约束满足问题
CONSTRAINT SATISFACTION PROBLEM (CSP)

张玲玲
计算机学院
zhanglling@xjtu.edu.cn

主要内容

1. 约束满足问题（CSP）
2. CSP的回溯搜索
3. 前向检验
4. CSP局部搜索

约束满足问题（CSP）

- 标准搜索问题：
  - 状态是一个“黑盒子”——支持后继、启发式和目标测试的任何数据结构。
- 约束满足问题：
  - 状态由变量$X_i \in X$定义，其值来自值域$D_i \in D$。
    - X是变量集合$\{X_1, \cdots, X_n\}$。
    - D是值域集合$\{D_1, \cdots, D_n\}$，每个变量X_i有自己的值域D_i。
    - 值域D_i是变量X_i的可能取值$\{v_1, \cdots, v_k\}$组成的集合。
  - 目标测试是一组约束，用于指定变量子集的值的允许组合。
    - C是描述变量取值的约束集合。
    - 每个约束C_i是有序对$\langle scope, rel \rangle$，其中$scope$是约束中的变量组，$rel$定义了这些变量取值应满足的关系。

约束满足问题（CSP）

- 每个约束C_i是有序对$\langle scope, rel\rangle$，其中$scope$是约束中的变量组，$rel$定义了这些变量取值应满足的关系。
- 关系可以显式地列出所有关系元组，也可以是支持以下两个操作的抽象关系：
  ① 测试一个元组是否为一个关系的成员
  ② 枚举所有关系成员

- 例如：如果变量X_1和X_2的值域均为$\{A, B\}$，约束是二者不能取相同值，关系可以描述为：
  ① $\langle (X_1, X_2), [(A, B), (B, A)]\rangle$
  ② $\langle (X_1, X_2), X_1 \neq X_2\rangle$

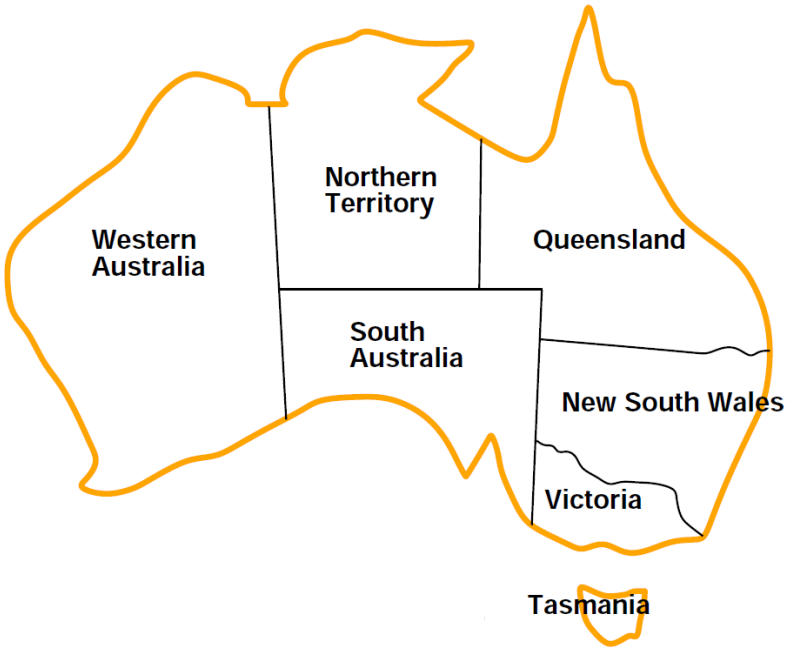
约束满足问题（CSP）

- 求解约束满足问题CSP，需要定义状态空间和解的概念。

- 问题的状态：对部分或全部变量的一个赋值，如 $\{X_i = v_i, X_j = v_j, \cdots\}$。

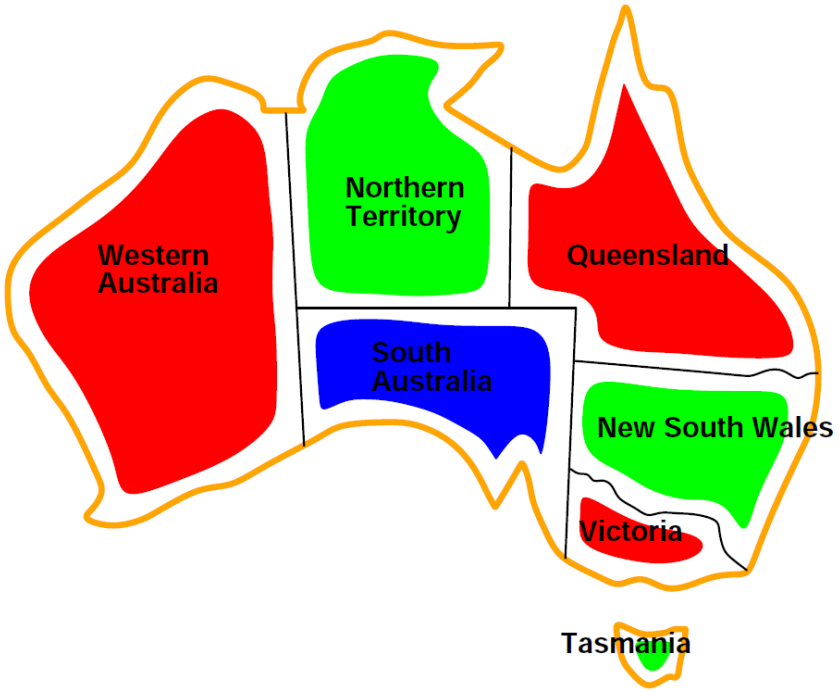
- 相容的（合法的）赋值：一个不违反任何约束条件的赋值。
- 完整赋值：每个变量都已赋值。
- CSP的解：相容的、完整的赋值。
- 部分赋值：只有部分变量赋值。

实例：地图着色问题



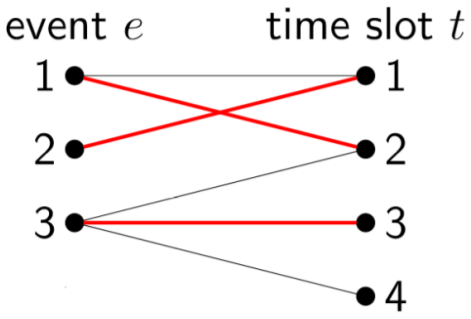
- 变量：$X = \{WA, NT, Q, NSW, V, SA, T\}$。
- 值域：$D_i = \{red, green, blue\}$
- 约束：相邻的区域要染成不同的颜色。
  - 例如：$WA \neq NT$，或者$(WA, NT) \in \{(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)\}$

实例：地图着色问题



- 解是完整且相容的赋值，例如，$\{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green\}$

真实世界的CSP



- 有E个事件和T个时隙。
- 每个事件e必须恰好放在一个时隙中。
- 每个时隙t最多可以有一个事件。

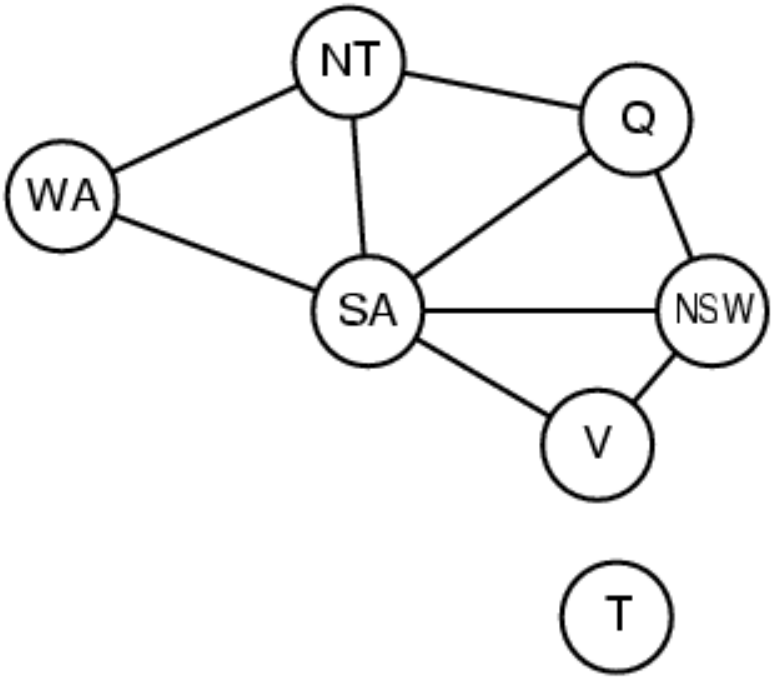
- 分配问题
  - 例如，谁教什么课
- 时间表问题
  - 例如，何时何地提供哪种课程？
- 运输调度
- 工厂调度

CSP的形式化

- 离散变量
  - 有限值域：
    - n个变量，值域大小$d \rightarrow O(d^n)$完整赋值
  - 无限值域：
    - 整数集合、字符串集合等
    - 例如，作业调度，变量是每个任务的开始/结束时间
    - 无法枚举所有可能取值的组合，需要约束语言，例如$StartJob_1 + 5 \leq StartJob_3$
- 连续变量
  - 例如，哈勃太空望远镜观测的开始/结束时间
  - 通过线性规划可在多项式时间内求解线性约束

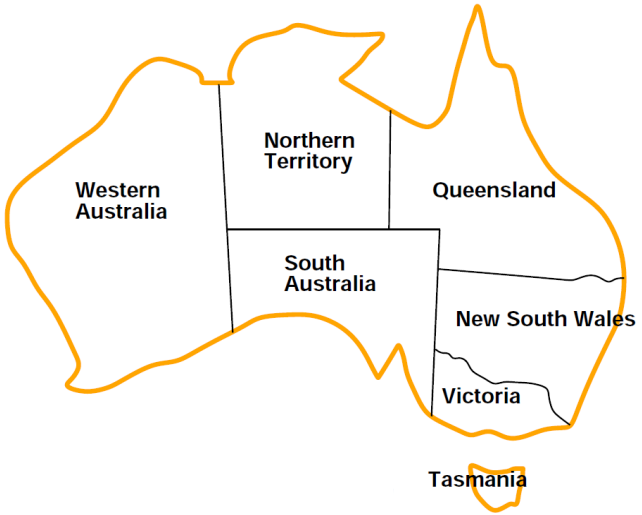
约束图

- 二元CSP：每个约束与两个变量相关
- 约束图：结点是变量，弧是约束



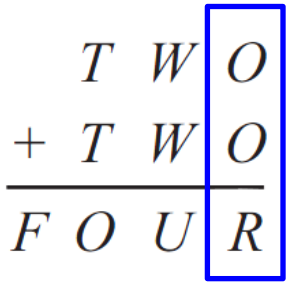
CSP的形式化

- 一元约束与单个变量相关
  - 例如，$SA \neq green$
- 二元约束与两个变量相关
  - 例如，$SA \neq WA$
- 高阶约束与三个或多个变量相关
  - 例如，密码算术谜题
  - 例如，八皇后问题
- 全局约束与任意个数的变量相关



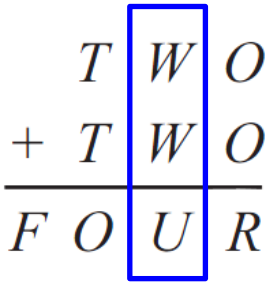
- 偏好约束（软约束），例如红色胜于绿色。
- 通常被处理成个体变量赋值的开销 $\Longrightarrow$ 约束优化问题。

举例：密码算术谜题



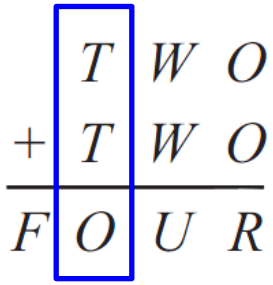
- 变量：$F\ T\ U\ W\ R\ O\ C_1\ C_2\ C_3$
- 值域：$\{0,1,2,3,4,5,6,7,8,9\}$
- 约束：$Alldiff(F,T,U,W,R,O)$
  - $O+O=R+10\cdot C_1$

举例：密码算术谜题



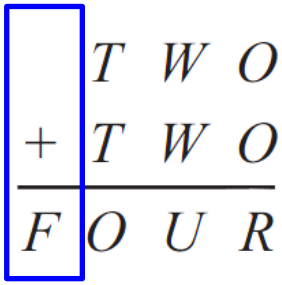
- 变量：$F\ T\ U\ W\ R\ O\ C_1\ C_2\ C_3$
- 值域：$\{0,1,2,3,4,5,6,7,8,9\}$
- 约束：$Alldiff(F,T,U,W,R,O)$
  - $O+O=R+10\cdot C_1$
  - $C_1+W+W=U+10\cdot C_2$

举例：密码算术谜题



- 变量：$F\ T\ U\ W\ R\ O\ C_1\ C_2\ C_3$
- 值域：$\{0,1,2,3,4,5,6,7,8,9\}$
- 约束：$Alldiff(F,T,U,W,R,O)$
  - $O+O=R+10\cdot C_1$
  - $C_1+W+W=U+10\cdot C_2$
  - $C_2+T+T=O+10\cdot C_3$

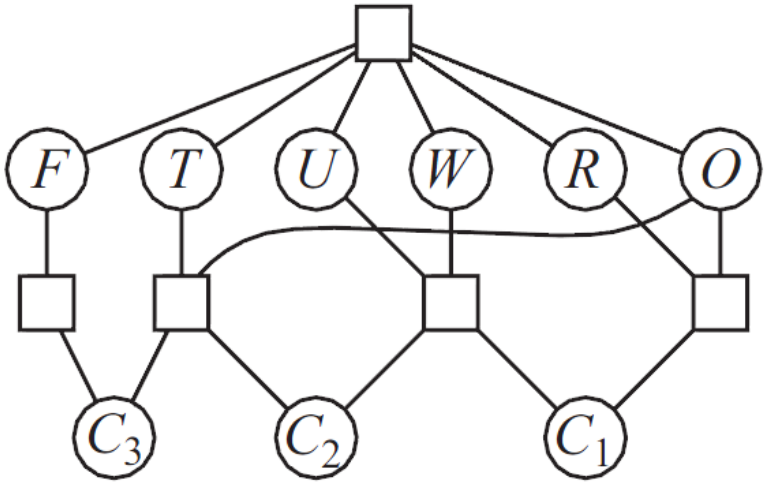
举例：密码算术谜题



- 变量：$F\ T\ U\ W\ R\ O\ C_1\ C_2\ C_3$
- 值域：$\{0,1,2,3,4,5,6,7,8,9\}$
- 约束：$Alldiff(F,T,U,W,R,O)$
  - $O+O=R+10\cdot C_1$
  - $C_1+W+W=U+10\cdot C_2$
  - $C_2+T+T=O+10\cdot C_3$
  - $C_3=F,\ T\neq 0,\ F\neq 0$

举例：密码算术谜题

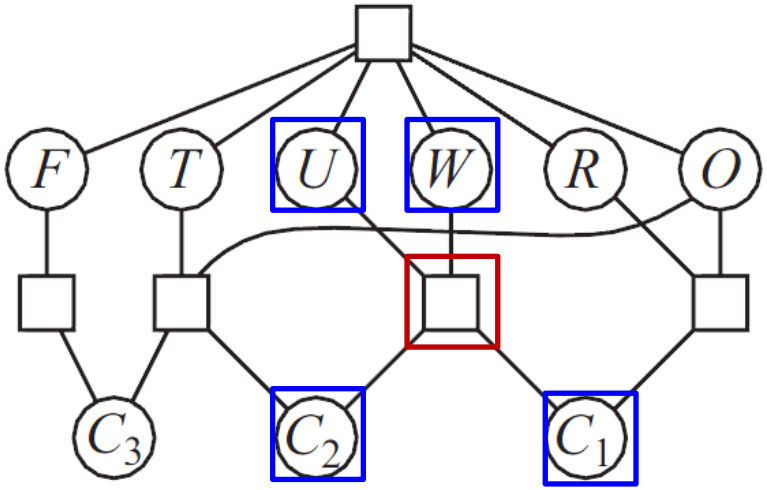
$$\begin{array}{r} T\ W\ O \\ +\ T\ W\ O \\ \hline F\ O\ U\ R \end{array}$$



- 变量：$F\ T\ U\ W\ R\ O\ C_1\ C_2\ C_3$
- 值域：$\{0,1,2,3,4,5,6,7,8,9\}$
- 约束：$Alldiff(F,T,U,W,R,O)$
  - $O+O=R+10\cdot C_1$
  - $C_1+W+W=U+10\cdot C_2$
  - $C_2+T+T=O+10\cdot C_3$
  - $C_3=F,\ T\neq 0,\ F\neq 0$

举例：密码算术谜题

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



- 变量：$F\ T\ U\ W\ R\ O\ C_1\ C_2\ C_3$
- 值域：$\{0,1,2,3,4,5,6,7,8,9\}$
- 约束：$Alldiff(F,T,U,W,R,O)$
  - $O+O=R+10\cdot C_1$
  - $C_1+W+W=U+10\cdot C_2$
  - $C_2+T+T=O+10\cdot C_3$
  - $C_3=F,\ T\neq 0,\ F\neq 0$

标准搜索形式化（递增）

- 状态：到目前为止的赋值。
- 初始状态：空赋值{}。
- 后继函数：赋值给一个与当前赋值不冲突的未赋值变量。
  → 如果没有合法赋值，将失败。（无法修复！）
- 目标测试：当前赋值是完整的。

① 对所有CSP都是一样的。

② 每个解出现在具有n个变量的深度n处 → 可以使用深度优先搜索。

③ 路径无关紧要，因此也可以使用完整状态形式化。

④ 在深度l，分支因子$b = (n - l)d$，d是值域大小，因此生成了一棵有$n!\, d^n$个叶子的搜索树！！！尽管可能的完整赋值只有d^n个！！！

回溯搜索

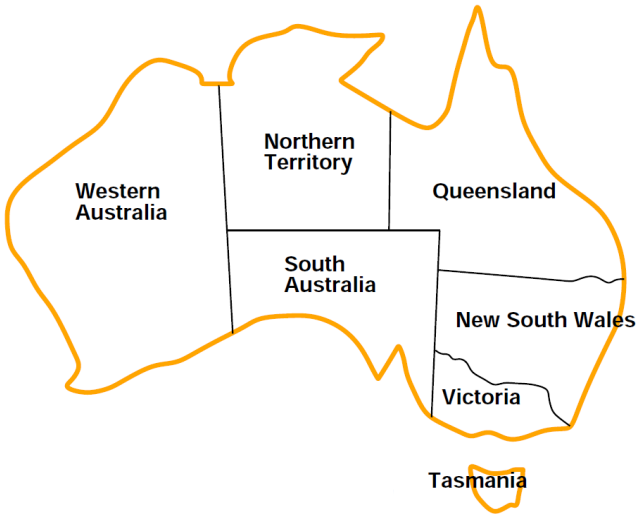
- CSP的变量赋值是可交换的，即$[WA = red \text{ then } NT = green]$与$[NT = green \text{ then } WA = red]$相同。
- 因此，给变量赋值的时候不需要考虑顺序，只需要考虑在搜索树的每个结点上赋值给单个变量。

  → 最大分支因子b = 值域大小d，并且有d^n个叶子。
- 具有单变量赋值的CSP的深度优先搜索称为回溯搜索。
  - 它每次为一个变量选择一个赋值，当没有合法的赋值时就回溯。
  - 它不断选择未赋值变量，轮流尝试变量值域中的每个值，试图找到一个解。
  - 一旦检测到不相容（不合法），回溯失败，返回上一次调用尝试另一个值。
- 回溯搜索是CSP的基本的无信息算法。
- 可以求解$n = 25$的n皇后问题。

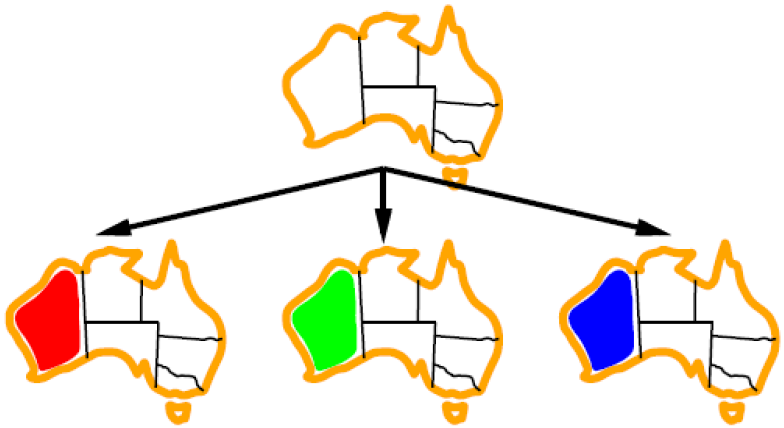
回溯搜索

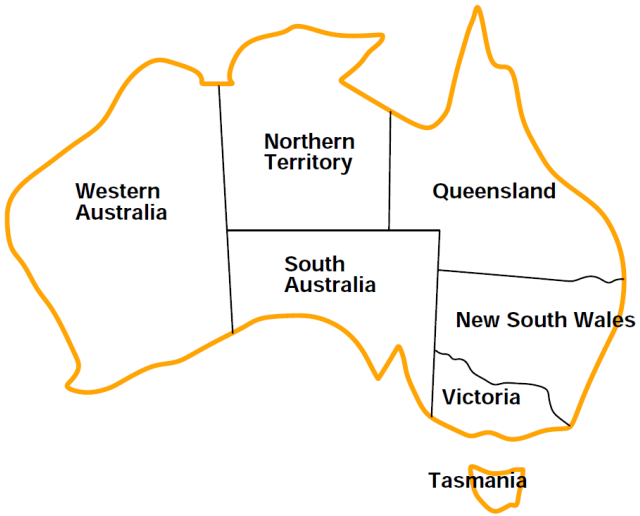
```
function BACKTRACKING-SEARCH(csp) returns solution/failure
    return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
    if assignment is complete then return assignment
    var ← 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 ← RECURSIVE-BACKTRACKING(assignment, csp)
            if result ≠ failure then return result
            remove {var = value} from assignment
    return failure
```

回溯搜索举例：地图着色问题

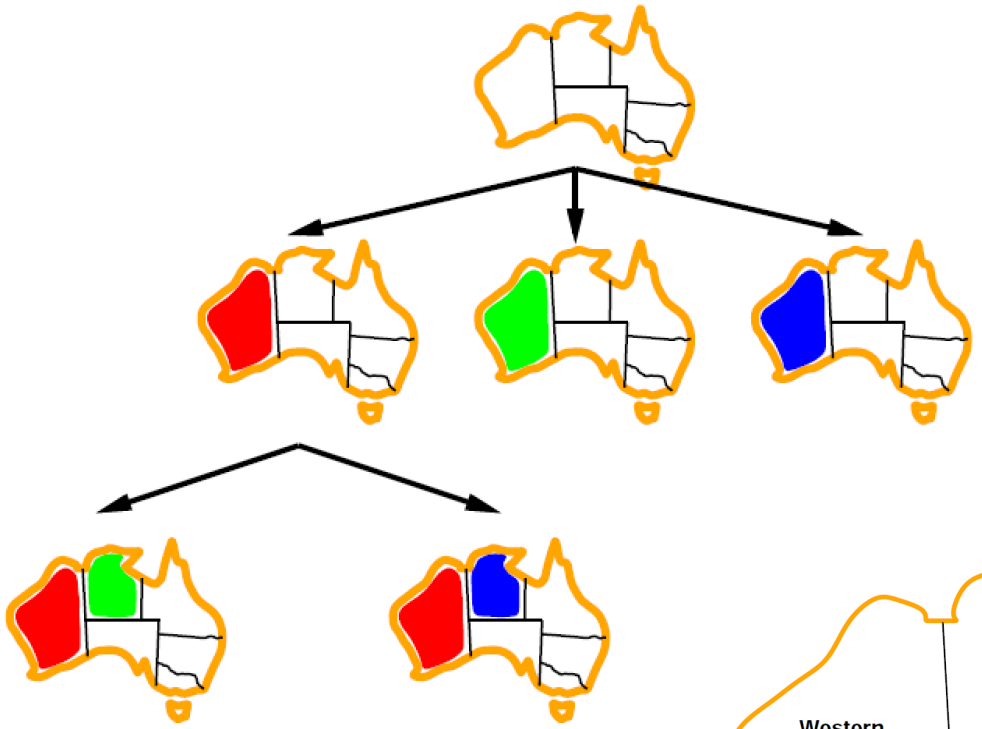


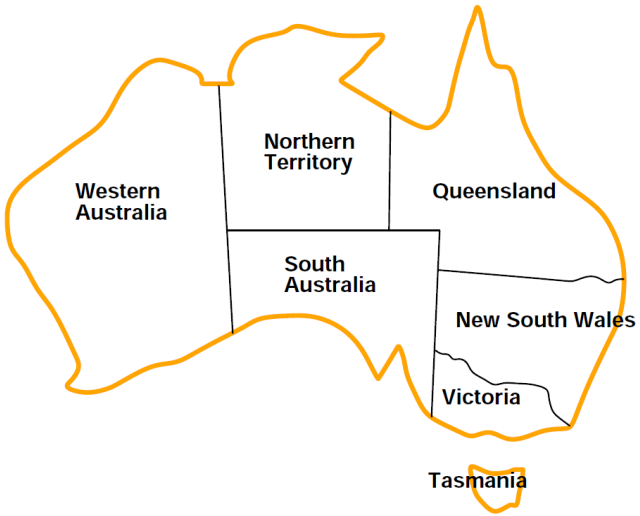
回溯搜索举例：地图着色问题



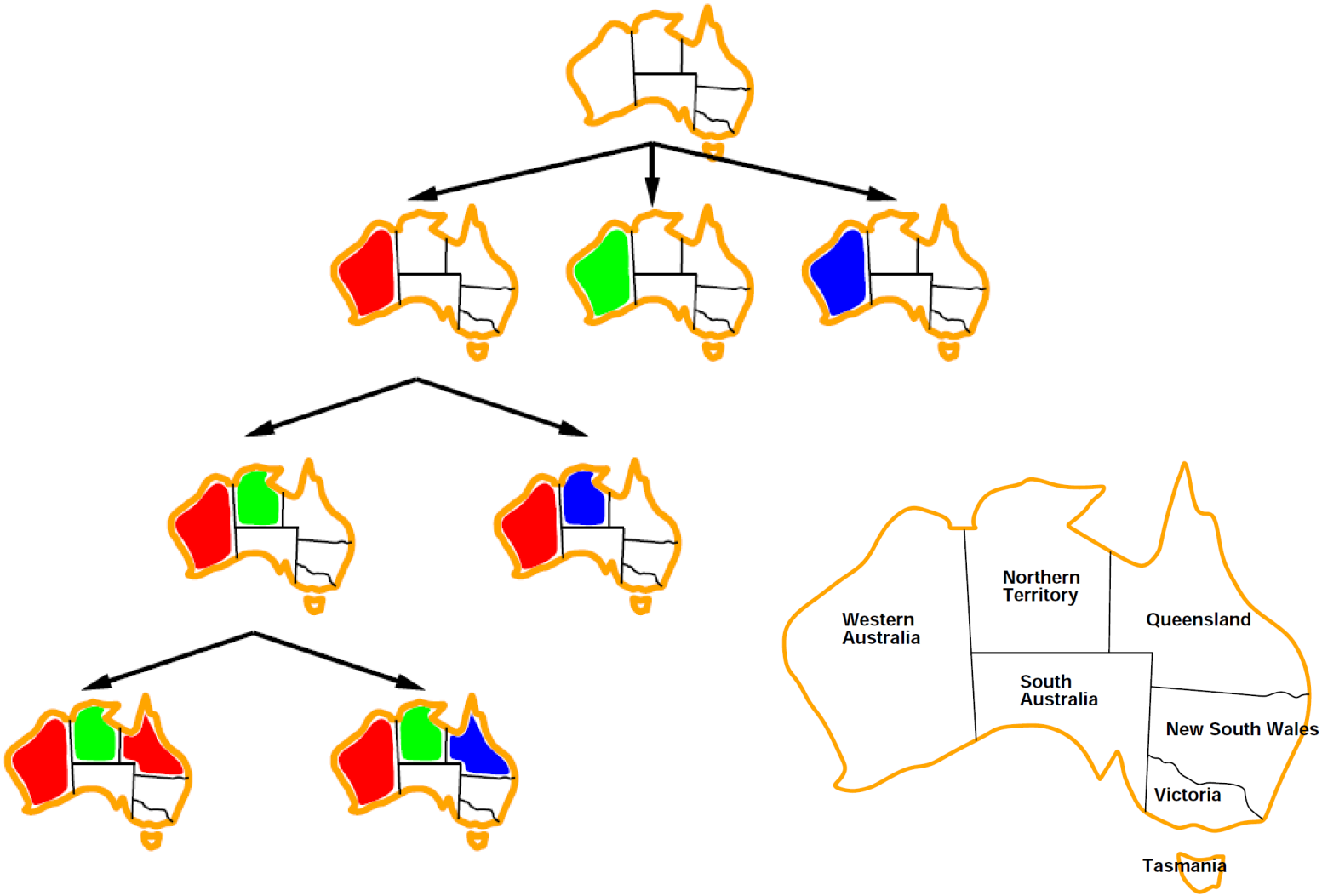


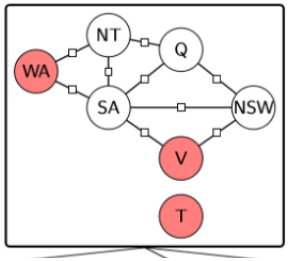
回溯搜索举例：地图着色问题

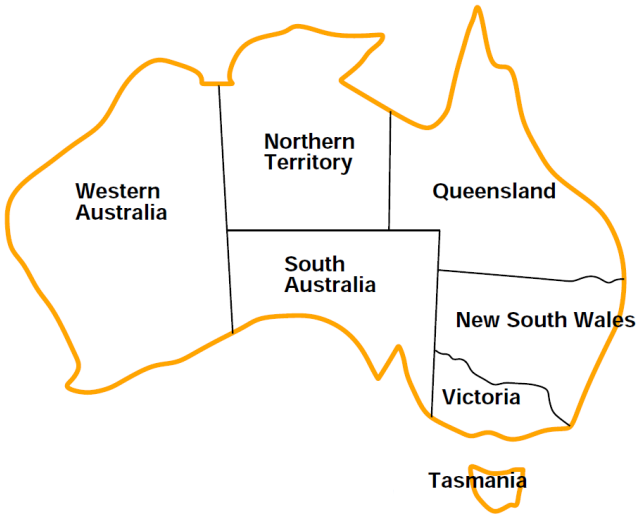


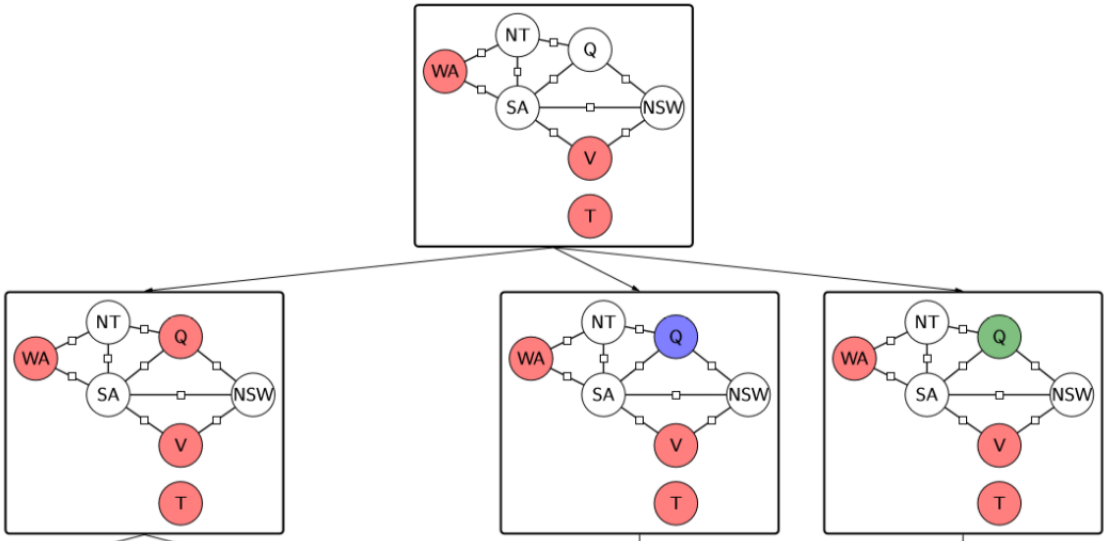


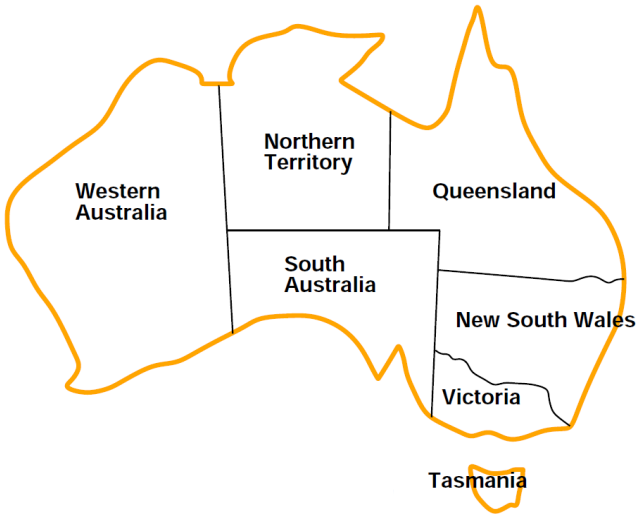
回溯搜索举例：地图着色问题

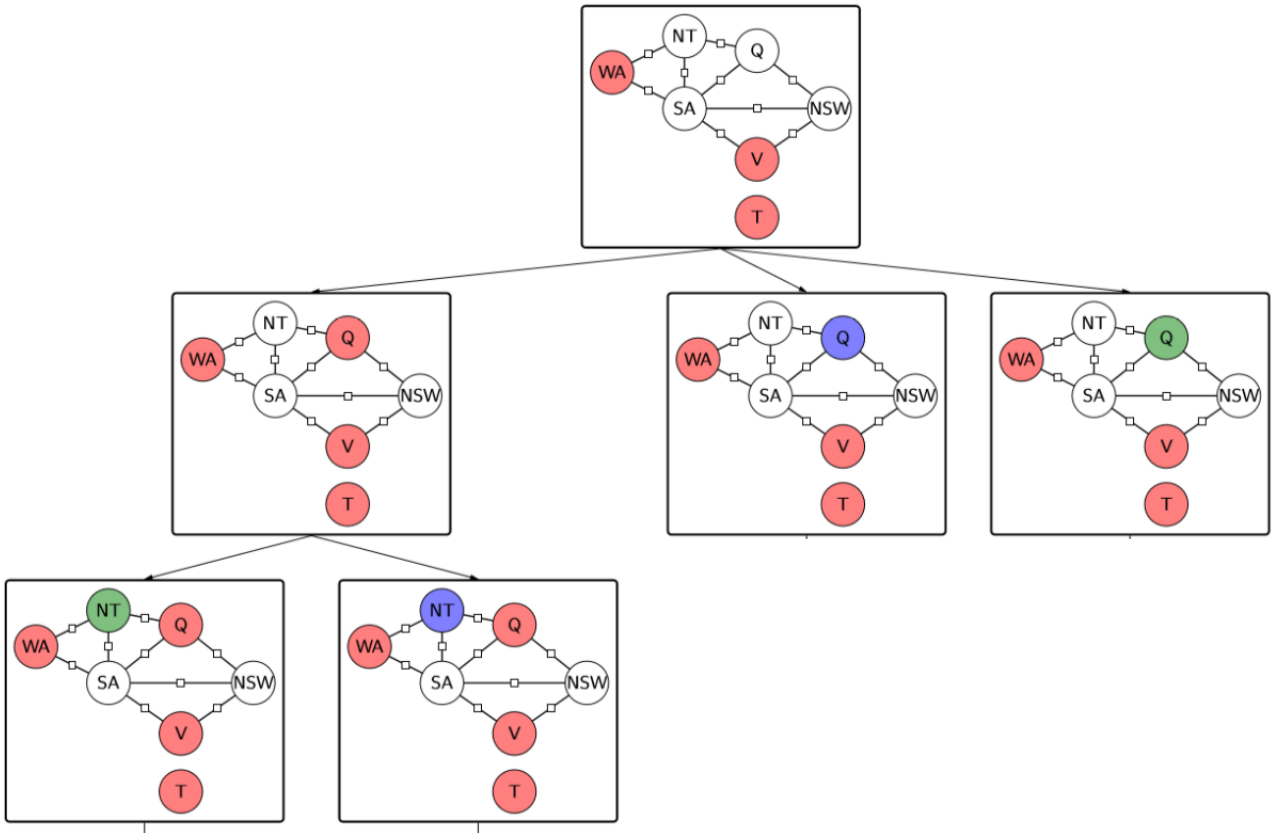


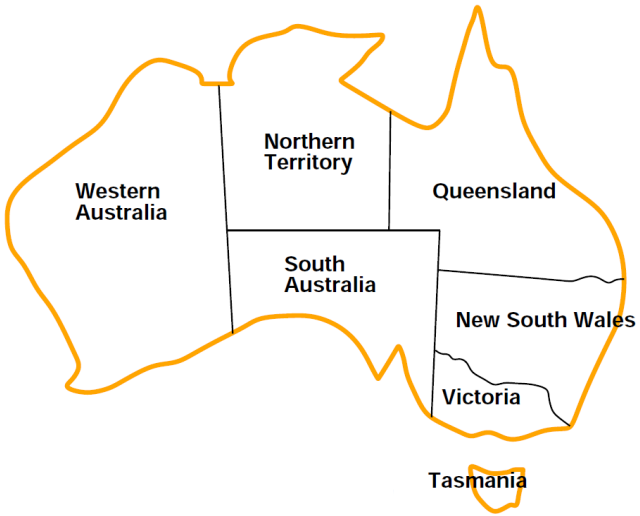

NT
Q
WA
SA
NSW
V
T

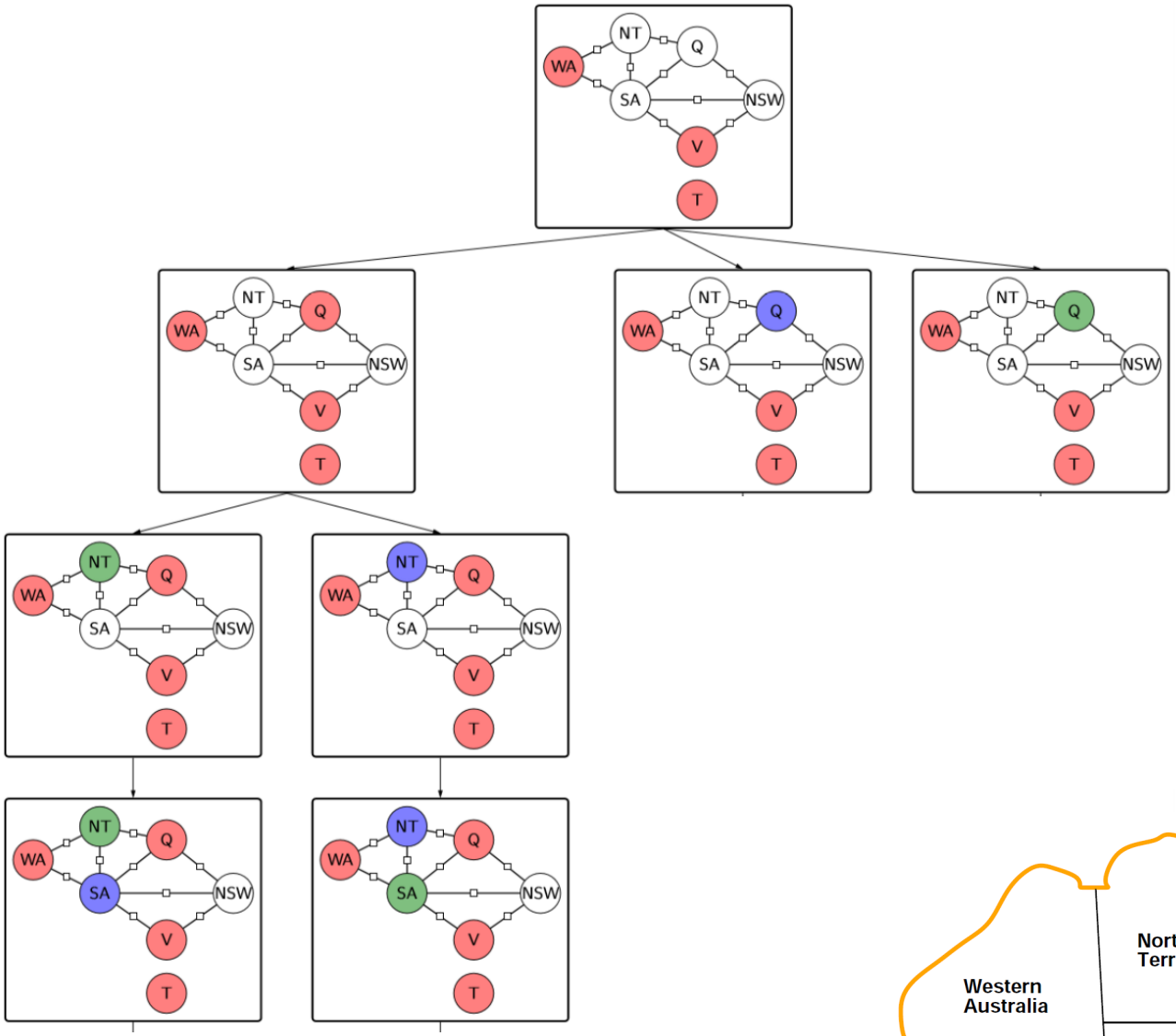

Northern Territory
Western Australia
Queensland
South Australia
New South Wales
Victoria
Tasmania

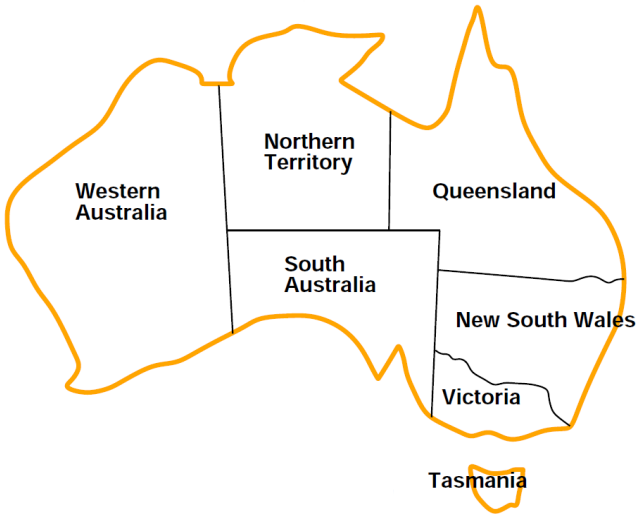

NT
Q
WA
SA
NSW
V
T

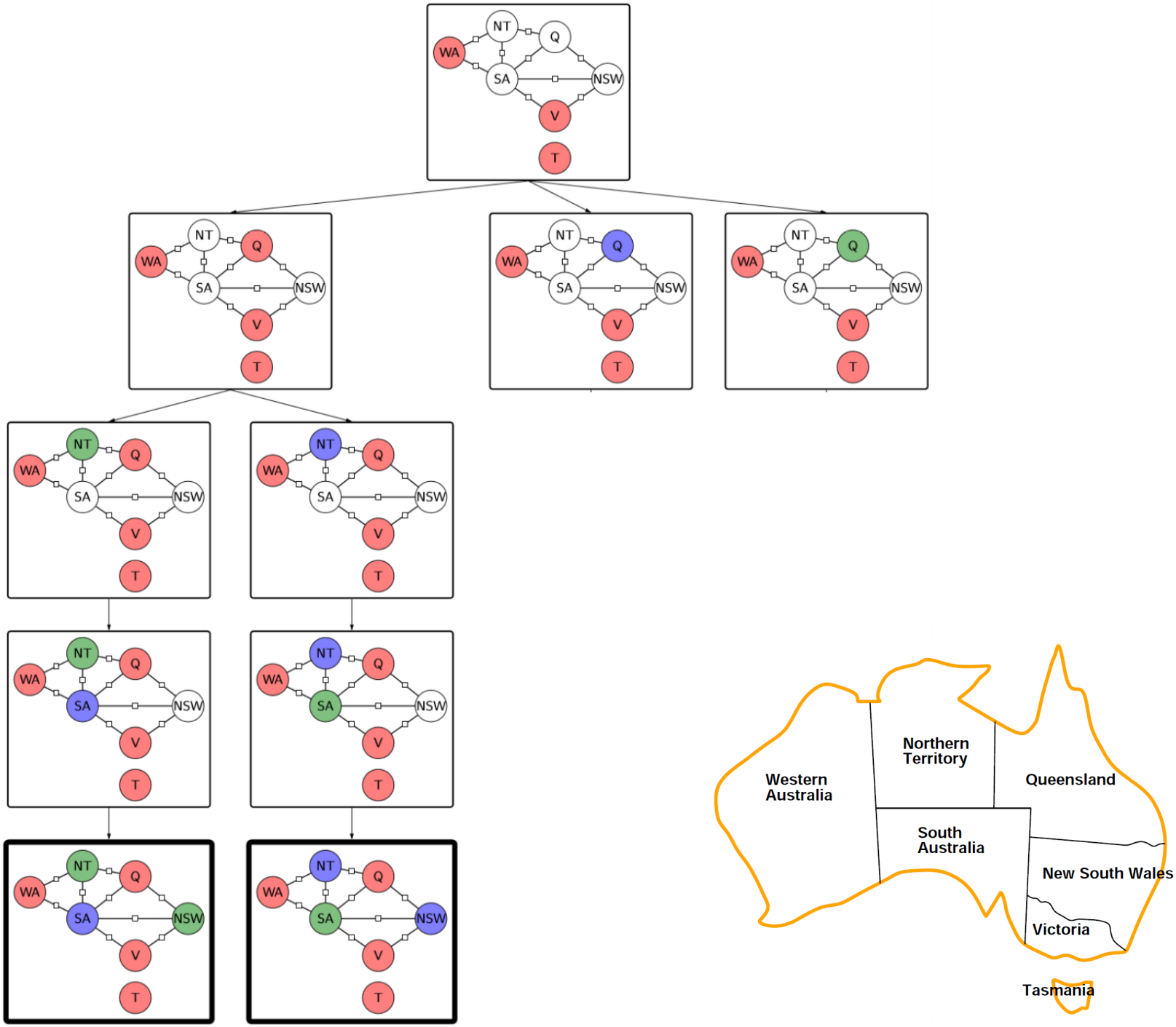

Northern Territory
Western Australia
Queensland
South Australia
New South Wales
Victoria
Tasmania


NT
Q
WA
SA
NSW
V
T

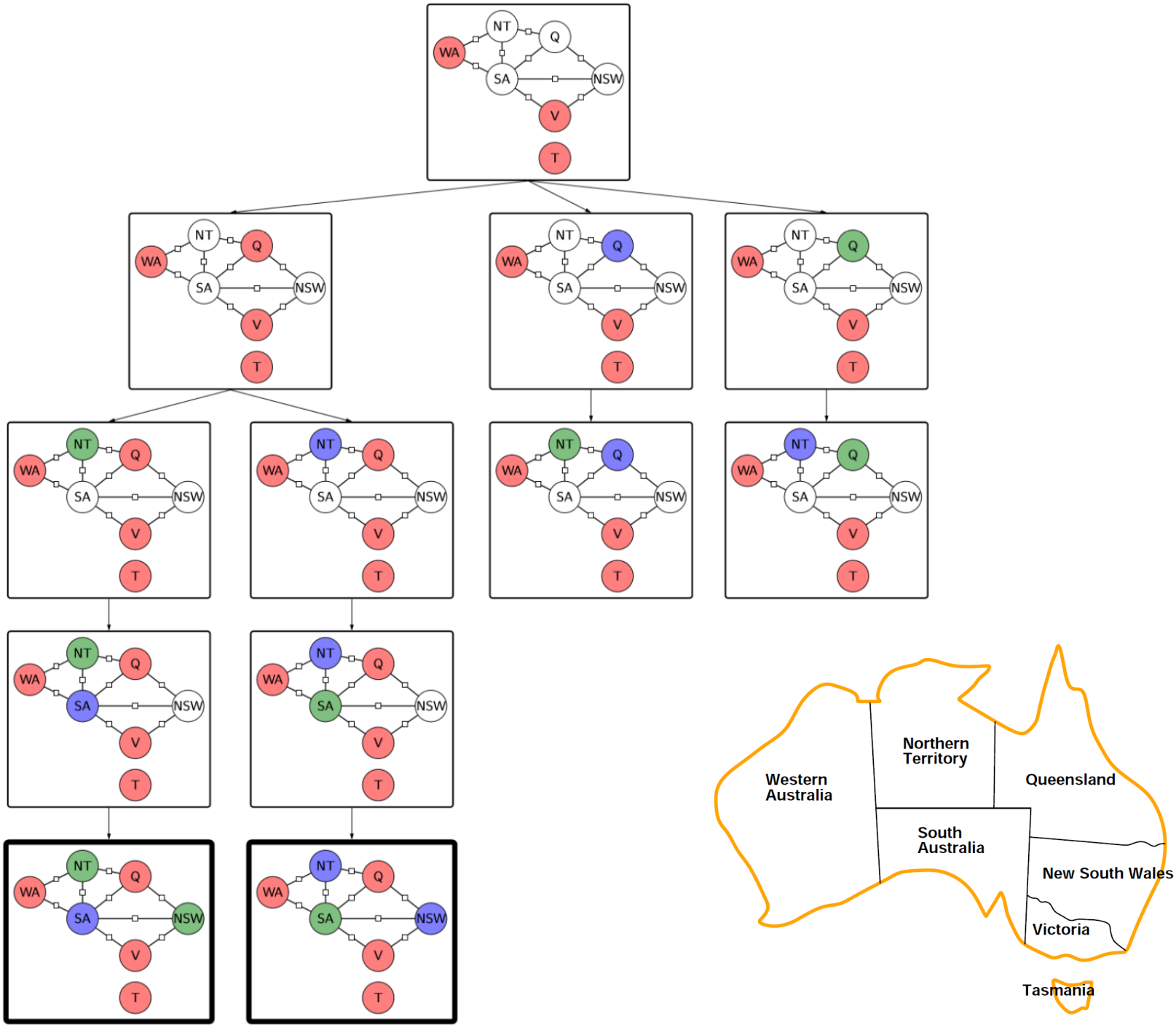

Northern Territory
Western Australia
Queensland
South Australia
New South Wales
Victoria
Tasmania


NT
Q
WA
SA
NSW
V
T


Northern Territory
Western Australia
Queensland
South Australia
New South Wales
Victoria
Tasmania


NT
Q
WA
SA
NSW
V
T

Northern Territory
Western Australia
Queensland
South Australia
New South Wales
Victoria
Tasmania


NT
Q
WA
SA
NSW
V
T

Western Australia
Northern Territory
Queensland
South Australia
New South Wales
Victoria
Tasmania

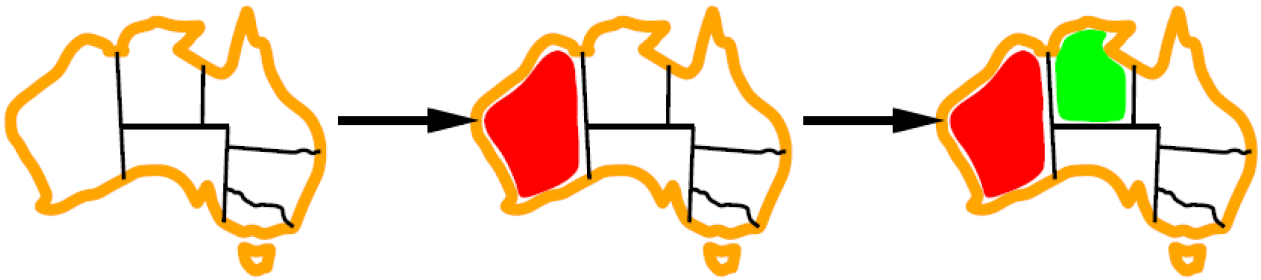
提高回溯效率

通过提供源自问题本身知识领域的特定启发式来提高无信息搜索算法的速度，现在发现无须领域知识也能有效提升求解CSP的速度：

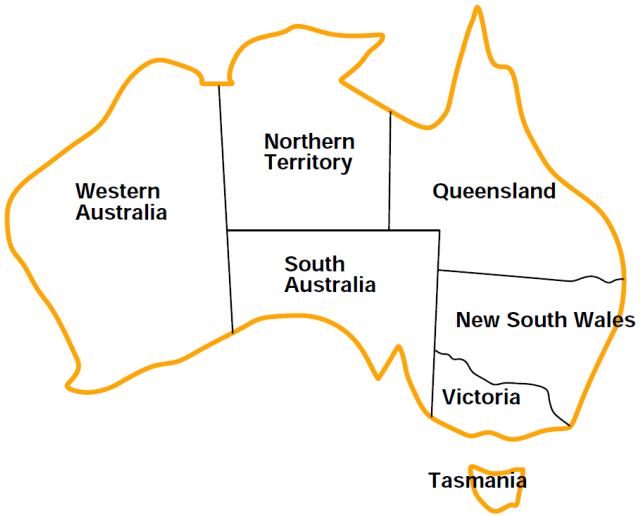
1. 接下来应对哪个变量赋值？
2. 应该以什么顺序尝试赋值？
3. 可以尽早发现不可避免的失败吗？
4. 可以利用问题结构吗？

最少剩余值启发式

- 最少剩余值（Minimum remaining values， MRV）
  选择“合法”取值最少的变量

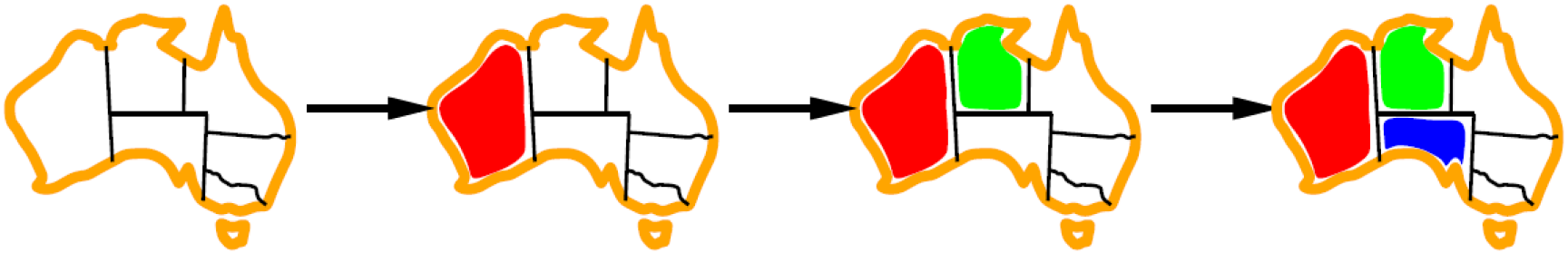


接下来应该给哪个变量赋值？

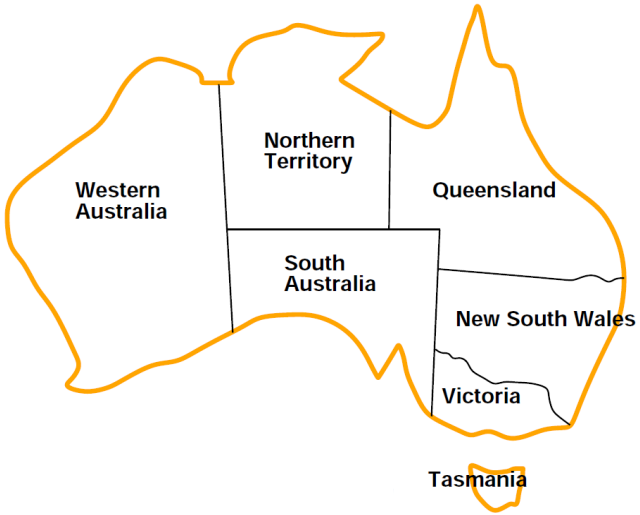


最少剩余值启发式

- 最少剩余值（Minimum remaining values， MRV）
  选择“合法”取值最少的变量

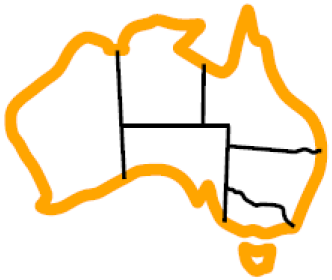


接下来应该给哪个变量赋值？

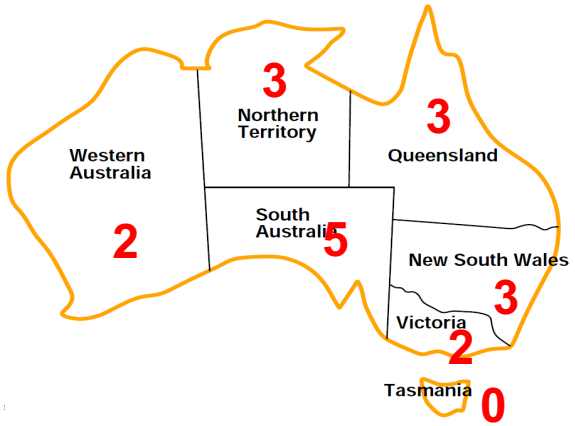


度启发式

- 最少剩余值启发式对选择第一个着色区域没有帮助，因为初始的时候都有三种着色可能。
- 度启发式：
  - 选择与未赋值变量约束最多的变量以降低未来分支因子

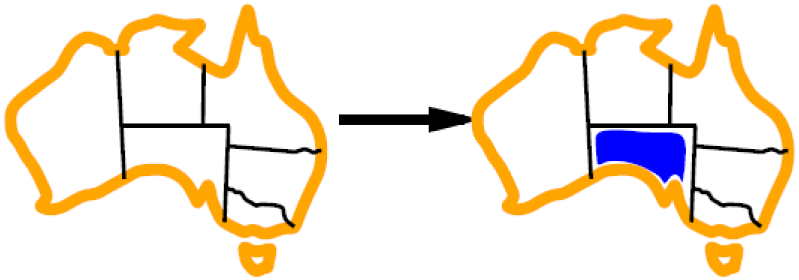


接下来应该给哪个变量赋值？

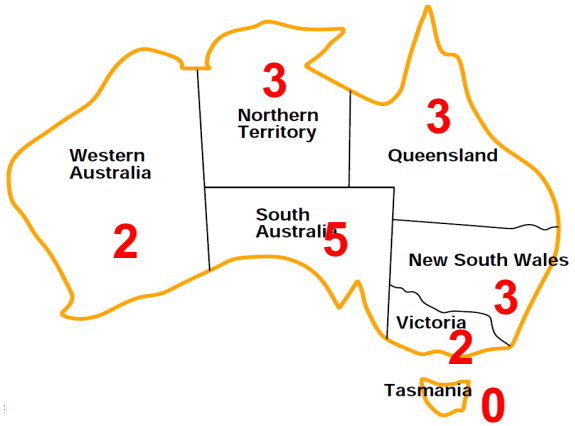


度启发式

- 最少剩余值启发式对选择第一个着色区域没有帮助，因为初始的时候都有三种着色可能。
- 度启发式：
  - 选择与未赋值变量约束最多的变量以降低未来分支因子



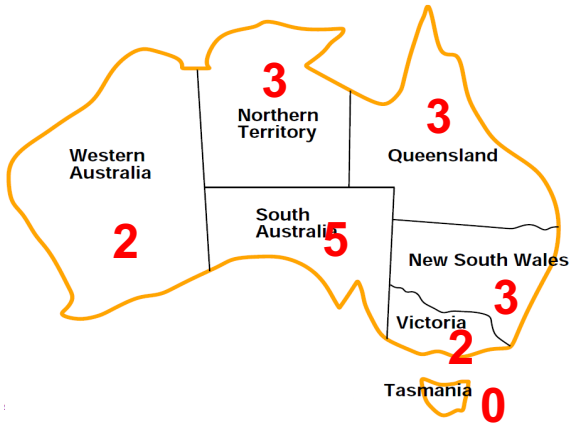
接下来应该给哪个变量赋值？



度启发式

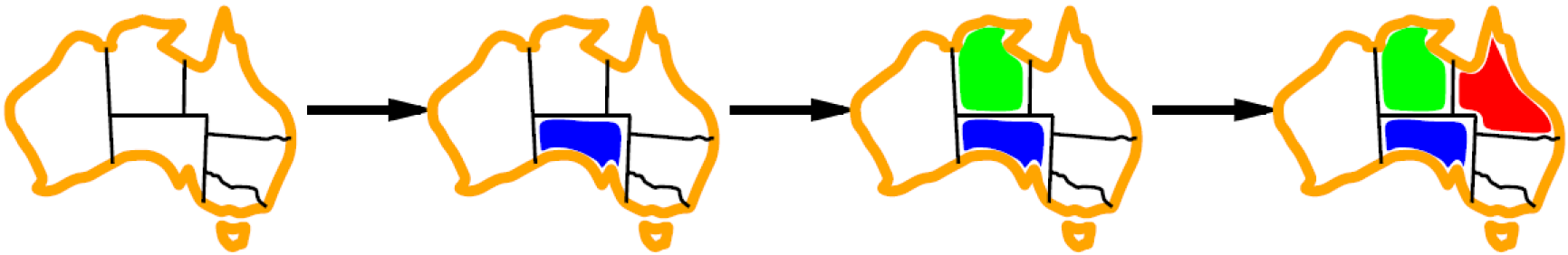
- 最少剩余值启发式对选择第一个着色区域没有帮助，因为初始的时候都有三种着色可能。
- 度启发式：
  - 选择与未赋值变量约束最多的变量以降低未来分支因子

接下来应该给哪个变量赋值？

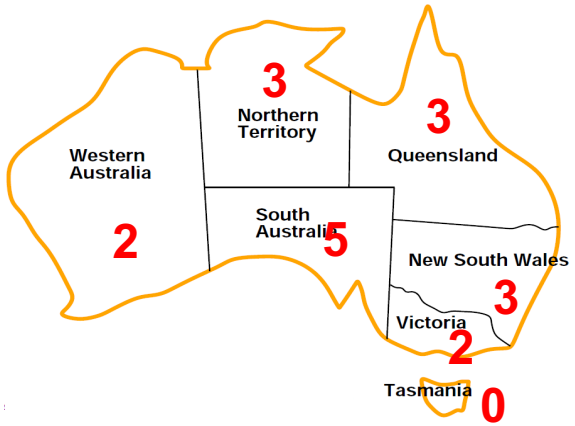


度启发式

- 最少剩余值启发式对选择第一个着色区域没有帮助，因为初始的时候都有三种着色可能。
- 度启发式：
  - 选择与未赋值变量约束最多的变量以降低未来分支因子

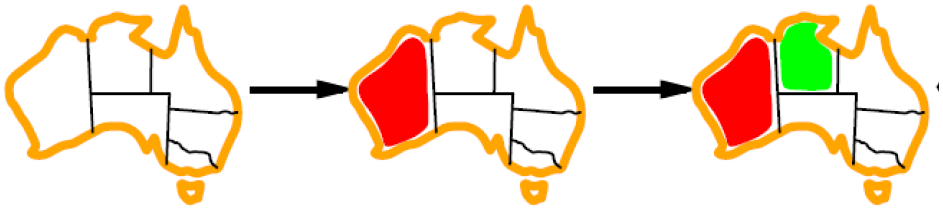


接下来应该给哪个变量赋值？

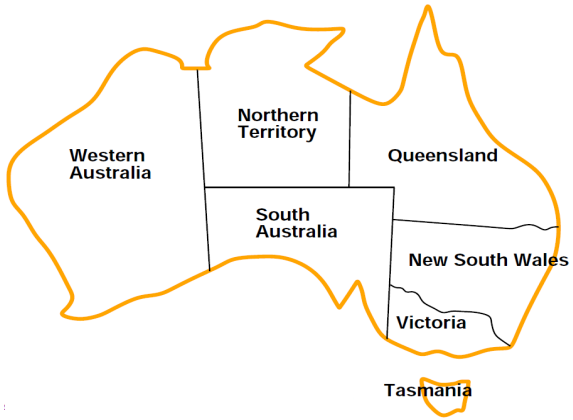


最少约束值启发式

- 给定一个变量，选择最少约束值
  - 排除剩余变量中最少的值的那个，试图为剩余变量赋值留下最大空间。

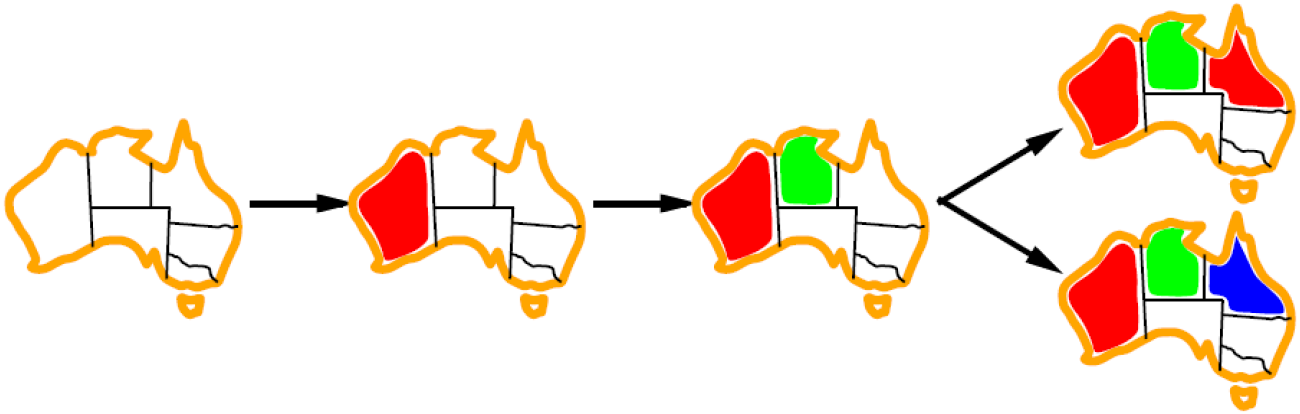


应该以什么顺序取值？

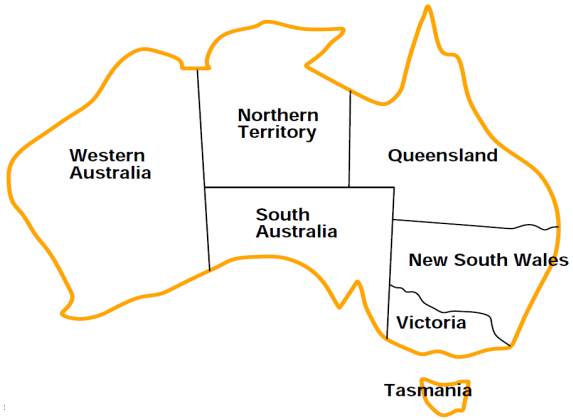


最少约束值启发式

- 给定一个变量，选择最少约束值
  - 排除剩余变量中最少的值的那个，试图为剩余变量赋值留下最大空间。

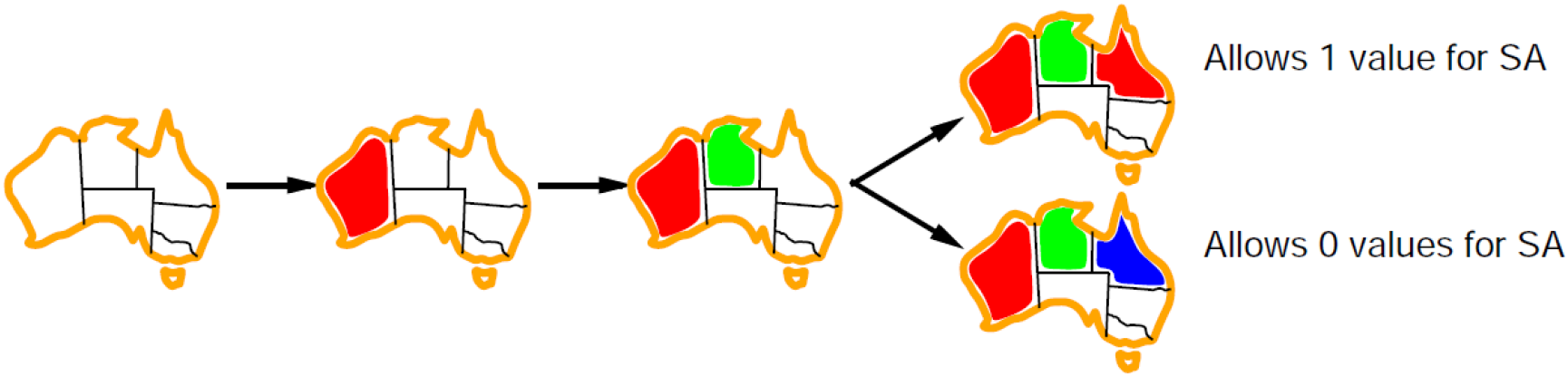


应该以什么顺序取值？

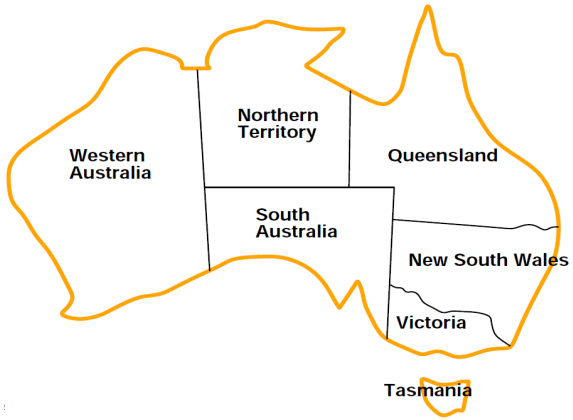


最少约束值启发式

- 给定一个变量，选择最少约束值
  - 排除剩余变量中最少的值的那个，试图为剩余变量赋值留下最大空间。

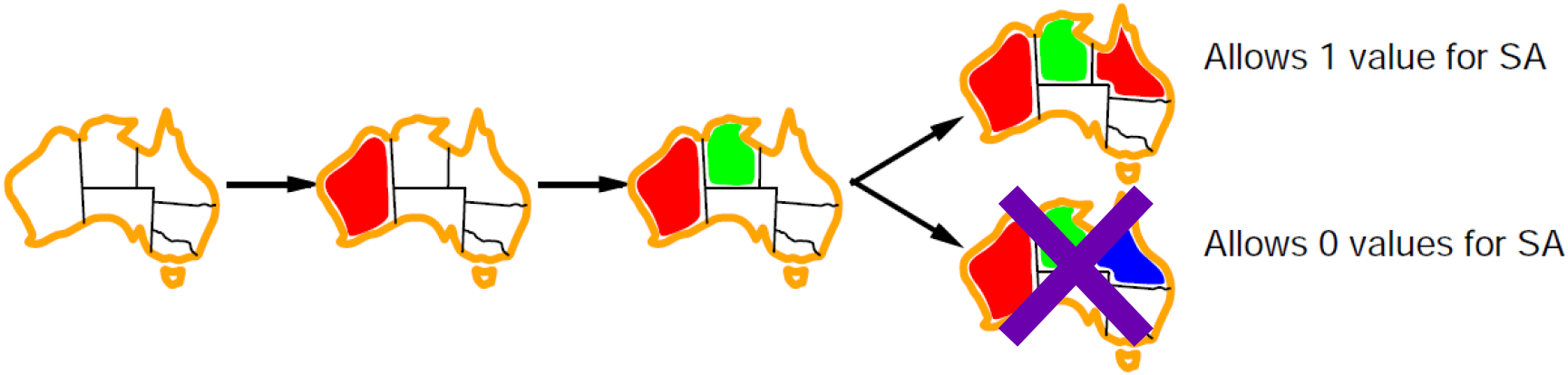


应该以什么顺序取值？

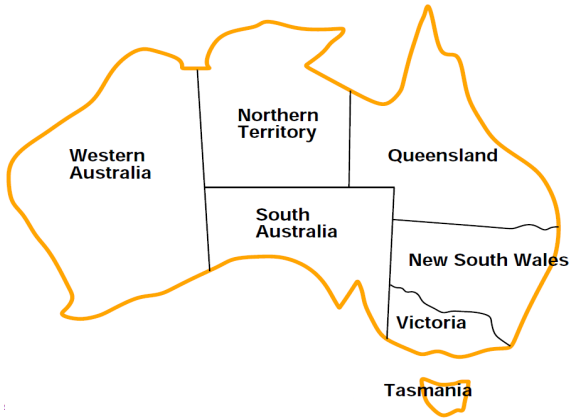


最少约束值启发式

- 给定一个变量，选择最少约束值
  - 排除剩余变量中最少的值的那个，试图为剩余变量赋值留下最大空间。

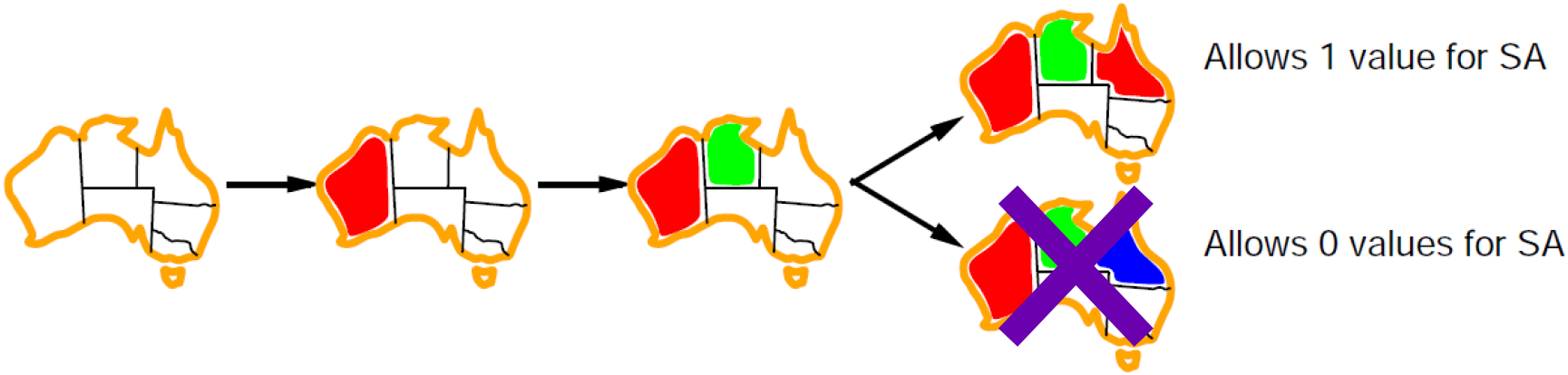


应该以什么顺序取值？

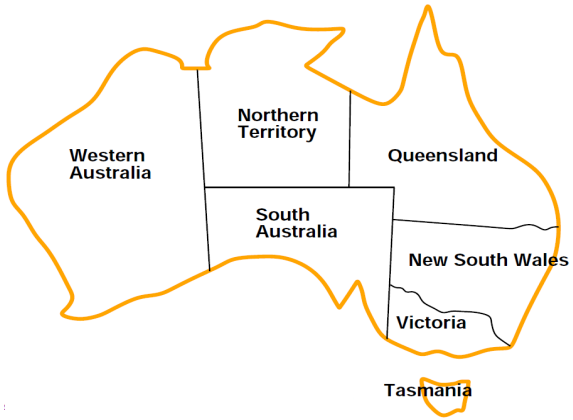


最少约束值启发式

- 给定一个变量，选择最少约束值
  - 排除剩余变量中最少的值的那个，试图为剩余变量赋值留下最大空间。

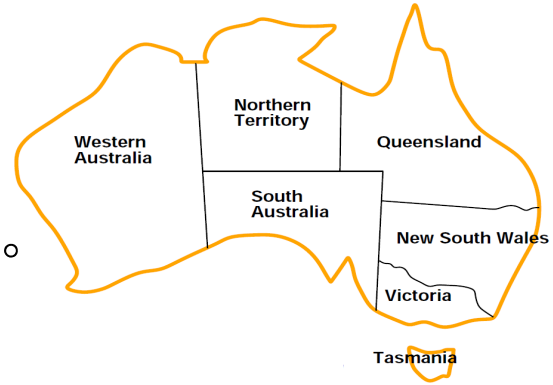


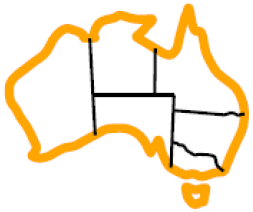
结合这些启发式方法可以使1000皇后问题成为可能。



前向检验

- 思想：
  - 跟踪未赋值变量的剩余合法值。
  - 过滤：清除错误的选项（违反约束）。
  - 当任何变量没有合法值时终止搜索。

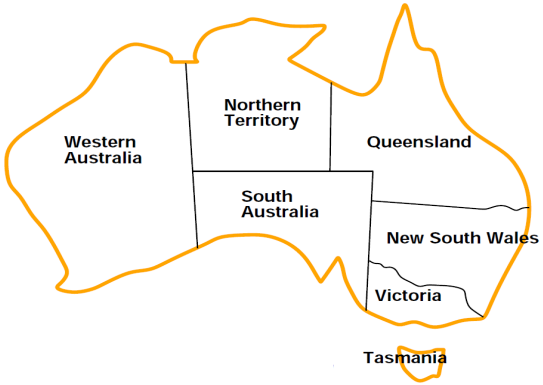


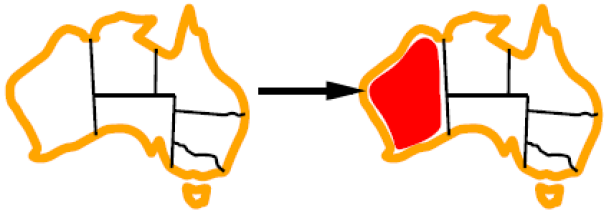


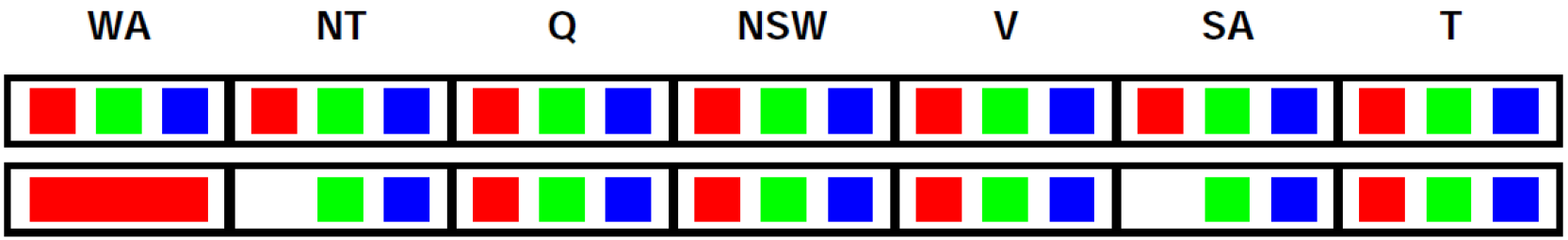
可以尽早发现不可避免的失败吗？

前向检验

- 思想：
  - 跟踪未赋值变量的剩余合法值。
  - 过滤：清除其违反约束的选择。
  - 当任何变量没有合法值时终止搜索。

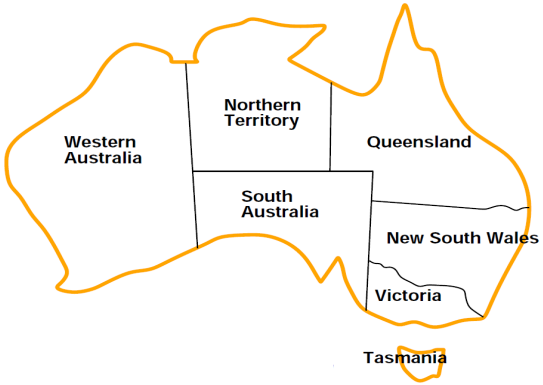


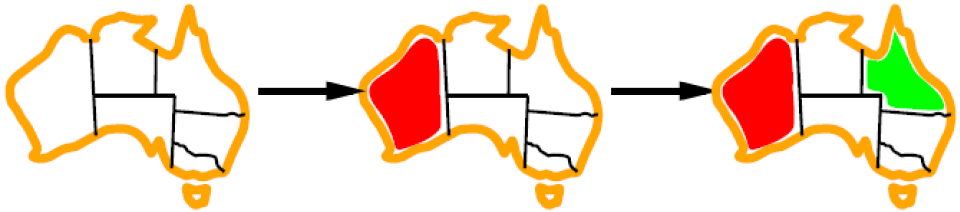




前向检验

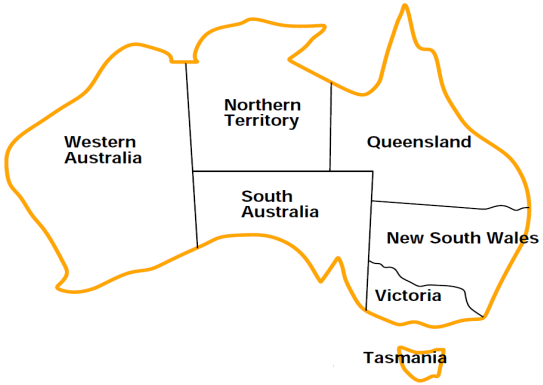
- 思想：
  - 跟踪未赋值变量的剩余合法值。
  - 过滤：清除其违反约束的选择。
  - 当任何变量没有合法值时终止搜索。

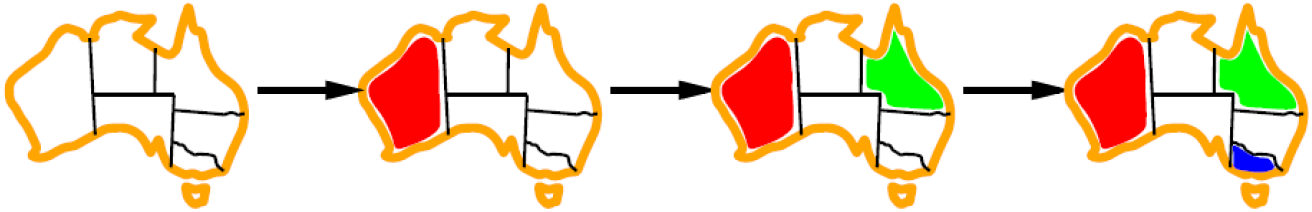




前向检验

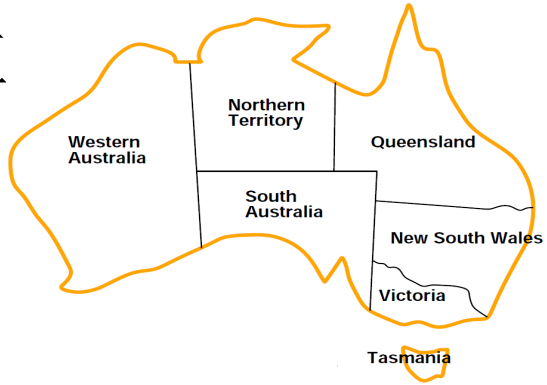
- 思想：
  - 跟踪未赋值变量的剩余合法值。
  - 过滤：清除其违反约束的选择。
  - 当任何变量没有合法值时终止搜索。

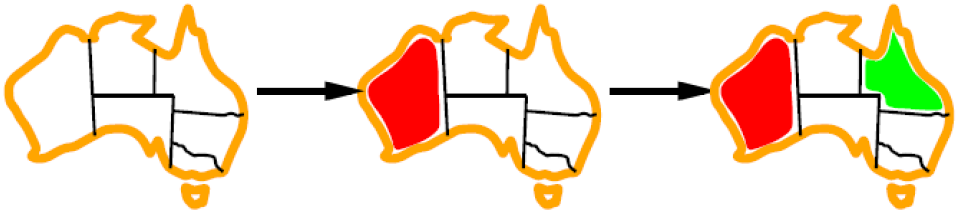


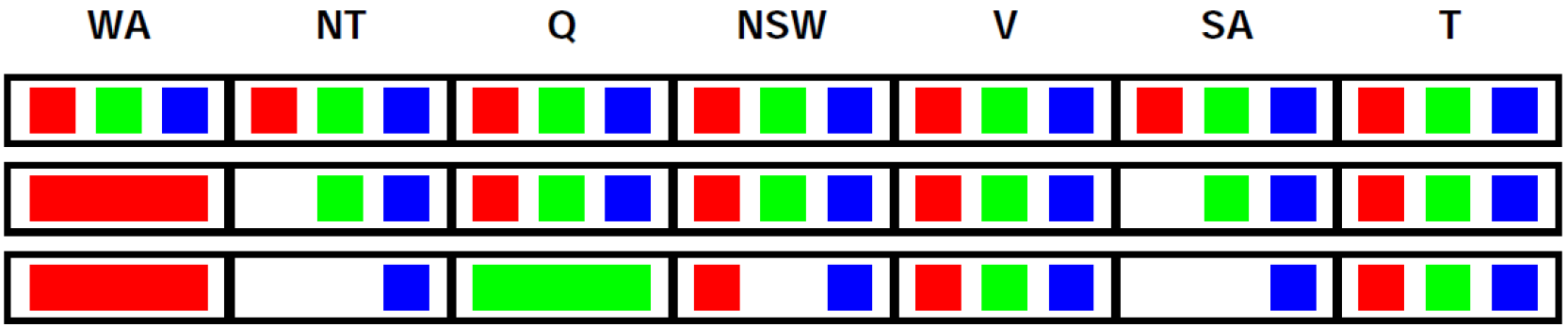


约束传播

- 前向检验将信息从赋值变量传播到未赋值变量，但不能为所有失败提供早期检验。



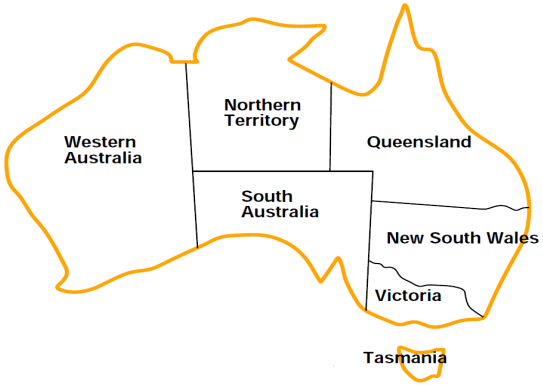


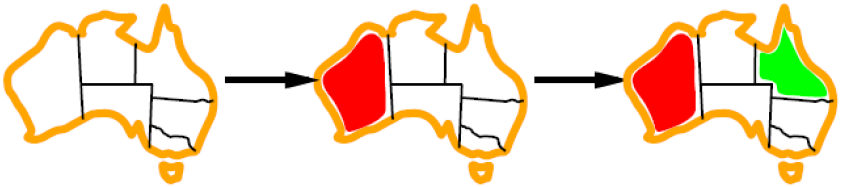


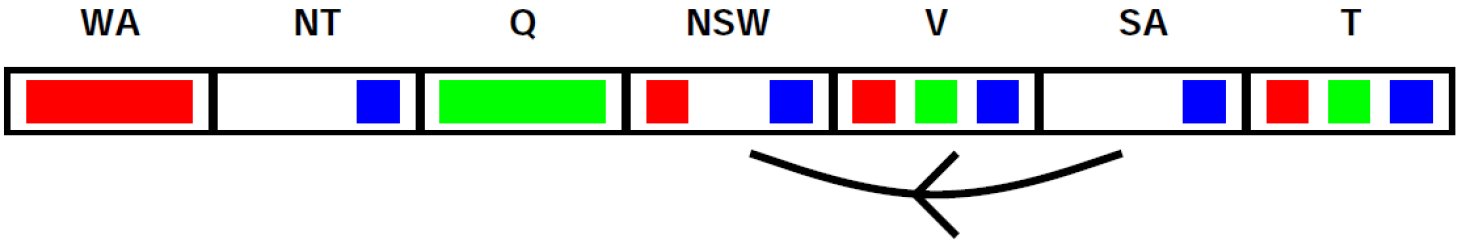
- NT和SA不能都是蓝色的！ 因为它们相邻。

弧相容

- 最简单的传播方式是使每个弧相容。
- 当且仅当对于X的每个值x都有某个允许的y时，X相对Y才是弧相容的。

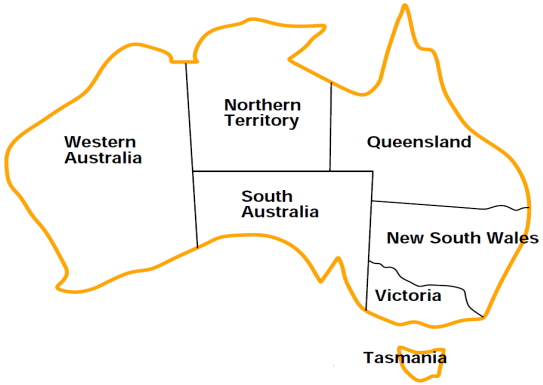


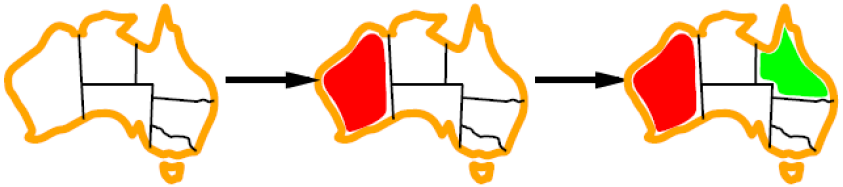


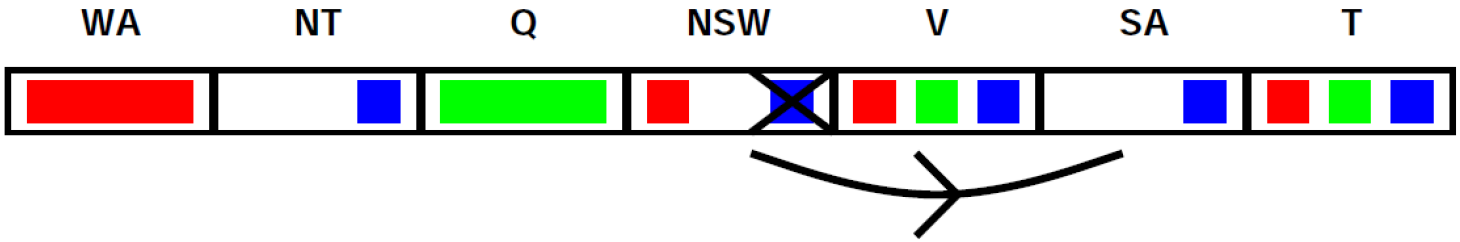


弧相容

- 最简单的传播方式是使每个弧相容。
- 当且仅当对于X的每个值x都有某个允许的y时，X相对Y才是弧相容的。

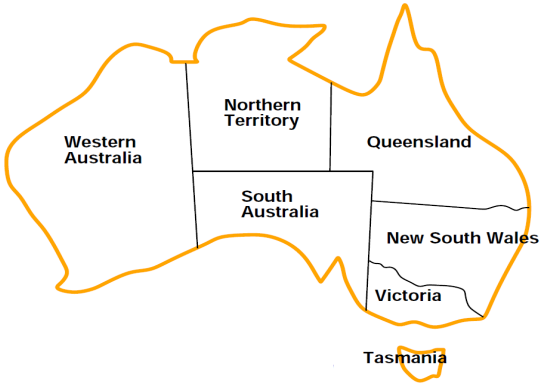


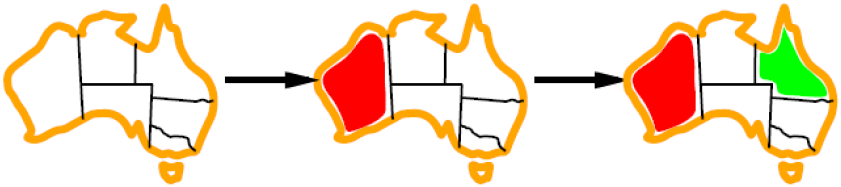


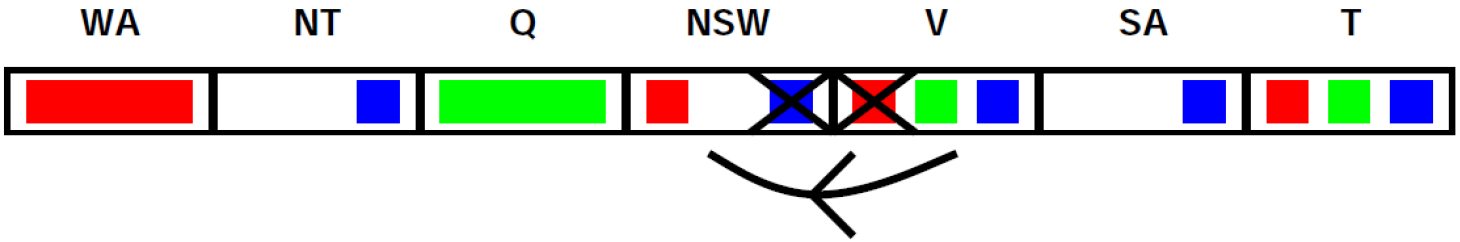


弧相容

- 最简单的传播方式是使每个弧相容。
- 当且仅当对于X的每个值x都有某个允许的y时，X相对Y才是弧相容的。



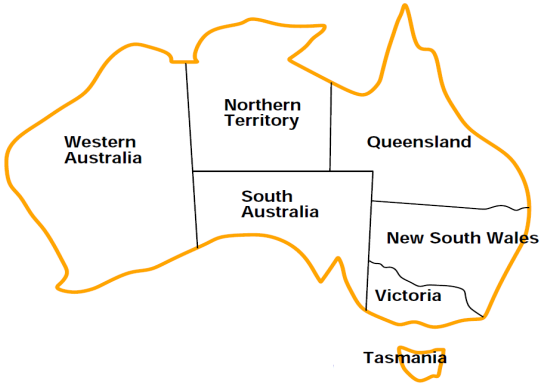


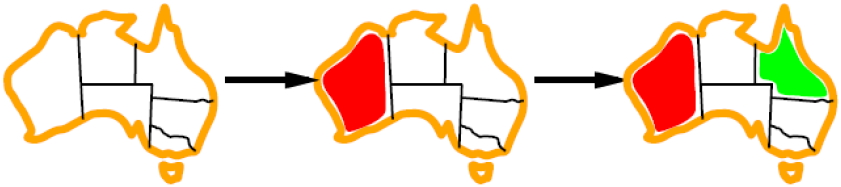


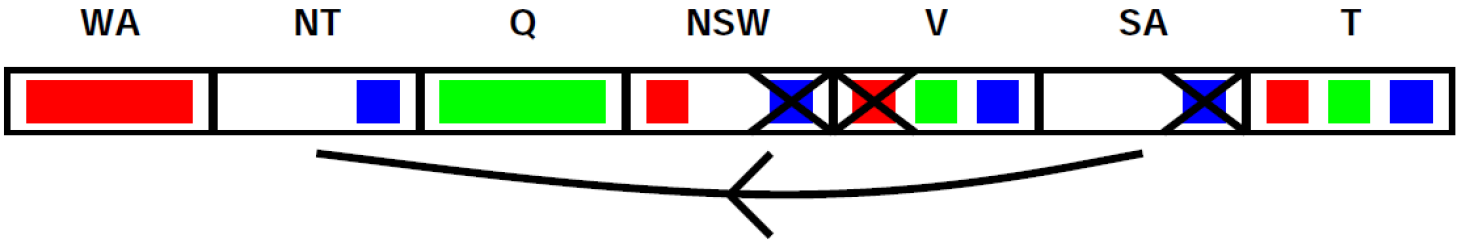
- 如果X丢失一个值，则需要重新检查X的邻域。

弧相容

- 最简单的传播方式是使每个弧相容。
- 当且仅当对于X的每个值x都有某个允许的y时，X相对Y才是弧相容的。







- 如果X丢失一个值，则需要重新检查X的邻域。
- 弧相容比前向检验更早地检测到失败。
- 可以作为预处理器执行，也可以在每次赋值后执行。

弧相容算法AC-3（Arc Consistent）

function AC-3(*csp*) **returns** false if an inconsistency is found and true otherwise
  inputs: *csp*, a binary CSP with components $(X,\ D,\ C)$
  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 REVISE(*csp*, $X_i,\ X_j$) **then**
      if size of $D_i\ =\ 0$ **then return** *false*
      for each X_k **in** X_i.NEIGHBORS - $\{X_j\}$ **do**
        add $(X_k,\ X_i)$ to *queue*
  return *true*

---

function REVISE(*csp*, $X_i,\ X_j$) **returns** true iff we revise the domain of X_i
  revised $\leftarrow$ *false*
  for each x **in** D_i **do**
    if no value y in D_j allows (x,y) to satisfy the constraint between X_i and X_j **then**
      delete x from D_i
      revised $\leftarrow$ *true*
  return *revised*

- 时间复杂度$O(n^2d^3)$，n是变量个数，d是变量值域最大元素个数。

举例

	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								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

- 约束：每行、每列、每个3*3方格内，数字不能重复。
- 最少剩余值：选择合法值最少的变量。
- 利用弧相容性。

$$A4 = \{1,2,3,4,5,6,7,8,9\}$$
$$A6 = \{1,2,3,4,5,6,7,8,9\}$$
$$E3 = \{1,2,3,4,5,6,7,8,9\}$$
$$E6 = \{1,2,3,4,5,6,7,8,9\}$$
$$I6 = \{1,2,3,4,5,6,7,8,9\}$$

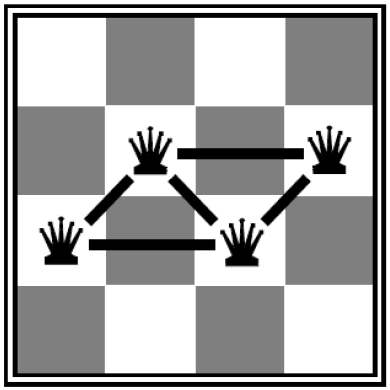
CSP局部搜索

- 爬山法和模拟退火通常在“完整”状态下工作，即赋值所有变量：初始状态是给每个变量都赋一个值，搜索过程是一次改变一个变量的取值。可能会违反一些约束。
  - 例如，8皇后问题。

- 局部搜索的目的：消除冲突。
- 变量选择：随机选择任何有冲突的变量。
- 通过最少冲突启发式选择值：
  - 选择违反约束最少的值。
  - 即“$h(n)$ =违反约束的总数”的爬山法。

举例：四皇后问题

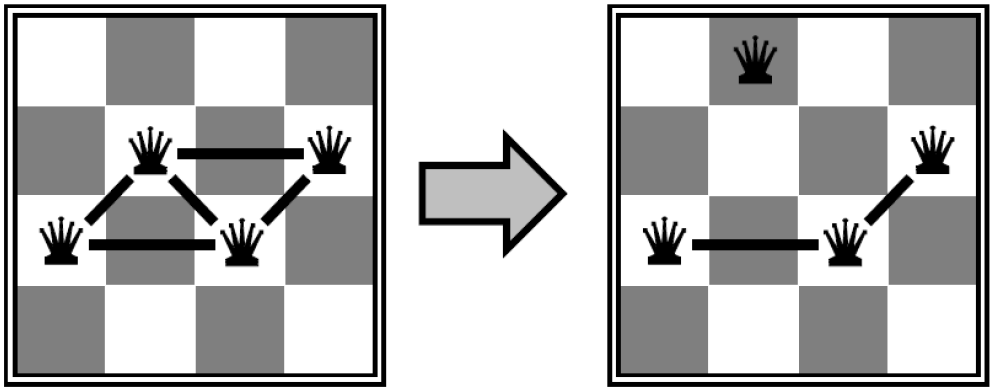
- 状态：四列四个皇后（$4^4 = 256$个状态）
- 行动：在列中移动皇后
- 目标测试：无攻击
- 评估函数： $h(n)$ =攻击的数量



h = 5

举例：四皇后问题

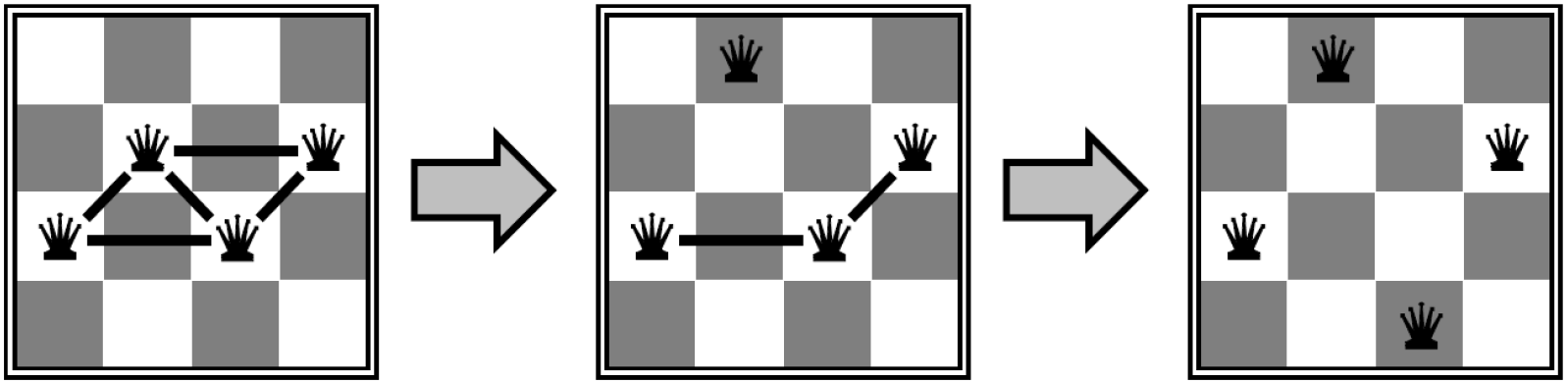
- 状态：四列四个皇后（$4^4 = 256$个状态）
- 行动：在列中移动皇后
- 目标测试：无攻击
- 评估函数： $h(n)$ =攻击的数量



h = 5　　h = 2

举例：四皇后问题

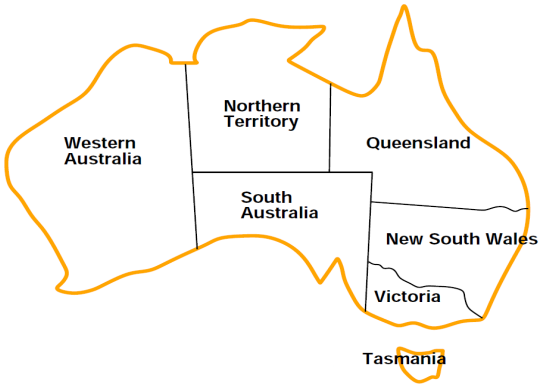
- 状态：四列四个皇后（$4^4 = 256$个状态）
- 行动：在列中移动皇后
- 目标测试：无攻击
- 评估函数： $h(n)$ =攻击的数量



- 给定任意初始状态，可以在几乎恒定的时间内对任意n求解n-皇后问题（例如，$n = 10{,}000{,}000$）。

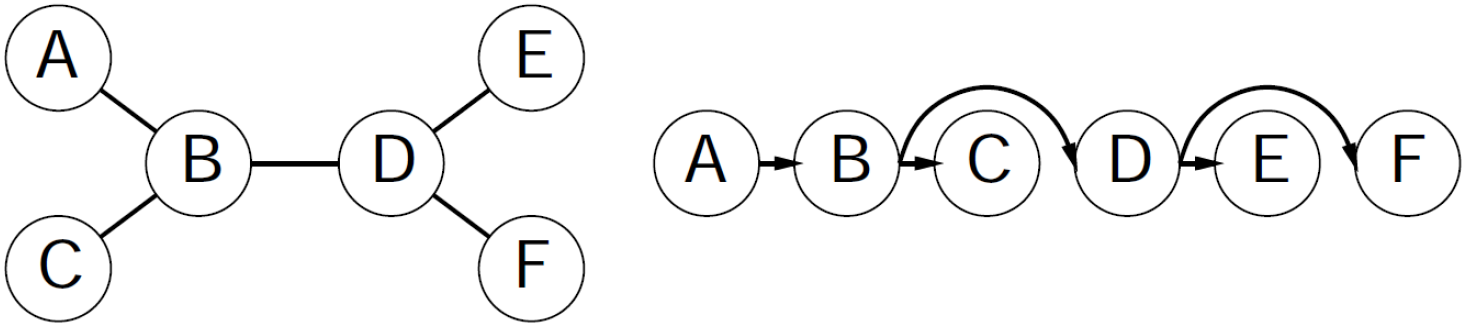
问题的结构与分解

- 以约束图表示的问题结构可用于快速找到解，而处理实际世界问题的唯一方法是将其分解为很多独立的子问题。



- 独立性可以简单地通过在约束图中寻找连通子图来确定。
- 每个连通子图对应于一个子问题CSP_i。如果赋值S_i是CSP_i的一个解，那么$\cup_i S_i$是$\cup_i CSP_i$的一个解。
- 假设每个CSP_i包含所有n个变量中的c个变量，c是常数，那么会有n/c个子问题，解决每个子问题最多花费d^c步工作，d是值域大小，总的工作量是$O(d^c n/c)$，是n的线性函数。
- 如果不分解，总的工作量是$O(d^n)$，是n的指数函数。

树结构CSP



- 定理：如果约束图没有循环（树结构），则CSP可以在时间$O(nd^2)$内求解。（线性时间）
- 求解树结构CSP：

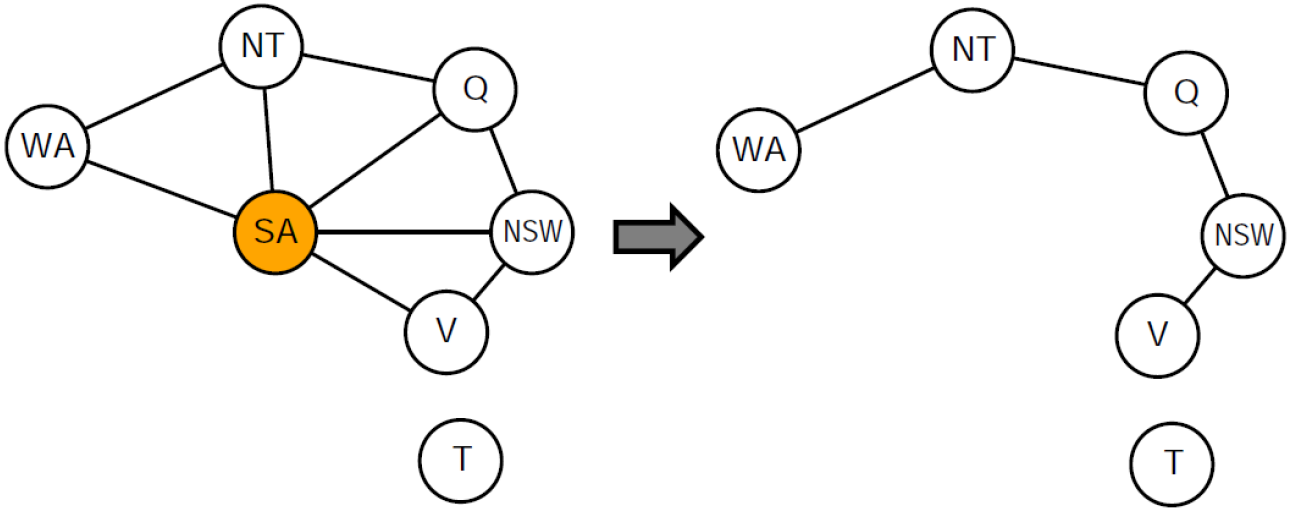
① 任意选择一个变量作为树的根，从根到叶对变量排序，以使每个变量在树中出现在父结点之后。（拓扑排序）

② 从$j = n, \dots, 2$，应用$\text{ARC} - \text{CONSISTENT}(\text{Parent}(X_j), X_j)$。

③ 从$j = 1, \dots, n$，对X_j赋值使其与$\text{Parent}(X_j)$相容。

近树结构CSP

- 条件：对一个变量赋值，删除这个变量、约束及与其邻接的值域。



- 割集条件：对一组变量赋值，使剩下的变量形成一棵树。
- 割集的大小为c $\Longrightarrow$ 总的运行时间为$O(d^c \cdot (n-c)d^2)$， c小非常快。

小结

- 约束满足问题CSP是一种特殊的问题：
  - 状态由一组固定变量的值定义。
  - 目标测试由对变量值的约束定义。
- 回溯搜索 = 每个结点分配一个变量的深度优先搜索。
- 变量排序和值选择启发式方法有很大帮助。
- 前向检验可防止导致以后失败的赋值。
- 约束传播（例如，弧相容）做额外工作来约束值并检测不相容（AC-3）。
- 在实践中，迭代式最少冲突通常很有效。
- 树结构CSP可以在线性时间内求解。

作业：课后题6.9和6.11

- 6.9 解释为什么在CSP搜索中，一个好的启发式选择变量的时候应该选择约束最多的变量，而选择值的时候应该选择受到约束最少的。

- 6.11 用AC-3算法说明弧相容对图6.1中的地图着色问题能够检测出部分赋值$\{WA = red, V = blue\}$的不相容。