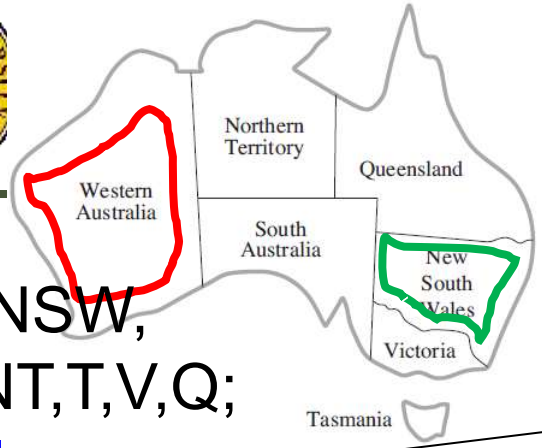
WA,NWS,
SA,NT,T,V,Q;
r, g, b

回溯搜索

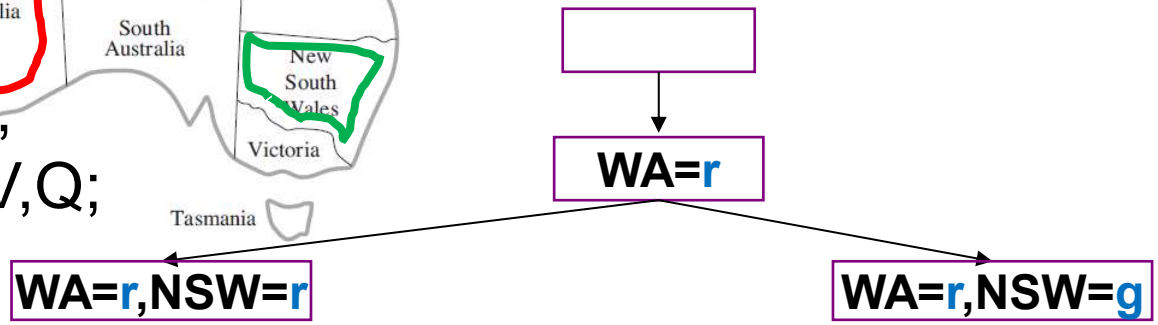


WA,NSW,
SA,NT,T,V,Q;
r, g, b

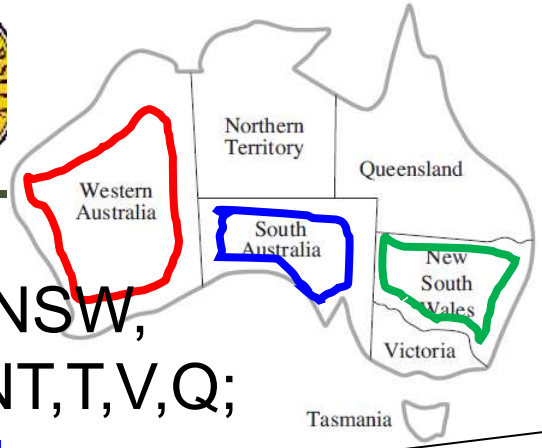
回溯搜索



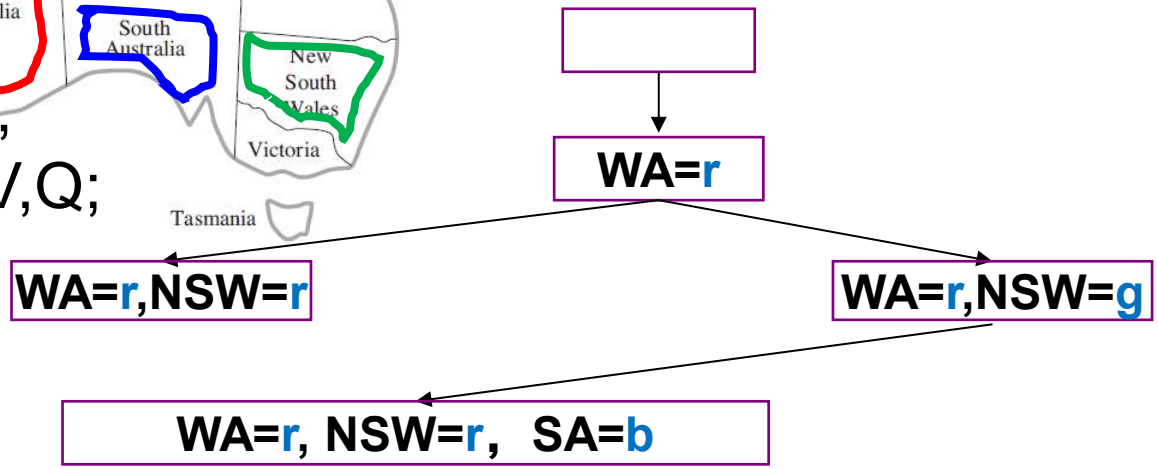
WA,NSW,
SA,NT,T,V,Q;
r, g, b



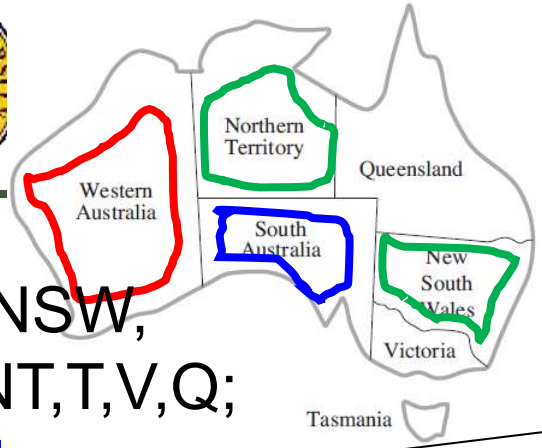
回溯搜索



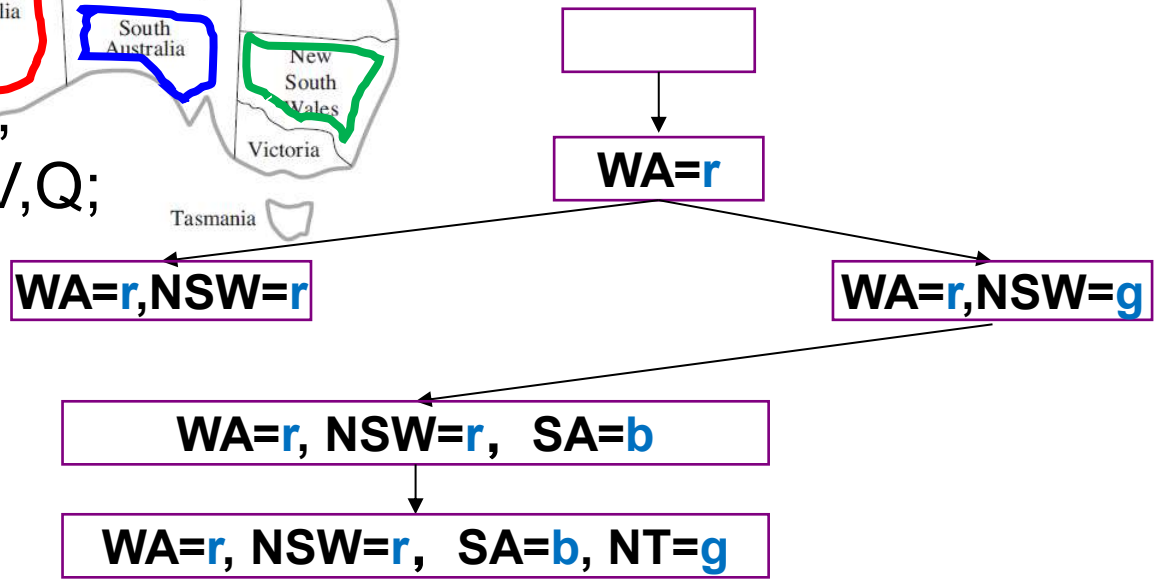
WA,NSW,
SA,NT,T,V,Q;
r, g, b



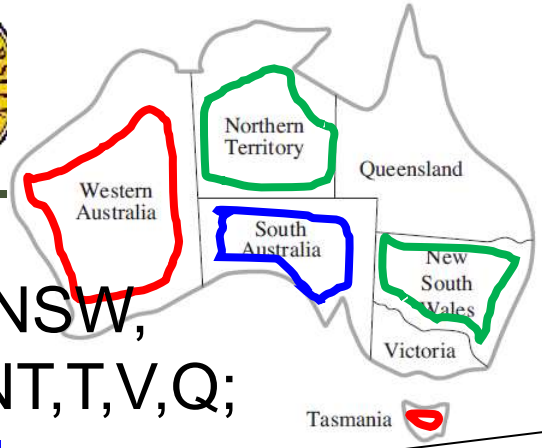
回溯搜索



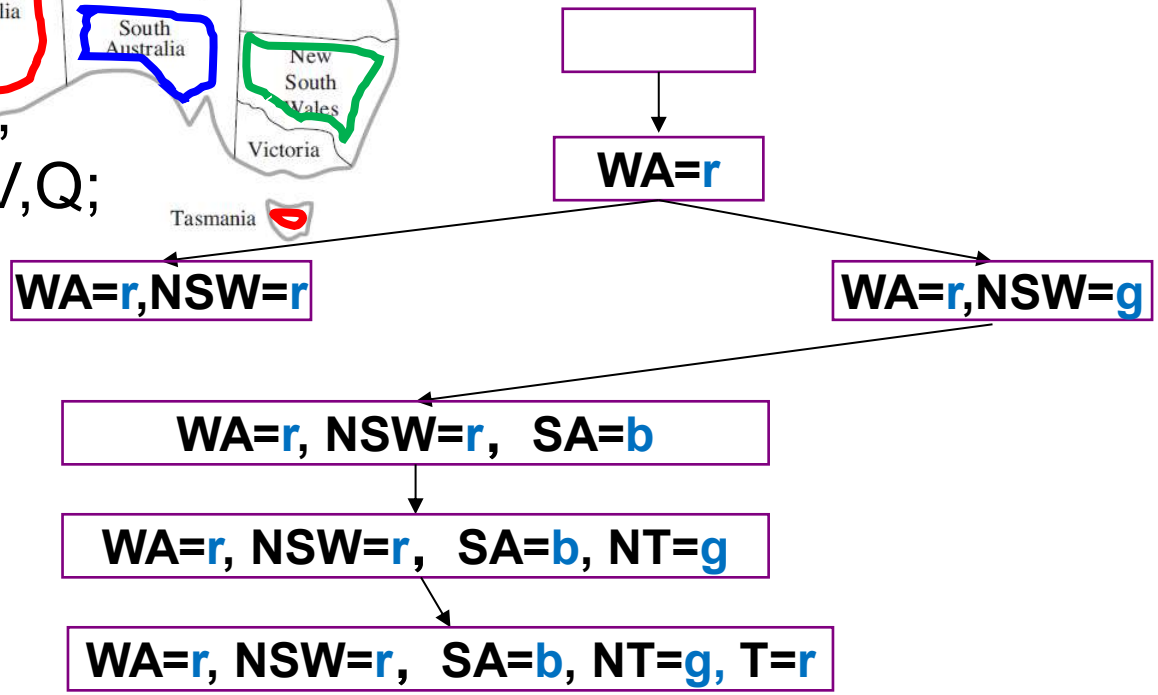
WA,NSW,
SA,NT,T,V,Q;
r, g, b



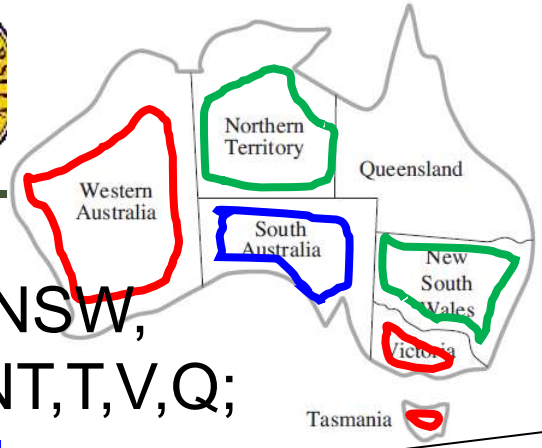
回溯搜索



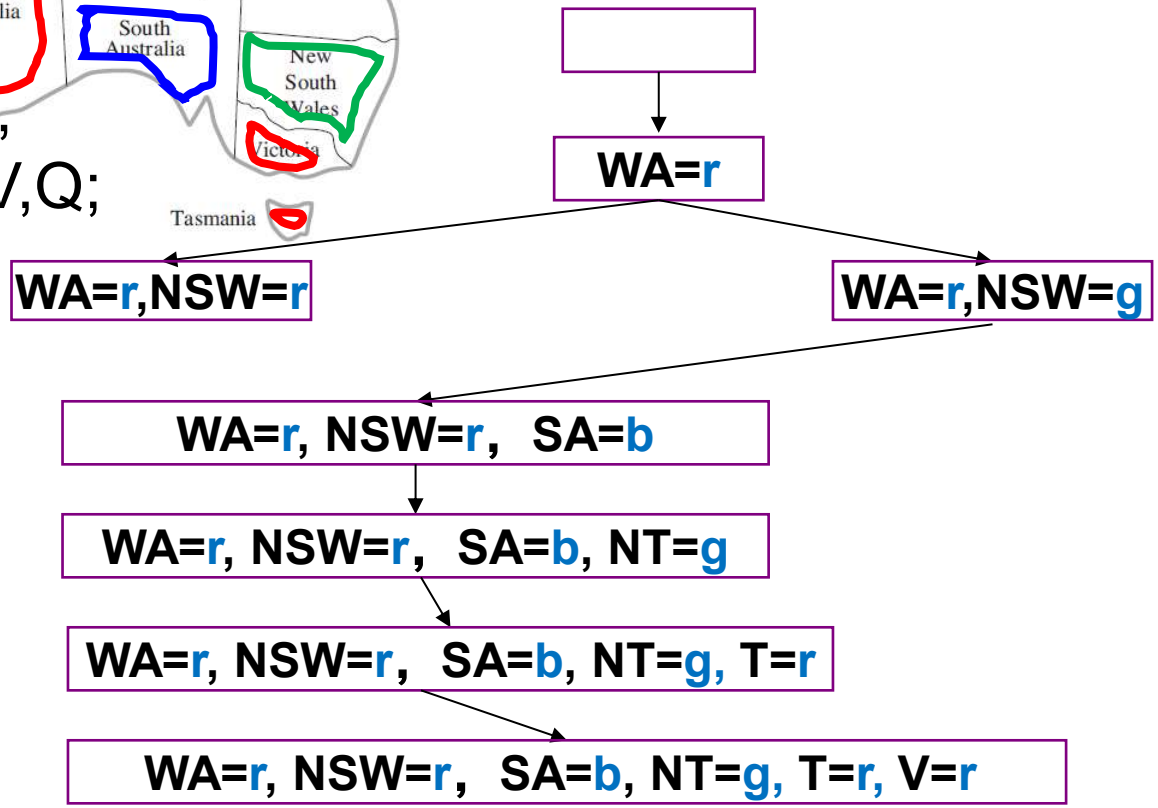
WA,NSW,
SA,NT,T,V,Q;
r, g, b



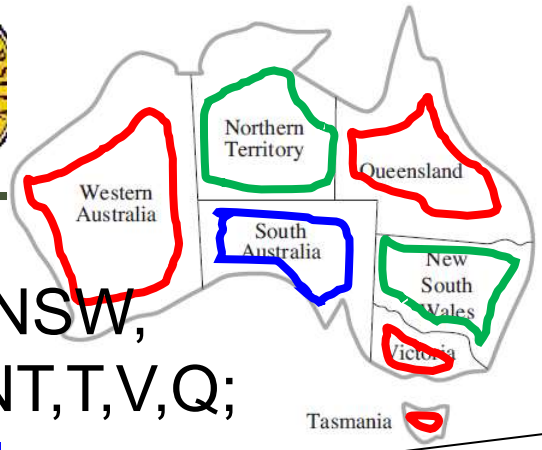
回溯搜索



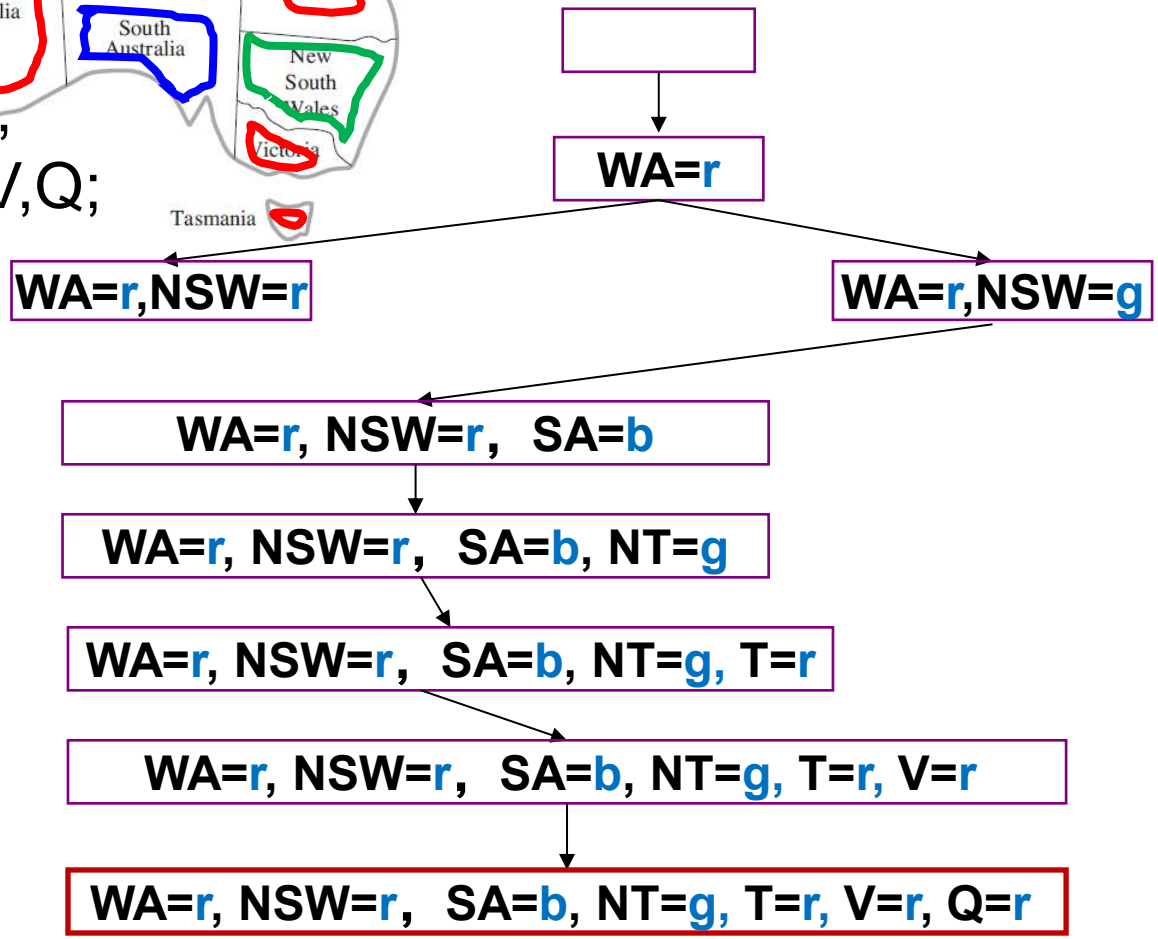
WA,NSW,
SA,NT,T,V,Q;
r, g, b



回溯搜索



WA,NSW,
SA,NT,T,V,Q;
r, g, b



计算复杂度：$\mathcal{O}(d^n)$

- 最差情况

提高计算效率

- 通用方法可以极大地提高计算效率
  - 预处理：预先检查相容性，缩小值域
    - 节点相容
    - 弧相容
  - 排序：
    - 变量被赋值的先后顺序
    - 值域中多个值的先后顺序
  - 推理：能否早点检测出必然出错的问题？
  - 结构：可以利用问题的结构吗？

提高计算效率

- 通用方法可以极大地提高计算效率
  - 预处理：预先检查相容性，缩小值域
    - 节点相容
    - 弧相容
  - 排序：
    - 变量被赋值的先后顺序
    - 值域中多个值的先后顺序
  - 推理：能否早点检测出必然出错的问题？
  - 结构：可以利用问题的结构吗？

节点相容

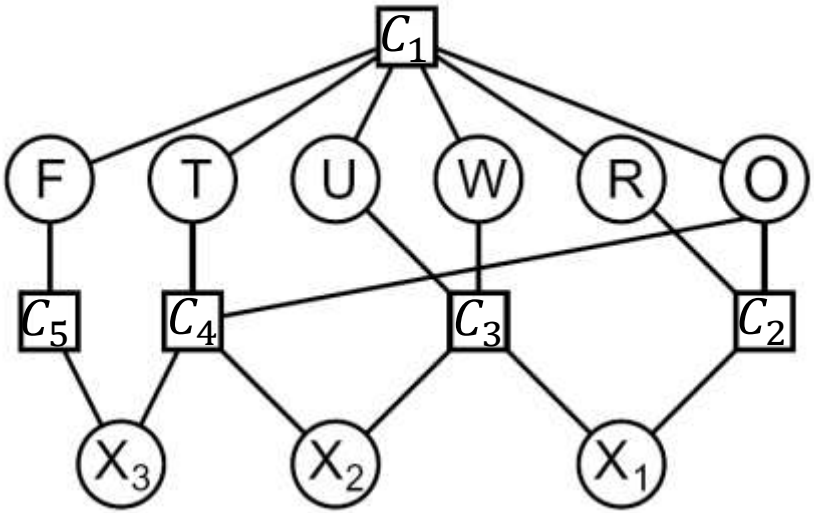
- 节点相容：单个变量(对应一个节点)值域中所有取值满足它的一元约束，即为节点相容

$$F \neq 0, X_3 \neq 0$$

算术密码

$$\begin{array}{r} T\ W\ O \\ +\ T\ W\ O \\ \hline F\ O\ U\ R \end{array}$$

C_1: alldiff(F, T, U, W, R, O)
C_2: $O + O = R + 10X_1$
C_3: $X_1 + W + W = U + 10X_2$
C_4: $X_2 + T + T = O + 10X_3$
C_5: $F = X_3 \neq 0$



弧相容

- 有向弧相容：X（弧尾）→Y（弧头）是相容的，当且仅当对变量X可取的任意值x都存在相容(满足约束)的赋值y
  - 约束：Y=2X, X和Y是0到10之间的整数
    - $D_x = D_y$ ={0, 1, 2, ..., 10}
    - $D_x = \{0,1,2,3,4,5\}$, D_y = {0, 1, 2, ..., 10}
    - D_x = {0, 1, 2, ..., 10} , $D_y = \{0,2,4,6,8,10\}$,
- 无向弧相容： X（弧尾）→Y（弧头）是相容的而且Y（弧尾）→X（弧头）也是相容的

$$D_x = \{0,1,2,3,4,5\},\ D_y = \{0,2,4,6,8,10\}$$

- CSP的弧相容(Arc Consistency)：所有变量对都是无向弧相容的
  - 弧相容检查：删除不相容的变量取值

AC-3算法

function AC-3(*csp*) **returns** the CSP, possibly with reduced domains
  inputs: *csp*, a binary CSP with variables $\{X_1, X_2, \ldots, X_n\}$
  local variables: *queue*, a queue of arcs, initially all the arcs in *csp*

  while *queue* is not empty **do**
    $(X_i, X_j) \leftarrow$ REMOVE-FIRST(*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*

- 约束传播：反复应用直到不再有矛盾
- 计算复杂度：$O(n^2d^3)$，可降低至$O(n^2d^2)$

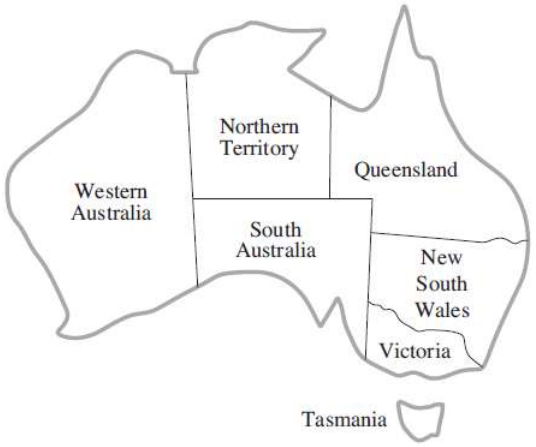
关于弧相容

- 经过弧相容判断后
  - 可能直接找到一个解
  - 可能简化问题

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7					4			8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

	1	2	3	4	5	6	7	8	9
A	4	8	3	9	2	1	6	5	7
B	9	6	7	3	4	5	8	2	1
C	2	5	1	8	7	6	4	9	3
D	5	4	8	1	3	2	9	7	6
E	7	2	9	5	6	4	1	3	8
F	1	3	6	7	9	8	2	4	5
G	3	7	2	6	8	9	5	1	4
H	8	1	4	2	5	3	7	6	9
I	6	9	5	4	1	7	3	8	2

  - 也可能没有任何作用



K-相容(K-Consistency)

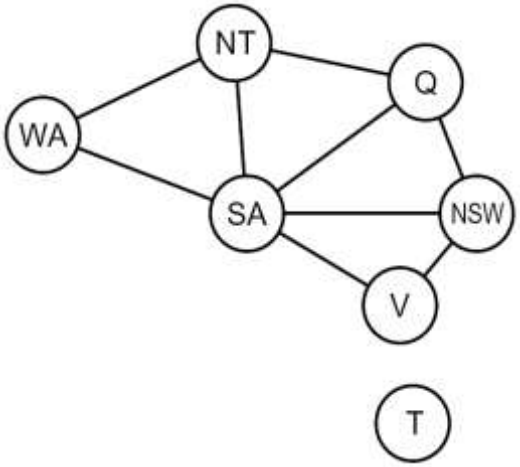
- K=1: 节点相容
- K=2: 弧相容
- K=3: 路径相容（Path Consistency）
  - $\{X, Y\} \rightarrow Z$
- K-相容:
  - $\{X_{i_1}, \cdots X_{i_{k-1}}\} \rightarrow X_{i_k}$
- 强K相容: 也即K, K-1，K-2，…，1相容
- K值越大，效果越明显，但是计算代价越大
- 只要求掌握2-相容的情况，即弧相容

提高计算效率

- 通用方法可以极大地提高计算效率
  - 预处理：预先检查相容性，缩小值域
    - 节点相容
    - 弧相容
  - 排序：
    - 变量被赋值的先后顺序
    - 值域中多个值的先后顺序
  - 推理：能否早点检测出必然出错的问题？
  - 结构：可以利用问题的结构吗？

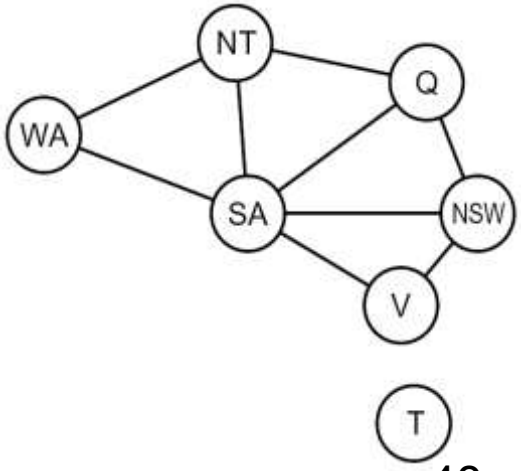
变量排序

- 最小剩余值(Minimum Remaining Values, MRV)
  - 合法取值最少的变量
  - 为何选择最小剩余值？(而不是最大剩余值)
  - 也被称为“最受约束变量”
  - “失败优先”启发式
- 在MRV无法抉择时启动度启发式(Degree Heuristic, DH)
  - 选择涉及对其他未赋值变量约束最大的变量



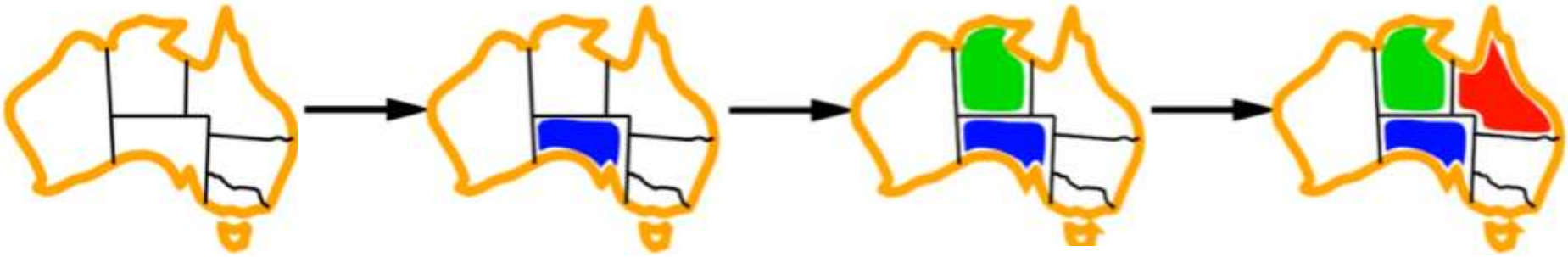
变量排序

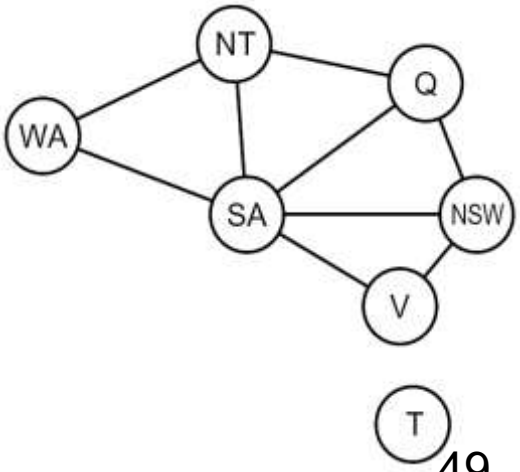
- 最小剩余值(Minimum Remaining Values, MRV)
  - 合法取值最少的变量
  - 为何选择最小剩余值？(而不是最大剩余值)
  - 也被称为“最受约束变量”
  - “失败优先”启发式



变量排序

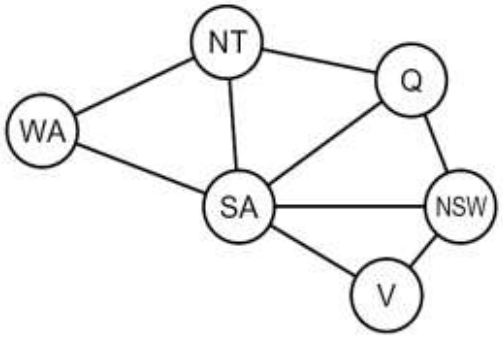
- 在MRV无法抉择时启动度启发式(Degree Heuristic, DH)
  - 选择涉及对其他未赋值变量约束最大的变量

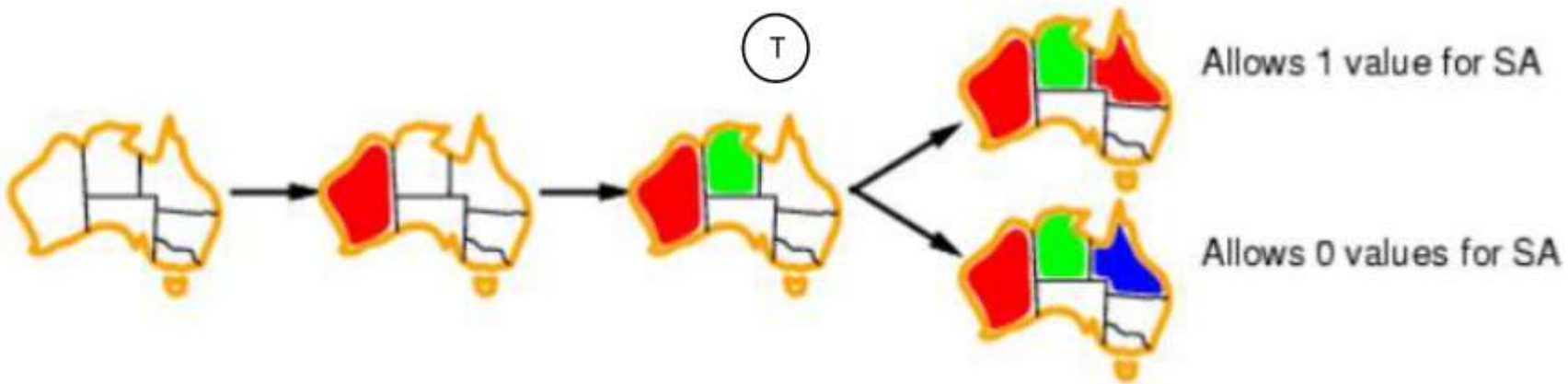




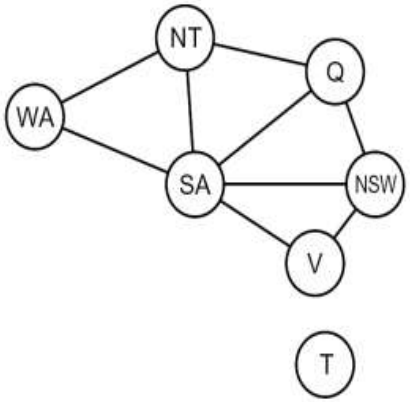
值的排序

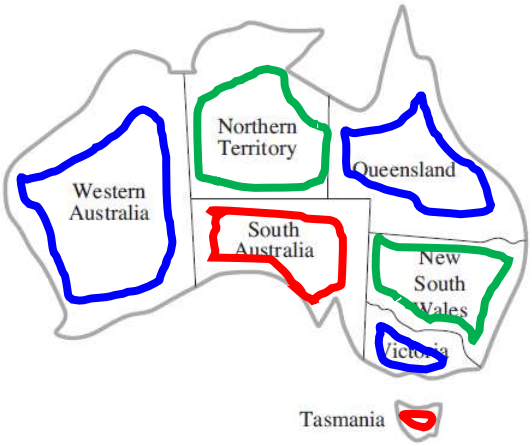
- 最少约束值（Least Constraining Value, LCV）
  - 优先选择使得邻居变量的可选值最多的值，即对邻居的约束最少
  - 为什么是最少而不是最多?



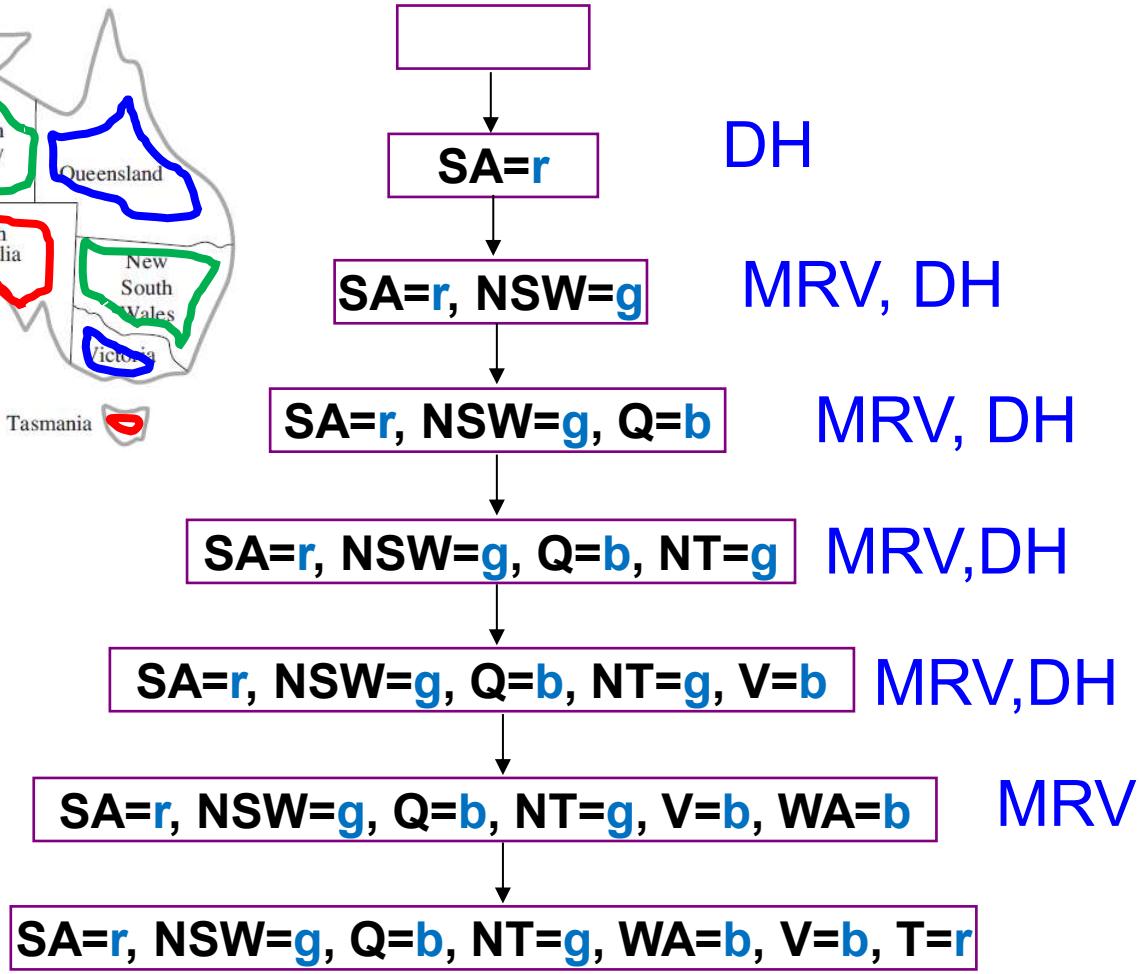


排序的效果





- 变量排序
  - MRV
  - DH
  - 首字母A-Z
- 值排序
  - LCV
  - r, g, b



一般情况下，计算复杂度仍为$\mathcal{O}(d^n)$

提高计算效率

- 通用方法可以极大地提高计算效率
  - 预处理：预先检查相容性，缩小值域
    - 节点相容
    - 弧相容
  - 排序：
    - 变量被赋值的先后顺序
    - 值域中多个值的先后顺序
  - 推理：能否早点检测出必然出错的问题？
  - 结构：可以利用问题的结构吗？

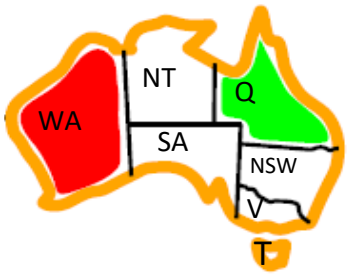
前向检验

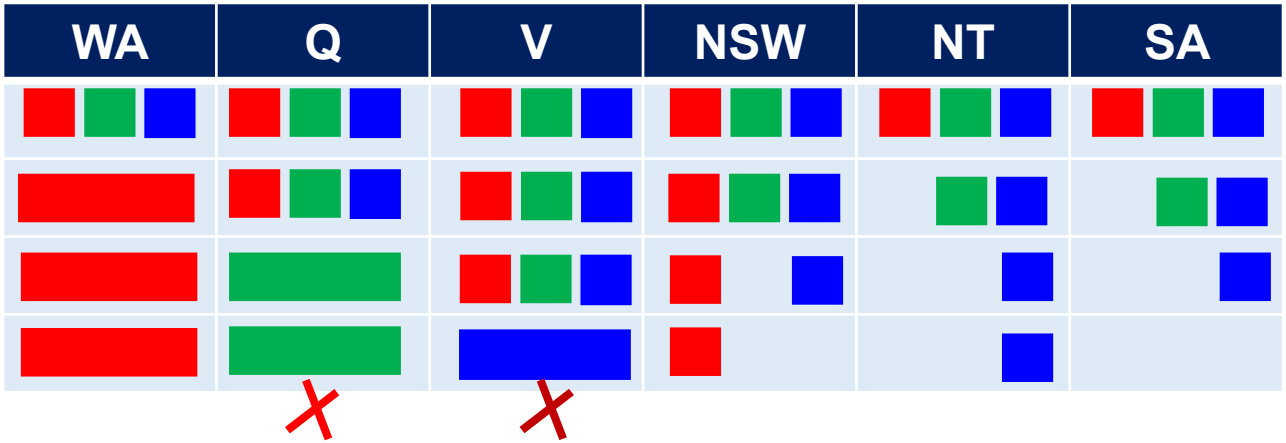
- 记录未赋值的变量的剩余合法值，并删除错误选项，当任一变量没有合法值时返回上一层（回溯）
  - 对每一个变量赋值时，更新剩余变量的值域
  - 搜索过程中动态检查弧相容性

WA	Q	V	NSW	NT	SA

弧相容保持

- 前向检验无法提前检测出所有的矛盾情况
  - 如：没有注意到NT和SA不可能同时为蓝色！

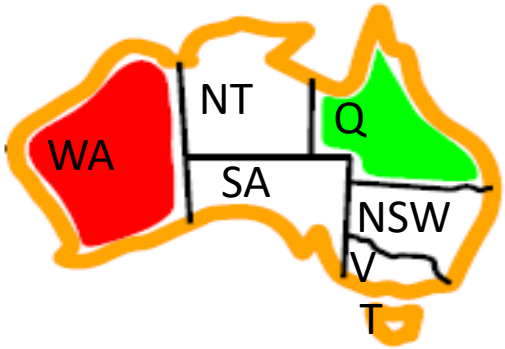




  - 原因：只检查其它未赋值变量与当前赋值变量的弧相容性，没有考虑其它变量之间的弧相容性
- 弧相容保持（Maintaining Arc Consistency, MAC）
  - 当X_i被赋值时，调用AC-3算法检查

    $\{(X_j, X_i)$: X_j是X_i的未赋值相邻变量$\}$的弧相容性
  - 约束传播

弧相容保持



WA	Q	V	NSW	NT	SA

前向检验

弧相容保持

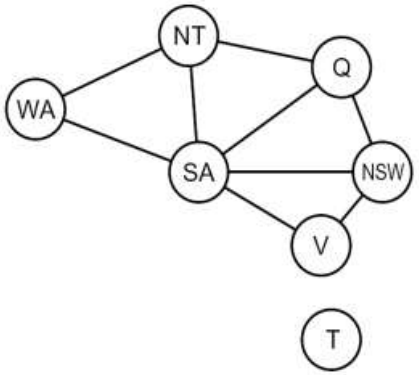
NT → SA, SA → NT

提高计算效率

- 通用方法可以极大地提高计算效率
  - 预处理：预先检查相容性，缩小值域
    - 节点相容
    - 弧相容
  - 排序：
    - 变量被赋值的先后顺序
    - 值域中多个值的先后顺序
  - 推理：能否早点检测出必然出错的问题？
  - 结构：可以利用问题的结构吗？

问题结构

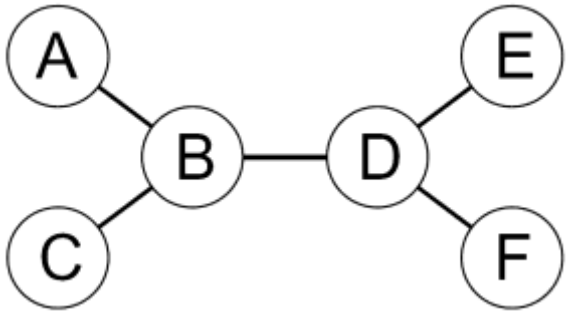
- 独立子问题：约束图的连通分量
  - e.g.: T岛与大陆不连通
  - $CSP = \cup_{i=1}^{m} CSP_i$
  - S_i是CSP_i的解
  - $\cup_{i=1}^{m} S_i$是CSP的解



- 假设有n个变量的约束图可以被拆分为n/c个只有c个变量的子问题
  - 最坏情况下的代价为$O((n/c)(d^c))$
  - e.g.: n=80, d=2, c=20
  - 2^{80}=40亿年，以1000万浮点/秒的速度计算
  - (4)(2^{20}) = 0.4秒，以1000万浮点/秒的速度计算

树状结构的CSP

- 树结构：任意两个节点只有一条（无重复）路径相连



- 定理：树结构的CSP可以在$O(nd^2)$时间内被求解
  - 一般CSP最坏情况下的时间为$O(d^n)$
- 算法：拓扑排序
  - 选择任意变量为根节点，即图变为树
  - 父节点到子节点为有向弧

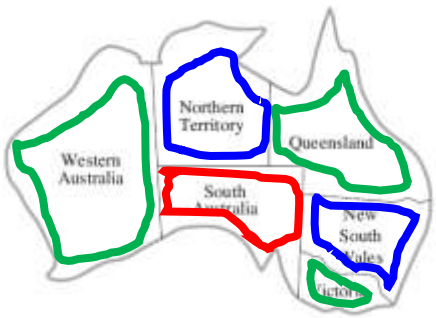
  - 对树做广度优先遍历，可得变量排序顺序

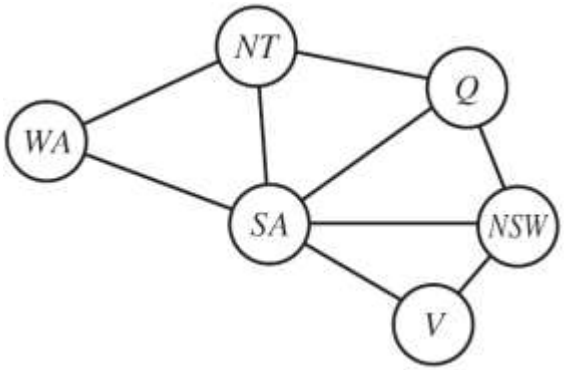
树状结构CSP的求解算法

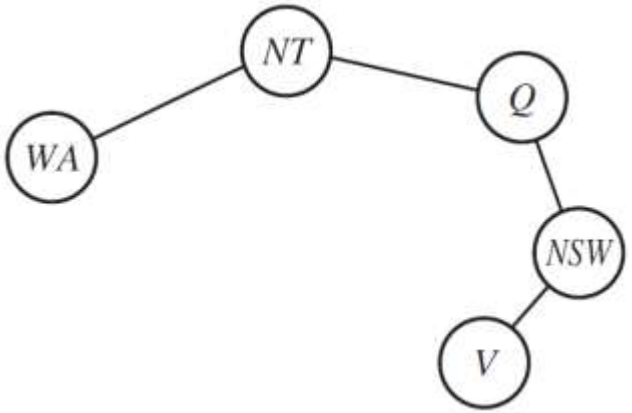
function TREE-CSP-SOLVER(csp) **returns** a solution, or failure
  inputs: csp, a CSP with components X, D, C

  $n \leftarrow$ number of variables in X
  $assignment \leftarrow$ an empty assignment
  $root \leftarrow$ any variable in X
  $X \leftarrow$ TOPOLOGICALSORT($X, root$)
  for $j = n$ **down to** 2 **do**
    MAKE-ARC-CONSISTENT(PARENT(X_j), X_j)
    if it cannot be made consistent **then return** *failure*
  for $i = 1$ **to** n **do**
    $assignment[X_i] \leftarrow$ any consistent value from D_i
    if there is no consistent value **then return** *failure*
  return *assignment*

一般CSP中的树结构





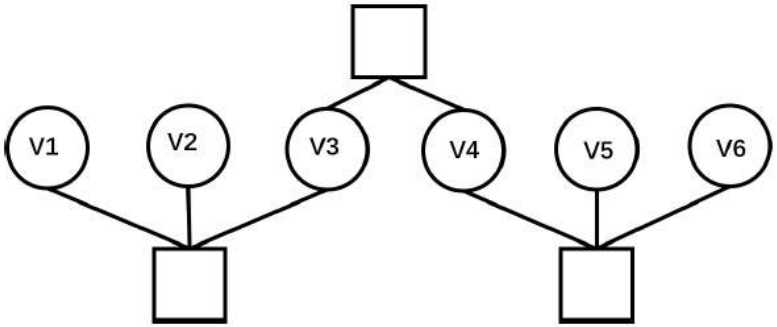


前向检验
有向弧相容
依次赋值

SA	WA	NT	Q	NSW	V

练习

2. 考虑下图所示的约束图，V1-V6 的取值范围为 1-9，约束关系已在图上标明，约束条件为取值不同。请问（A）、（B）、（C）是否弧相容？请在是或否上打 √（本小题共 6 分）



V1={2,4,5}	V2={1,2,4}	V3={2,3,7}	V4={1,4,5}	V5={2,6,9}	V6={3,6,8}

(A)

V1={2}	V2={2,4}	V3={2}	V4={4}	V5={6,9}	V6={6,9}

(B)

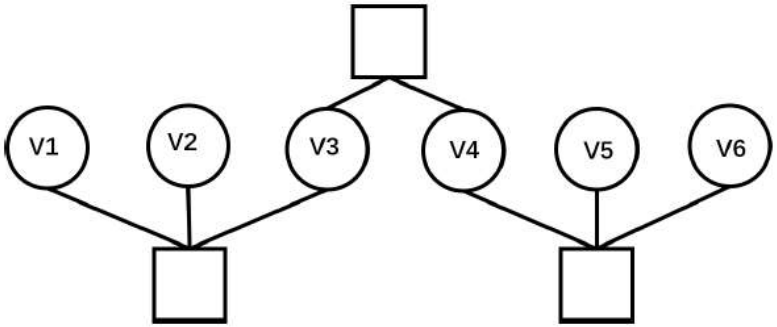
V1={2,4,5}	V2={1,2,4}	V3={2,3,7}	V4={1,4,5}	V5={2,6,9}	V6={9}

(C)

(A) 是 或 否 弧相容；
(B) 是 或 否 弧相容；
(C) 是 或 否 弧相容；

练习：答案

2. 考虑下图所示的约束图，V1-V6 的取值范围为 1-9，约束关系已在图上标明，约束条件为取值不同。请问（A）、（B）、（C）是否弧相容？请在是或否上打 √（本小题共 6 分）



V1={2,4,5}	V2={1,2,4}	V3={2,3,7}	V4={1,4,5}	V5={2,6,9}	V6={3,6,8}

(A)

V1={2}	V2={2,4}	V3={2}	V4={4}	V5={6,9}	V6={6,9}

(B)

V1={2,4,5}	V2={1,2,4}	V3={2,3,7}	V4={1,4,5}	V5={2,6,9}	V6={9}

(C)

(A) ~~是~~ 或 否 弧相容；
(B) 是 或 ~~否~~ 弧相容；
(C) 是 或 ~~否~~ 弧相容；