人工智能原理与算法

第6章-约束满足问题

中国科学院自动化研究所
国科大人工智能学院
雷 震

- 定义约束满足问题
- 约束传播：CSP中的推理
- CSP的回溯问题
- CSP局部搜索
- 问题的结构

6.1 定义约束满足问题-问题引入

- 一个约束满足问题（CSP）包含三个部分$\mathcal{X}$、$\mathcal{D}$和$\mathcal{C}$：
  - $\mathcal{X}$： 变量集合$\{X_1, X_2, \ldots, X_n\}$
  - $\mathcal{D}$： 值域集合$\{D_1, D_2, \ldots, D_n\}$， X_i的值域D_i由其可能的取值$\{v_1, v_2, \ldots, v_k\}$组成
  - $\mathcal{C}$： 是约束集合，用来规定允许的值的组合
  - C_i 是有序对$< scope, rel >$，其中$scope$是约束中的变量组，rel定义了变量取值应满足的关系。

6.1 定义约束满足问题-问题引入

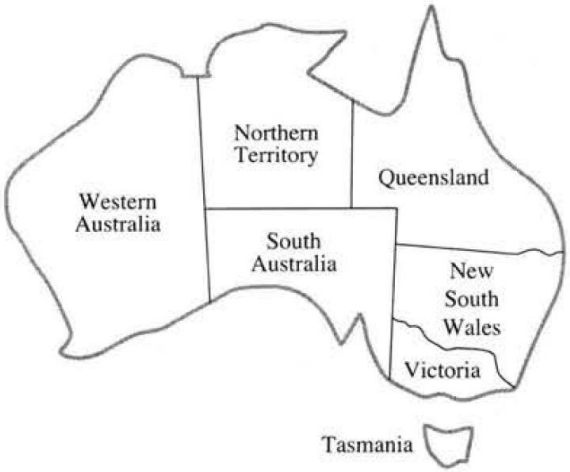
- 例如，如果 X_1 和 X_2 的域都是 $\{1,2,3\}$，那么约束“X_1 必须大于 X_2”可以表示为
  - $\langle (X_1, X_2), \{(3,1),(3,2),(2,1)\}\rangle$或$\langle (X_1, X_2), X_1 > X_2\rangle$。
- 为求解CSP，需要定义状态空间和解的概念
  - 问题的状态由对部分或全部变量的一个赋值来定义，

$$\{X_i = v_i, X_j = v_j, \ldots\}$$

  - 一个不违反任何约束条件的赋值称作相容的或者合法的赋值
  - 完整赋值是指每个变量都已赋值，部分赋值是指只有部分变量赋值。CSP的解是相容的、完整的赋值

6.1 定义约束满足问题-问题示例：地图着色

- 澳大利亚地图：每个州及边界
  - 任务：为地图每个区域涂上红色、绿色或者蓝色，要求相邻区域颜色不能相同



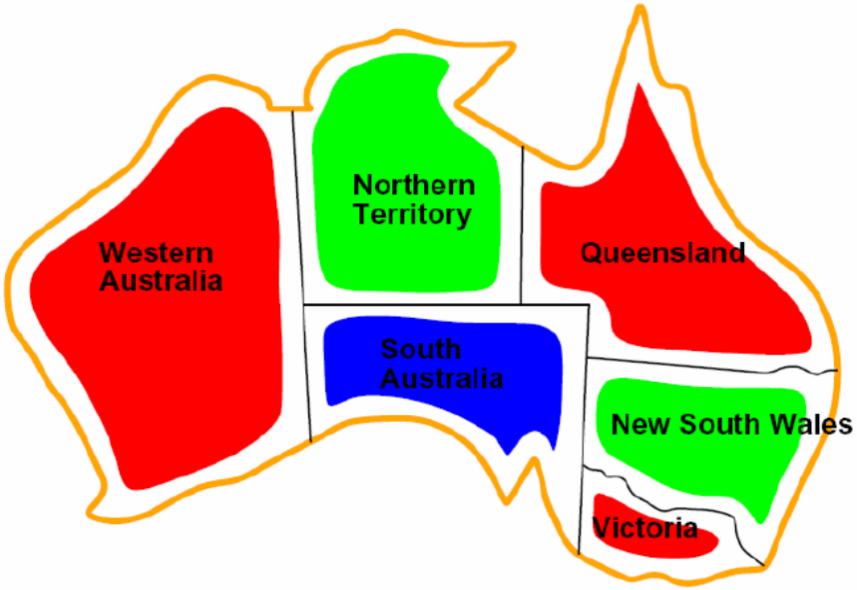
- 将其形式化为 CSP:
  - 图中区域定义为变量

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

  - 每个变量的值域$D_i = \{red, green, blue\}$
  - 约束: $C = \{SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V, WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V\}$
  - $SA \neq WA$是$< (SA, WA), SA \neq WA >$的快捷表示，$SA \neq WA$可以枚举如$\{(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)\}$

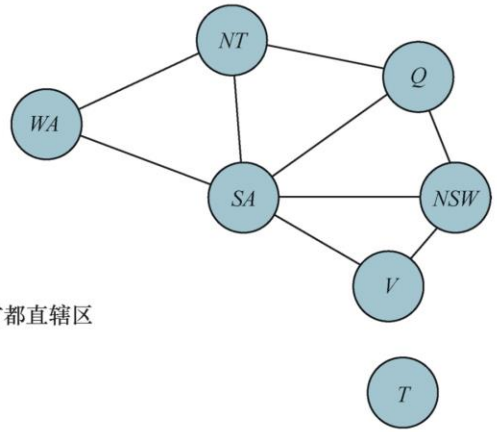
6.1 定义约束满足问题-问题示例：地图着色

- 这个问题有很多可能的解
  - 例如: $\{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = red\}$



6.1 定义约束满足问题-问题示例：地图着色

- 将 CSP 可视化为**约束图**(constraint graph) 。图的节点对应于问题的变量，图的边连接同一约束中的任意两个变量。
  - 形式为CSP的优势：相比于原子的状态空间搜索器，CSP 求解器可以快速消除大面积搜索空间。
  - 例如，一旦我们在澳大利亚问题中选择了$\{SA = blue\}$，就可以得出结论，它的 5 个相邻变量都不能取值为 blue。不使用约束的搜索过程必须考虑这 5 个相邻变量的**3^5=243**种赋值；有了约束，我们只需考虑**2^5=32**种赋值，计算量减少了 87%。



6.1 定义约束满足问题-问题示例：作业调度

- 工厂有很多日常工作调度问题，要满足各种约束。
  - 考虑汽车装配调度问题。整个作业由不同任务组成，我们可以将每个任务建模成一个变量，其中每个变量的值为任务开始时间，由整数分钟数表示。
  - 约束为“一个任务必须在另一个任务之前完成”(例如，安装车轮必须在安装轮毂盖之前完成)和“一次只能同时执行一定数量的任务”等断言。约束还可以指定任务完成所需的时间。

6.1 定义约束满足问题-问题示例：作业调度

- 汽车组装问题
  - 15 个任务：安装轮轴(axle)(前、后)，固定 4 个车轮(wheel)(左和右、前和后)，拧紧每个车轮的螺母(nuts)，固定轮毂盖(cap)，并检查(inspect)最终装配。可以将任务表示为 15 个变量:

$$X = \{Axle_F, Axle_B, Wheel_{RF}, Wheel_{LF}, Wheel_{RB}, Wheel_{LB}, Nuts_{RF}, Nuts_{LF}, Nuts_{RB}, Nuts_{LB}, Cap_{RF}, Cap_{LF}, Cap_{RB}, Cap_{LB}, Inspect\}$$

  - 变量的值是任务的开始时间
  - **优先约束**：任务T_1必须在任务T_2前完成，$T_1 + d_1 \leq T_2$

如轮轴先于车轮安装，安装需要10 分钟，则约束如下：

$$Axle_F + 10 \leq Wheel_{RF}; Axle_F + 10 \leq Wheel_{LF};$$

$$Axle_B + 10 \leq Wheel_{RB}; Axle_B + 10 \leq Wheel_{LB};$$

6.1 定义约束满足问题-问题示例：作业调度

- 汽车组装问题
  - 接下来，我们必须固定每个车轮(需要1分钟)，拧紧螺母(2分钟)，最后安装轮毂盖(1分钟， 但暂未表示):

$$Wheel_{RF} + 1 \leq Nuts_{RF}; Nuts_{RF} + 2 \leq Cap_{RF};$$

$$Wheel_{LF} + 1 \leq Nuts_{LF}; Nuts_{LF} + 2 \leq Cap_{LF};$$

$$Wheel_{RB} + 1 \leq Nuts_{RB}; Nuts_{RB} + 2 \leq Cap_{RB};$$

$$Wheel_{LB} + 1 \leq Nuts_{LB}; Nuts_{LB} + 2 \leq Cap_{LB};$$

  - 假设有4个工人来安装车轮，但他们必须共用一个工具来辅助安装轮轴。此时需要一个**析取约束**表示 $Axle_F$和 $Axle_B$在时间上不能重合

$$Axle_F + 10 \leq Axle_B \text{ 或 } Axle_B + 10 \leq Axle_F$$

  - 这一约束看起来更加复杂，结合了算术约束和逻辑约束。

6.1 定义约束满足问题-问题示例：作业调度

- 汽车组装问题
  - 定义断言：检查时最后一项任务，需要3分钟。对除$Inspect$之外的每个变量，加入形如 $X + d_x \leq Inspect$的约束。
  - 假设整个过程只有30分钟，这样得到所有变量的值域

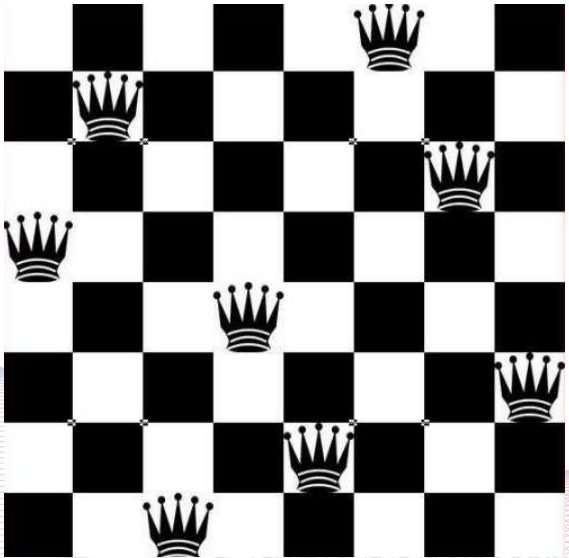
$$D_i = \{1,2,3,\ldots,30\}$$

  - CSP适合解决此类问题，即使约束变量有几千个。在一些情况下，有的约束过于复杂，难以形式化，可使用更先进的规划技术

6.1 定义约束满足问题-CSP形式体系的变体

- CSP的形式化
  - 最简单的 CSP 所涉及的变量具有**离散有限域**。地图着色问题和带有时间限制的调度问题都属于这类问题。
  - 8 皇后问题也可以看作是一个有限域 CSP， 其中变量$Q_1, \ldots, Q_8$对应第 $1 \sim 8$ 列中的皇后，每个变量的域为该列皇后可能的行号，

$D_i = \{1, 2, 3, 4, 5, 6, 7, 8\}$。约束为不允许两个皇后在同一行或同一对角线上。



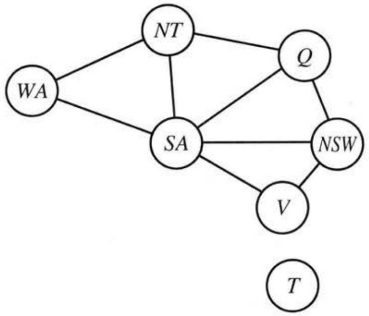
6.1 定义约束满足问题-CSP形式体系的变体

- 离散值域如果是**无限的**，直接用约束语言替代
  - 例如，$T_1 + d_1 \leq T_2$，不再可能枚举(T_1, T_2)所有可能的赋值来求解
  - 对于整数变量的**线性约束**，存在特殊的求解算法（暂不讨论）
  - 可以证明，不存在求解整数变量上一般**非线性约束**的算法——这个问题是不可判定的
- **连续值域**的约束满足问题在现实世界中十分常见，在运筹学领域有广泛的研究
  - 最著名的一类连续值域CSP是线性规划问题，约束必须是线性等式或线性不等式，可以在变量个数多项式时间内求解
  - 二次规划、二阶二次曲线规划

6.1 定义约束满足问题-CSP形式体系的变体

- 考察CSP约束的类型
  - **一元约束**：只限制单个变量取值，如地图着色问题中南澳洲人不喜欢绿色

    $< (SA), SA \neq green >$
  - **二元约束**：与两个变量有关，例如，$SA \neq NSW$，可以表示为约束图



  - **高阶约束**：如断言Y的值处于X和Z之间，可以表示成三元约束

    $Between(X, Y, Z)$
  - 变量个数任意的约束称为**全局约束**。例如，Alldiff，指的是约束中所有变量必须取不同的值，在数独游戏中，一行或一列所有变量必须满足Alldiff 约束

6.1 定义约束满足问题-CSP形式体系的变体

- 密码算术谜题
  - 密码算术中每个字母表示不同的数字，表示为六变量约束

    $Alldiff(F,T,U,W,R,O)$
  - 此谜题的四列算式可表示为如下涉及多个变量的n元约束

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

$$O + O = R + 10 \cdot C_1$$

$$C_1 + W + W = U + 10 \cdot C_2$$

$$C_2 + T + T = O + 10 \cdot C_3$$

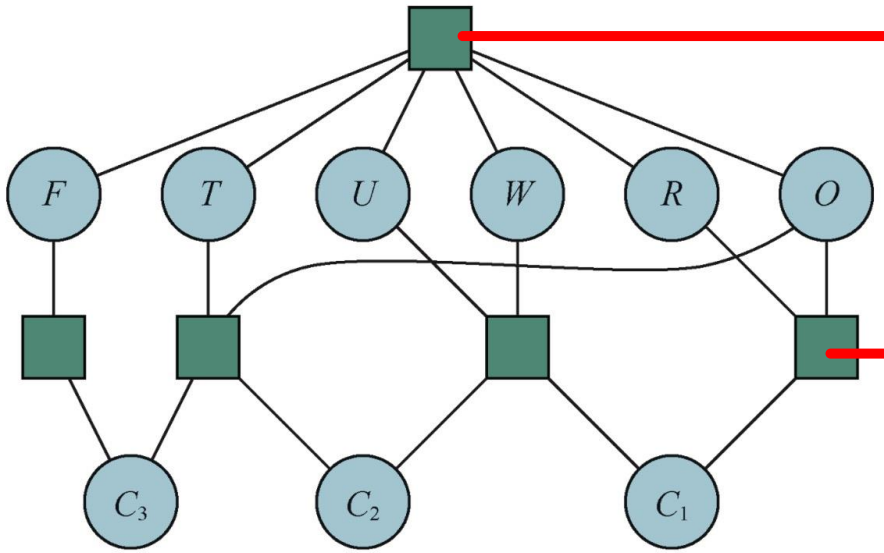
$$C_3 = F$$

  - 其中，C_1，C_2，C_3表示十、百、千位上的进位变量

6.1 定义约束满足问题-CSP形式体系的变体

- 密码算术谜题
  - 这些约束可以用**约束超图**表示
  - 圆圈结点为普通节点
  - 方块表示n元约束的超节点
  - C_1， C_2， C_3表示每列进位

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



（b）

Alldiff约束

相加约束

6.1 定义约束满足问题-CSP形式体系的变体

- **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为**二元约束**，所以可以转换任何CSP成只含**二元约束**
- 另一种将n元CSP转换成二元CSP的方法是**对偶图转换**
  - 创建一个新图，原图中**每个约束用一个变量**表示，每对有**同样变量的约束**用一个**二元约束表示**
  - 例如：如果原图中有$\{X, Y, Z\}$ 和约束$< (X, Y, Z), C_1 >$、$< (X, Y), C_2 >$，对偶图中则有两个变量$\{C_1, C_2\}$和二元约束$< (X, Y), R_1 >$，(X, Y)是两个约束的共有变量，R_1为定义共有变量约束的新关系

6.1 定义约束满足问题-CSP形式体系的变体

- Alldiff比二元约束描述问题更简单更不容易出错
- 对全局约束可能设计专用的推理方法
- 绝对约束和偏好约束
  - 针对前者，例如在大学排课问题中：没有教授可以同一时间出现在不同教室
  - 针对后者，例如R教授可能偏好上午授课，而N教授偏好在下午授课
- **偏好约束**通常被处理成**个体变量赋值开销**，这时CSP可以用基于路径或局部的最优搜索方法求解，称为约束优化问题，简写为**COP**

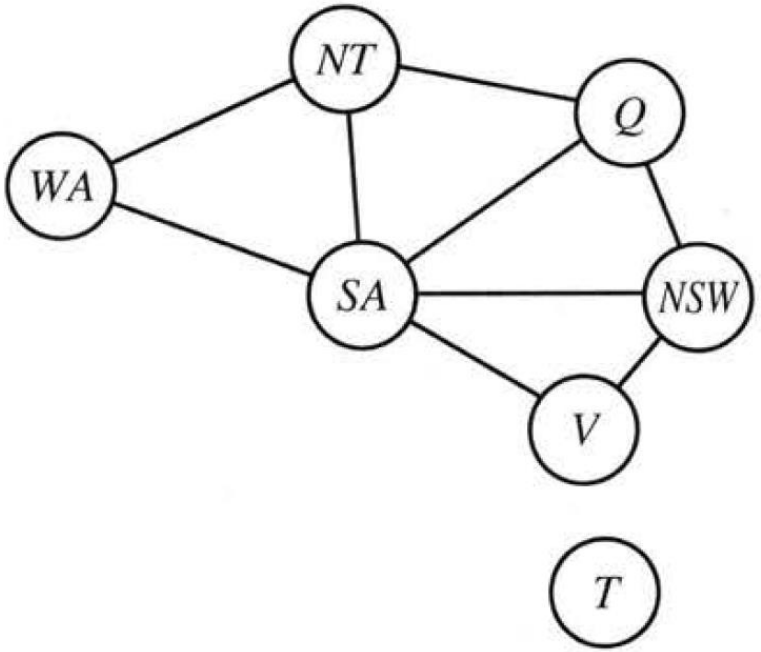
- 定义约束满足问题
- 约束传播：CSP中的推理
- CSP的回溯问题
- CSP局部搜索
- 问题的结构

6.2 约束传播-CSP中的推断

- 在常规的状态空间搜索中，算法只能做一件事：搜索（从几种可能性中选择新的变量赋值）
- 在CSP中，算法可以搜索，可以做一种称为约束传播的特殊推理：
  - 使用约束来减小一个变量的合法取值范围
  - 影响跟此变量有约束关系的另一变量的取值
  - 约束与搜索可以交替进行，也可以作为搜索前的预处理步骤
- 核心思想：局部相容性

6.2 约束传播-CSP中的推断

- 将每个变量看作图中的一个节点，将每个二元约束看作一条边，则增强图中每一部分局部相容性的过程会导致整个图中不相容的值被删除。



6.2 约束传播-节点相容

- 结点相容：如果单个变量值域中的所有取值满足它的一元约束，就称此变量是结点相容的
  - 例如，地图着色问题中，南澳洲人不喜欢绿色，变量SA值域变化 $\{red, green, blue\} -> \{red, blue\}$，即为结点相容
  - 如果网络中每个变量都是结点相容的，则此网络是结点相容的
  - 结点相容总能消除所有一元约束
  - 所有n元约束均可转换为二元约束，故定义只含有二元约束的CSP求解器

6.2 约束传播-弧相容

- 如果CSP中某变量值域中的所有取值满足该变量的所有二元约束，则成此**变量是弧相容**的。
- 更形式地，对于变量 X_i 、 X_j ，若对D_i中每个数值在D_j中都存在一些数值满足弧(X_i, X_j)的二元约束，则称X_i**相对**X_j**是弧相容**的。
- 如果每个变量相对其他变量都是弧相容的，则称该**网络是弧相容**的。

6.2 约束传播-弧相容

- 例子：考虑约束$Y = X^2$，X和Y都是数字，显示地写出约束为

  $$< (X, Y), \{(0,0), (1,1), (2,4), (3,9)\} >$$

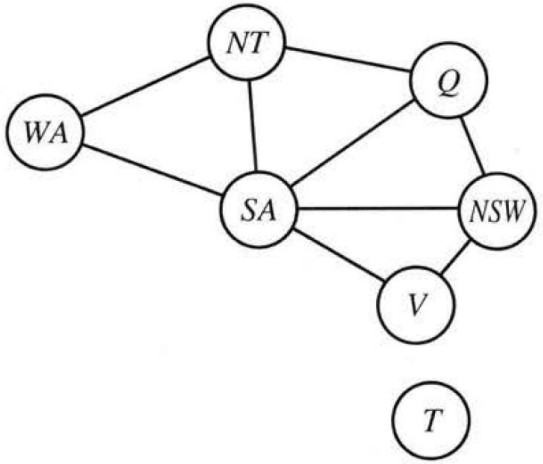
  - X相对于Y是弧相容的，$X = \{0，1，2，3\}$
  - Y相对于X是弧相容的，$Y = \{0，1，4，9\}$
  - 整个CSP就是弧相容的

6.2 约束传播-弧相容

- 考虑澳大利亚地图着色问题，考虑(SA, WA)之间的不同色约束：

$$\{(red, green), (red, blue), (green, red),$$
$$(green, blue), (blue, red), (blue, green)\}$$

- 不管如何为SA（或WA）选择取值，另一个都还有合法取值
- 此时应用弧相容对变量的值域没有任何影响



6.2 约束传播-弧相容

- 最流行的弧相容算法是AC-3算法（维护弧相容队列）

function AC-3(*csp*) **returns** false（如果发现不一致）或true（其他情况）
  queue ← 一个弧的队列，初始化时包含*csp*中的所有弧

  while *queue*不空 **do**
    (X_i, X_j) ← POP(*queue*)
    if REVISE(*csp*, X_i, X_j) **then**
      if D_i的规模 = 0 **then return** false
      for each X_k **in** X_i.NEIGHBORS − $\{X_j\}$ **do**
        将(X_k, X_i)添加到*queue*中
  return true

function REVISE(*csp*, X_i, X_j) **returns** true当且仅当我们修改X_i的域
  revised ← false
  for each *x* **in** D_i **do**
    if D_j中不存在使(x,y)满足(X_i, X_j)约束的值 **then**
      将*x*从D_i中删除
      revised ← true
  return *revised*

6.2 约束传播-弧相容

- 最流行的弧相容算法是AC-3算法（维护弧相容队列）
  - 初始化一个队列包含所有弧
  - 弹出弧(X_i, X_j),若D_i未发生变化，则处理下一条弧。否则将所有指向X_i的弧(X_k, X_i)重新插入队列准备检验
  - 如果D_i变成空集，整个CSP没有相容解，AC-3返回失败
  - 否则，继续检查，试图缩小变量值域直到队列中没有弧

6.2 约束传播-弧相容

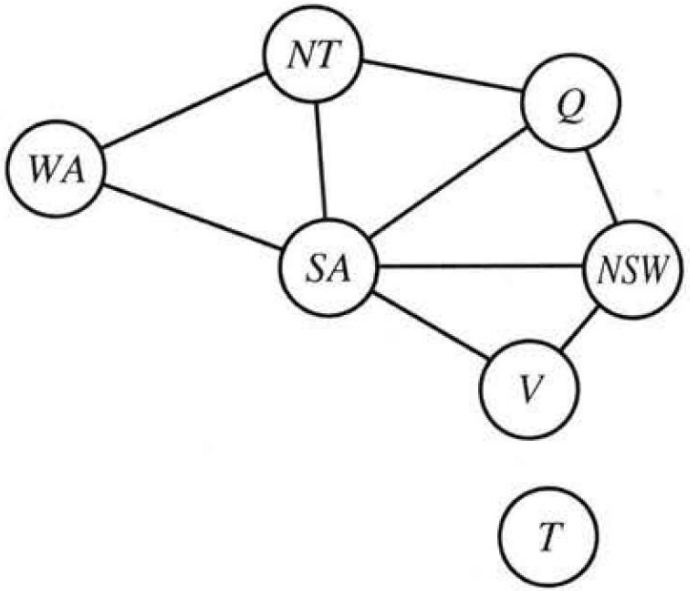
- AC-3的算法复杂度分析如下：
  - 假设CSP中有n个变量，变量值域最大为d个元素，c个二元约束。每条弧(X_i, X_j)最多只能插入队列d次，因为X_i至多有d个值可删除。检验一条弧的相容性可以在$O(d^2)$时间内完成，最坏情况算法的时间复杂度为$O(cd^3)$

6.2 约束传播-弧相容

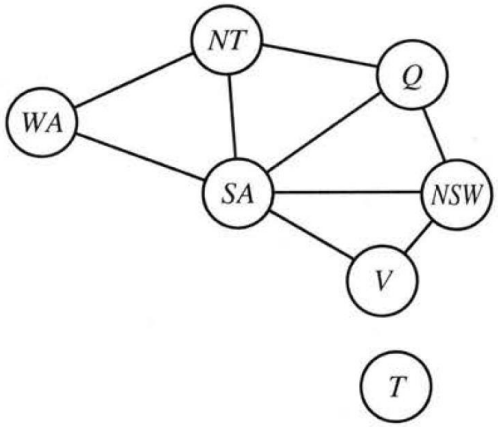
- 将弧相容概念扩展成处理n元约束是可能的，通常称为**通用弧相容**或**超弧相容**
- 变量X_i相对某n元约束是通用弧相容的，指的是对X_i值域中的每个值v都存在一组取值，其中X_i的值为v，其他值都在对应变量的值域中
- 所有变量值域为$\{0,1,2,3\}$，变量X对约束$X < Y < Z$相容，需要从X的值域中把2和3删除
- 弧相容可能缩小变量的值域，甚至能直接找到解（每个变量的值域大小都为1时）
- 或者有时发现CSP无解（一些变量值域大小为0）
- 对于有些网络，弧相容会失败，无法做出足够的推理

6.2 约束传播-弧相容失败

- 两色（红和蓝）的澳大利亚着色问题
  - 相容赋值$\{WA = red, SA = blue\}$和$\{WA = blue, SA = red\}$



6.2 约束传播-弧相容失败



- 需要更强的相容概念
- 路径相容
  - 两个变量的集合$\{X_i, X_j\}$对于第三个变量X_m是相容的，指对$\{X_i, X_j\}$的每一个相容赋值$\{Xi = a, X_j = b\}$，X_m都有合适的取值同时使得$\{Xi, X_m\}$和$\{Xm, X_j\}$是相容的
  - 从X_i途径X_m到X_j的路径
  - 使集合$\{WA, SA\}$对NT路径相容
  - 相容赋值$\{WA = red, SA = blue\}$和$\{WA = blue, SA = red\}$
  - 则NT既不能是red也不能是blue，故两种相容赋值均要删除，无解！

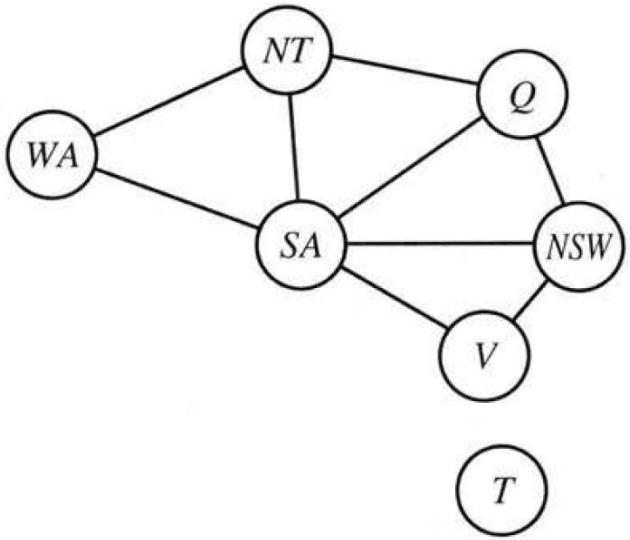
6.2 约束传播-k-相容

- 定义
  - 如果对于任何$k-1$个变量的相容赋值，第k个变量总能被赋予一个和前$k-1$个变量相容的值，则此CSP是k相容的
- 特点
  - 1-相容，2-相容，3-相容分别对应结点、弧、路径相容
  - 强k相容：如果一个图满足任意 $j(1 \leq j \leq k)$，都是j-相容的
  - 求解有n个结点且强n相容的CSP

    对每个变量X_i，搜索值域找到与$X_j(1 \leq j \leq i-1)$相容的值，$O(n^2d)$
  - 建立n相容的算法，在最坏情况下，时间和空间复杂度均是n的指数级

6.2 约束传播-全局约束

- 全局约束，可能涉及任意个约束变量
  - 例如，Alldiff约束表示所有变量必须取不同的值。
  - 如果变量m个，可能取值n个，且$m > n$，则约束不可能满足
  - 首先，移出一个单值变量，从其他变量的值域中删除该值。重复这个过程直到没有单值变量。如果出现了空的值域或变量多余值域大小，就检测到了不相容
- 例子
  - 检测$\{WA = red, NSW = red\}$的不相容
  - SA, NT和Q通过Alldiff约束连接
  - 应用AC-3算法，值域缩小为$\{green, blue\}$
  - 三个变量，两种颜色，违反了Alldiff约束，不相容



6.2 约束传播-全局约束

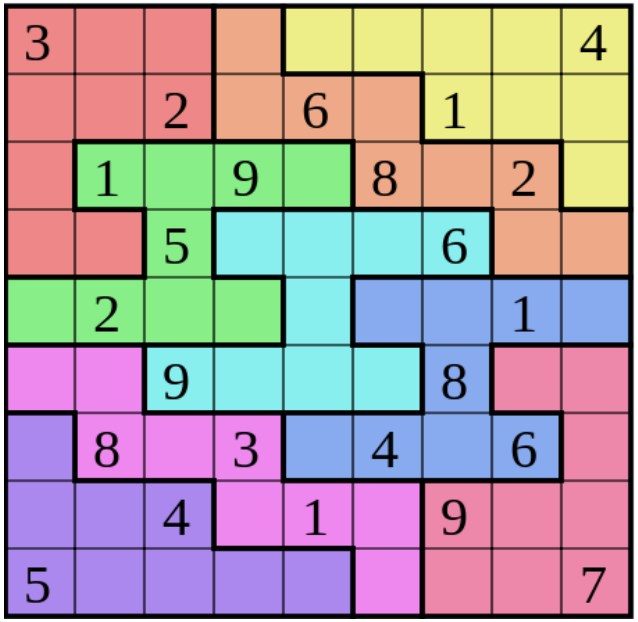
- 另一个重要的高阶约束是**资源约束**（atmost约束）
  - $P_1, \ldots, P_4$表示资源调度任务中执行4项任务的人数，若总人数不超过10个人，则表示为：$atmost(10, P_1, P_2, P_3, P_4)$
  - 检验当前值域最小值之和就能检测出矛盾，如果每个变量的值域都是$\{3,4,5,6\}$，atmost约束无法满足！
  - 删除变量值域中与其他变量的值域中的最小值不相容的最大值来保持相容性，如果变量值域都是$\{2,3,4,5,6\}$，那么5和6就可以从每个变量的值域中删去

6.2 约束传播-全局约束

- 大型整数值资源限制问题
  - 在航班调度系统中，假设有两次航班F_1和F_2，分别有165和385个座位，每次航班可承载的乘客数初始值域为$D_1 = [0, 165]$和$D_2 = [0, 385]$
  - 添加一个约束，两次航班总乘客数为420：$F_1 + F_2 = 420$
  - 通过传播边界约束，两个值域削减到：$D_1 = [35,165]$和$D_2 = [255,385]$
- CSP是边界相容：每个变量X和其取值上下界，每个变量Y都存在某个取值满足X和Y的约束

6.2 约束传播-数独游戏实例

- 数独的棋盘有81个方格，有些方格中预先填好了从1到9的数字。游戏要求所有方格都要填上数字，只允许一个数字在任一行、一列或3*3方框内出现一次。一行、一列或一方框称为一个单元
- 人类求解的话可能需要几十分钟，但用CSP求解器计算时间不会超过0.1秒



6.2 约束传播-数独游戏实例

- 规则：只允许一个数字在任一行、一列或3x3方框内出现一次
  - 81个变量的CSP，A1到A9表示第一行的9个变量（从左到右），到最后一行用I1到I9表示，空格变量值域$\{1,2,\ldots,9\}$，填好数字的方格值域就是自身
  - 27个Alldiff约束，每行、每列、每方框

	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		I		3		

(a)

	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

(b)

图 6-4 （a）一个数独问题。（b）它的解

$$Alldiff(A1, A2, A3, A4, A5, A6, A7, A8, A9)$$
$$Alldiff(B1, B2, B3, B4, B5, B6, B7, B8, B9)$$
$$\ldots$$
$$Alldiff(A1, B1, C1, D1, E1, F1, G1, H1, I1)$$
$$Alldiff(A2, B2, C2, D2, E2, F2, G2, H2, I2)$$
$$\ldots$$
$$Alldiff(A1, A2, A3, B1, B2, B3, C1, C2, C3)$$
$$Alldiff(A4, A5, A6, B4, B5, B6, C4, C5, C6)$$
$$\ldots$$

6.2 约束传播-数独游戏实例

- 假设Alldiff约束已全部展开为二元约束（如$A1 \neq A2$），可以直接使用AC-3算法
- E_6：方框约束：~~2，8，1，7~~，列约束：~~5，6，2，8，9，3~~，$E_6 = 4$
- I_6：列弧相容约束：~~5，6，2，4，8，9，3~~，I_5弧相容约束：~~1~~，$I_6 = 7$，$A_6 = 1$
- AC-3只能求解较简单的数独谜题。复杂的推理策略，如三链数删减法
- 数独求解所用的弧相容，路径相容等，通用于所有的CSP。对每个新问题领域只需定义问题约束，便可使用通用约束求解机制

	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		I		3		

（a）

	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

（b）

- 定义约束满足问题
- 约束传播：CSP中的推理
- CSP的回溯问题
- CSP局部搜索
- 问题的结构

6.3 回溯搜索-引入

- 数独游戏被设计成通过**约束间的推理**来求解。但很多CSP只用推理是不能求解的，这时必须通过**搜索**来求解，首先讨论部分赋值的回溯搜索算法

6.3 回溯搜索-引入

回溯搜索

- 标准深度优先搜索
  - CSP中有n个值域大小为d的变量，顶层分支因子nd，下一层分支因子$(n-1)d$，依次类推，搜索树含$n! \times d^n$个叶子，实际最多只有d^n个！
- 忽视CSP的至关紧要性质：**可交换性**
  - 可交换性：行动的先后顺序对结果没有影响。CSP是可交换的，不需要考虑赋值的顺序，只需考虑搜索树的一个结点的单个变量。
  - 澳大利亚地图着色问题，只需要在SA=red，SA=green和SA=blue之间选择，不需在SA=red和WA=blue之间选择。
  - 有了这个限制，叶结点的个数减少到了d^n个

6.3 回溯搜索-引入

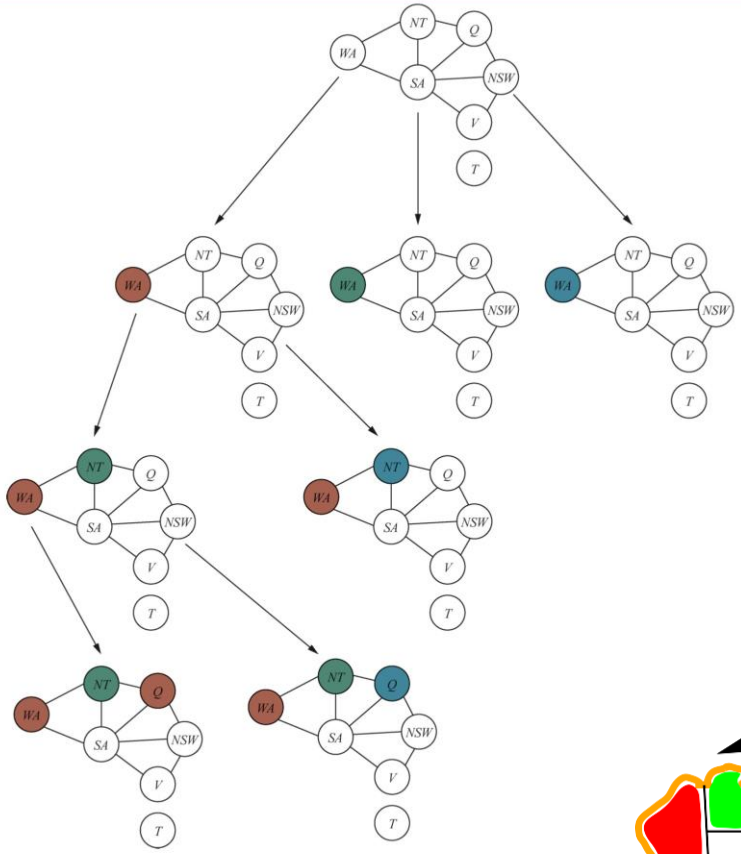
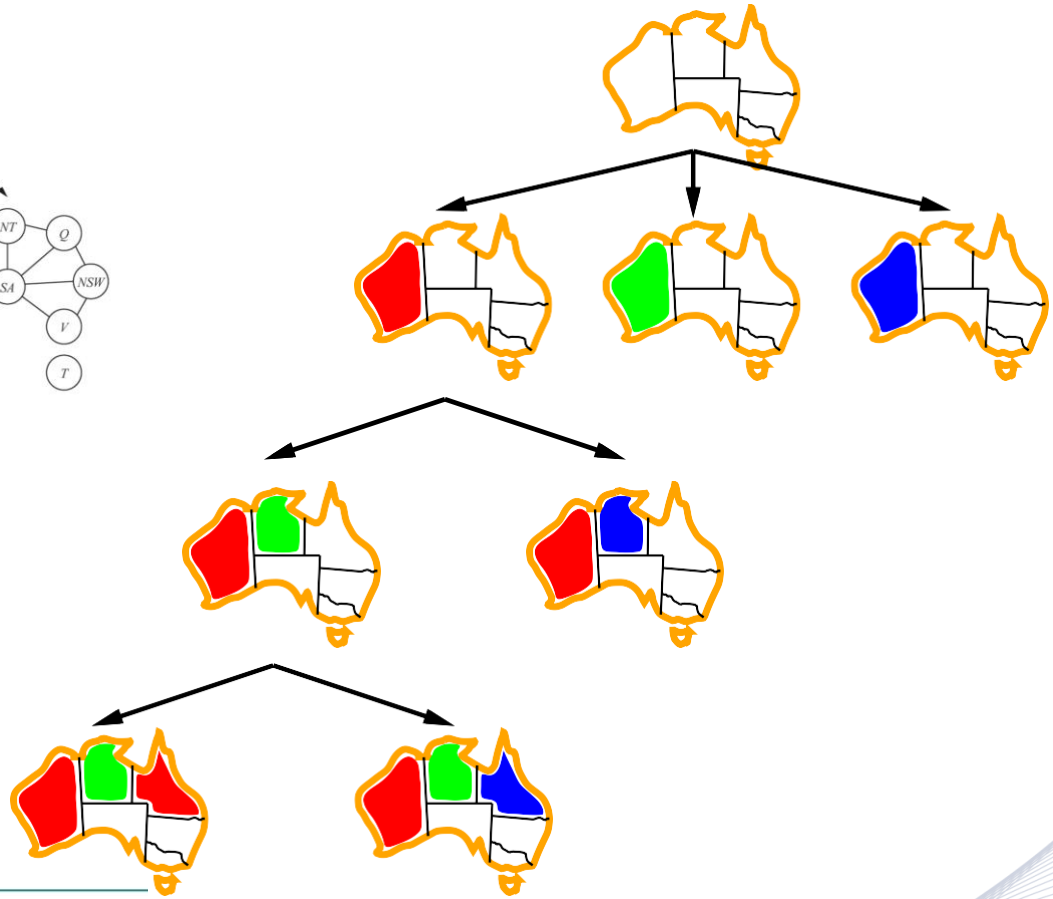


图 6-6　图 6-1 中地图着色问题的部分搜索树



6.3 回溯搜索-引入

```
function BACKTRACKING-SEARCH(csp) returns 一个解或failure
  return BACKTRACK(csp, {})

function BACKTRACK(csp, assignment) returns 一个解或failure
  if assignment是完整的 then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
  for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
    if value与assignment一致 then
      将{var = value}添加到assignment中
      inferences ← INFERENCE(csp, var, assignment)
      if inferences ≠ failure then
        将inferences添加到csp中
        result ← BACKTRACK(csp, assignment)
        if result ≠ failure then return result
        从csp中删除inferences
      从assignment中删除{var = value}
  return failure
```

图 6-5　约束满足问题的简单回溯算法。该算法以第 3 章的递归深度优先搜索为模型。函数 SELECT-UNASSIGNED-VARIABLE 和 ORDER-DOMAIN-VALUES 实现了 6.3.1 节中讨论的通用启发式算法。函数 INFERENCE 可以根据需要选择性地使用弧一致性、路径一致性或 k 一致性检测。如果一个赋值导致了失败（无论是在 INFERENCE 还是在 BACKTRACK 中），那么该赋值（包括从 INFERENCE 得到的值）将被撤销，然后重新尝试一个新的赋值

6.3 回溯搜索-引入

- 无需领域特定知识也能有效解决CSP，只需将上图中的函数复杂化
  - 下一步该给哪个变量赋值(SELECT-UNASSIGNED-VARIABLE)?，按什么顺序尝试它的值(ORDER-DOMAIN-VALUES)?
  - 每步搜索应做怎样的推理(INFERENCE)?
  - 搜索到达某赋值违反约束时，搜索本身能避免重复这样的失败吗?

6.3 回溯搜索-变量和取值顺序

- 变量和取值顺序
  - 回溯算法包括这样一行

$$var \leftarrow Select - Unassigned - Variable(csp, assignment)$$

  - 最简单策略是按照列表顺序$\{X_1, X_2, \ldots\}$选择未赋值变量，静态的变量排序**低效**!
  - 例如：在赋值WA=red和NT=green后，SA只能是blue，比给Q赋值更有意义

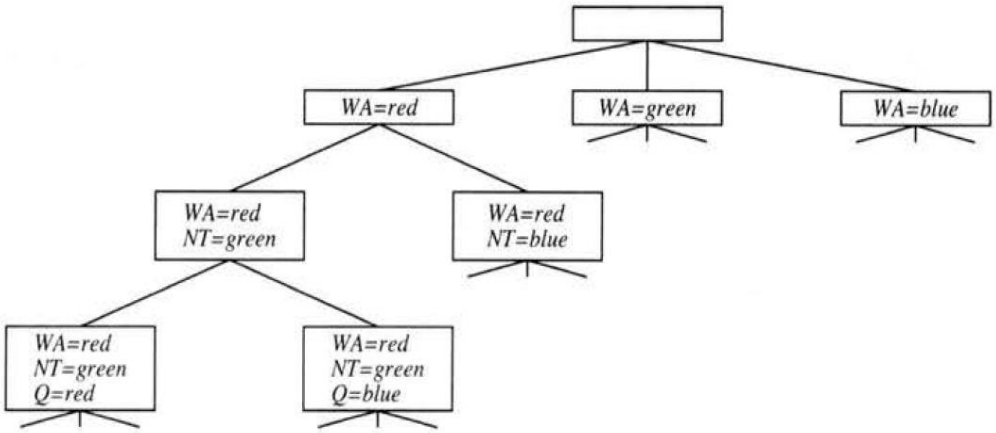


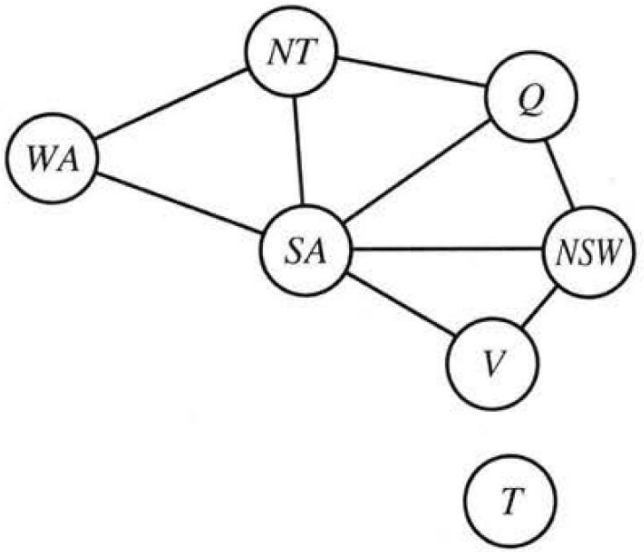
图 6.6　图 6.1 中地图着色问题的部分搜索树

6.3 回溯搜索-变量和取值顺序

- **最少剩余值（MRV）启发式**：

也称“最受约束变量”或“失败优先”启发式（最可能很快导致失败的变量，从而对搜索树剪枝）

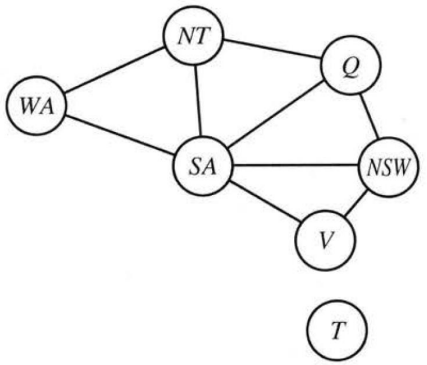
- 选择“合法”取值最少的变量
- 相比随机的或静态的排序，MRV启发式通常性能更好，有时要好到1000倍以上



6.3 回溯搜索-变量和取值顺序

度启发式

- 选择与其他未赋值变量约束最多的变量来试图降低未来的分支因子
- SA的度最高为5；其他变量的度为2或者3，T的度为0
- 一旦一个变量被选定，算法需要检验它的取值顺序，为此，有时**最少约束值**启发式很有效
  - 优先选择的值是给邻居变量留下更多的选择
  - 假设已经赋值WA=red和NT=green，下一步给Q赋值，若Q赋值蓝色，SA无合法值，所以Q更应赋值红色



6.3 回溯搜索-变量和取值顺序

- **为什么变量选择是失败优先，而值的选择是失败最后呢?**
  - 选择具有最少剩余值的变量通过早期的有效剪枝而有助于最小化搜索树中的结点数
  - 对于值的排序，**窍门是只需找到一个解**；首先选择最有可能的值是有意义的，如果需要枚举所有的解而不是只找一个解，值的排序毫无意义

6.3 回溯搜索-搜索与推理交替进行

搜索与推理交替进行

- AC-3和其他算法的推理考虑的是如何能在搜索前缩小变量的值域空间
- 搜索过程中的推理更加有力：每次决定给某变量某个值时，都有机会推理其邻接变量的值域空间
- 最简单的推理形式是**前向检验**

只要变量X被赋值，前向检验过程对它进行弧相容检查

即，对每个通过约束与X相关的未赋值变量Y，从Y的值域中删去与X不相容的那些值

6.3 回溯搜索-搜索与推理交替进行

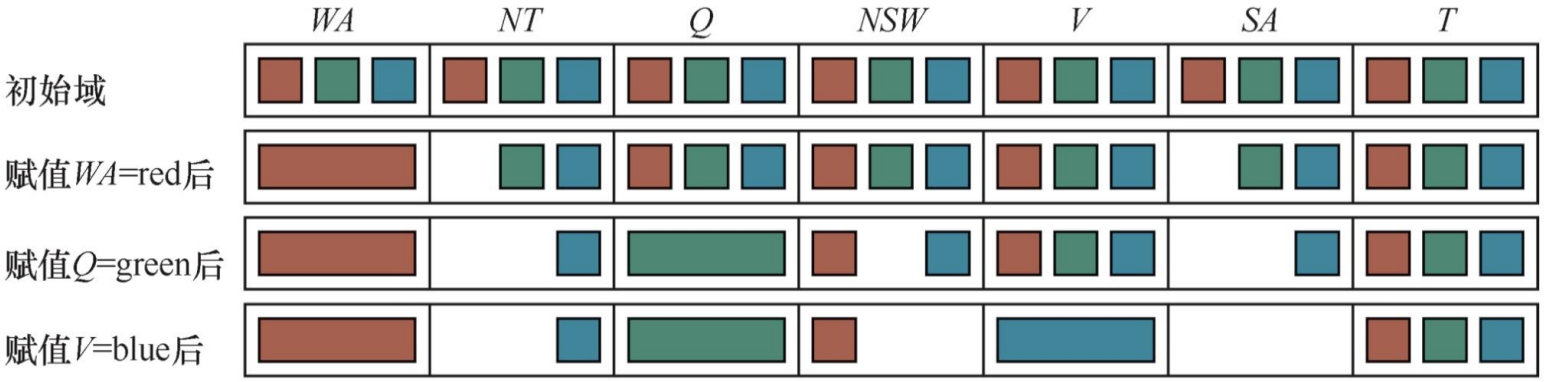
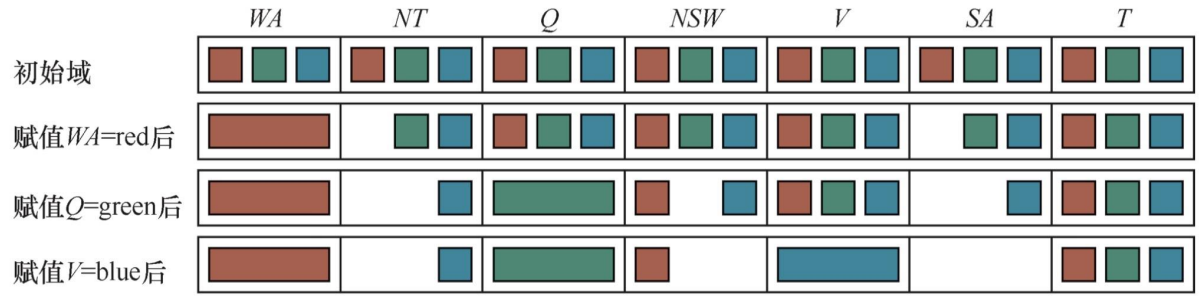


图 6-7　带前向检验的地图着色搜索过程。首先赋值 WA = red；然后前向检验从其相邻变量 NT 和 SA 的域中删除 red。赋值 Q = green 后，从 NT、SA 和 NSW 的域中删除 green。赋值 V = blue 后，从 NSW 和 SA 的域中删除 blue，此时 SA 没有合法值

- 在地图着色搜索中使用前向检验的回溯过程
  - 在赋值WA=red和Q=green之后，NT和SA的值域都缩小到了只有单个值；通过WA和Q的约束传播信息删除了这些变量的一些分支
  - 当赋值V=blue之后，SA的值域为空，与问题约束不相容，算法立刻回溯

6.3 回溯搜索-搜索与推理交替进行



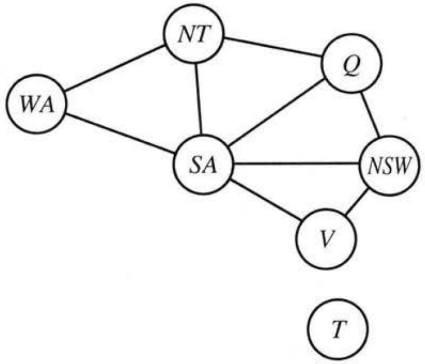
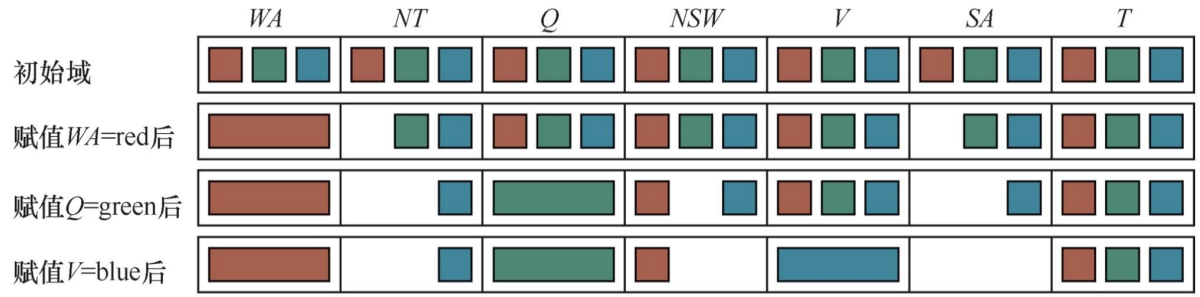


图 6-7　带前向检验的地图着色搜索过程。首先赋值 *WA* = red；然后前向检验从其相邻变量 *NT* 和 *SA* 的域中删除 red。赋值 *Q* = green 后，从 *NT*、*SA* 和 *NSW* 的域中删除 green。赋值 *V* = blue 后，从 *NSW* 和 *SA* 的域中删除 blue，此时 *SA* 没有合法值

- 联合使用MRV启发式和前向检验
  - 赋值{WA=red}后，直觉上应该处理邻接的变量NT和SA，然后才是其他变量
  - MRV正是这么做的：NT和SA都有两个值，先选一个，再选另一个，然后依次才是Q、NSW和V。最后确定T的值

6.3 回溯搜索-搜索与推理交替进行



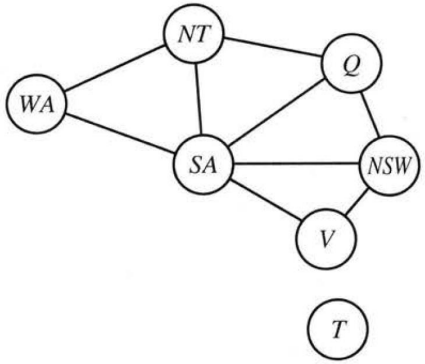


图 6-7　带前向检验的地图着色搜索过程。首先赋值 *WA* = red；然后前向检验从其相邻变量 *NT* 和 *SA* 的域中删除 red。赋值 *Q* = green 后，从 *NT*、*SA* 和 *NSW* 的域中删除 green。赋值 *V* = blue 后，从 *NSW* 和 *SA* 的域中删除 blue，此时 *SA* 没有合法值

- 尽管前向检验能够检测很多不相容，它无法检测出所有不相容
  - 它使当前变量弧相容，但并不向前看使其他变量弧相容，前向检验向前看的不够远
  - 如图第三行，给WA和Q赋值后，NT和SA只能是蓝色，但他们邻接不能取相同值

6.3 回溯搜索-搜索与推理交替进行

- **维护弧相容**（MAC）算法
  - 能检测出这类不相容性。当Xi赋值后，INFERENCE调用AC-3时，不考虑所有弧，而是从X_i邻接的弧$\{X_j, X_i\}$中所有未赋值变量X_j开始，进行约束传播，一旦某变量值域为空，AC-3调用失败并立即回溯
  - 前向检验在检查第一步与MAC相同，不同的是，前向检验在变量值域发生变化时并不递归传播约束

6.3 回溯搜索-智能回溯：向后看

- 约束满足问题的简单回溯算法，当一个分支搜索失败时采取简单的处理原则
  - 退回前一个变量并且尝试另外一个值，这称为时间回溯，重新访问的是时间最近的决策点
  - 考虑着色问题，按照固定变量顺序：Q, NSW, V, T, SA, WA, NT，应用简单回溯算法求解。假设部分赋值为{Q=red，NSW=green，V=blue，T=red}，处理SA时，任何值都违反约束，再退回到T。显然，对T重新着色不能解决该问题。

6.3 回溯搜索-智能回溯：向后看

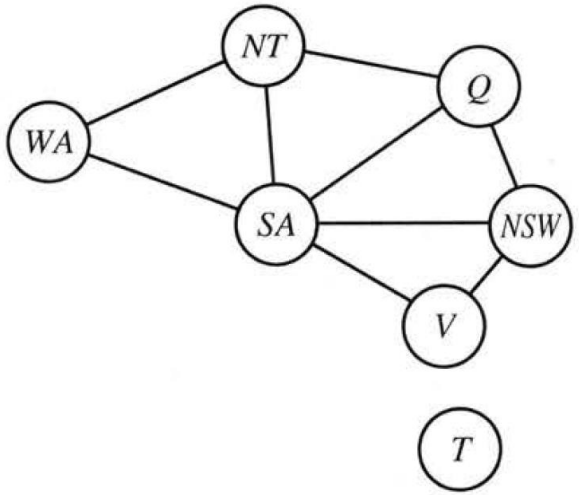
- 智能回溯：向后看
- 更智能的回溯方法时退回到可能解决这个问题的变量——导致SA赋值失败的变量集合中的变量
  - 跟踪与SA的某些赋值冲突的一组赋值，这个集合称为SA的**冲突集**，如{Q=red，NSW=green，V=blue}
  - **回跳**方法回溯到冲突集中时间最近的赋值，回跳将跳过T而尝试V的新值，通过简单修改BACKTRACK实现
  - 在检验合法值的时候保存冲突集，如果没有找到合法值，按照失败标记返回冲突集中时间最近的元素

6.3 回溯搜索-智能回溯：向后看

- 智能回溯：向后看
  - 前向检验算法可以不需要额外工作量就能提供冲突集：当基于赋值$X = x$的前向检验删除变量Y的值域中的值时，应该把$X = x$加入到Y的冲突集里
  - 回跳发生在值域中每个值都和当前赋值冲突的情况下，前向检验能检测到这个事件并且能阻止搜索到达这样的结点！可以证明回跳剪掉的每个分支在前向检验算法中也被剪枝，因此，简单的回跳在前向检验搜索中是多余的

6.3 回溯搜索-智能回溯：向后看

- 有冲突的回跳
  - $\{WA = red,\ NSW = red,\ T = red\}$，给NT、Q、V、SA赋值
  - 用完了NT的所有可能取值，但无解，实际上NT有和前面的赋值相容的值并没有导致失败的完全冲突集。所以，只可能是赋值的三个变量与未赋值的四个变量有冲突
  - 引出关于NT的冲突集的更深入的概念：前面的变量集合致使NT连同任何后继变量一起没有相容解



6.3 回溯搜索-智能回溯：向后看

- 新的冲突集如何计算
  - SA失败了，其冲突集是$\{WA, NT, Q\}$
  - 回跳到Q，Q将SA的冲突集吸收到自己的冲突集里，即$\{NT, NSW\}$，新的冲突集是$\{WA, NT, NSW\}$，即在给定赋值后，从Q向前无解
  - 回跳到NT，得到新冲突集$\{WA, NSW\}$
  - 回跳到NSW
- 令X_j是当前变量，再令$conf(X_j)$为其冲突集。如果X_j的每个可能取值都失败了，回跳到$conf(X_j)$中最近的变量X_i，并使用下列公式：

$$conf(Xi) \leftarrow conf(Xi) \cup conf(X_j) - \{Xi\}$$

6.3 回溯搜索-约束学习

约束学习

- 从冲突集中找出引起问题的最小变量集。这组变量及其相应值称为**无用赋值**
- 如：$\{WA = red,\ NT = green,\ Q = blue\}$，前向检验得知这个状态无用，SA没法合法赋值，可以剪掉这个分支，因为以后的搜索不会遇到这种组合。如果赋值是从V和T开始的，就得记录这组无用。

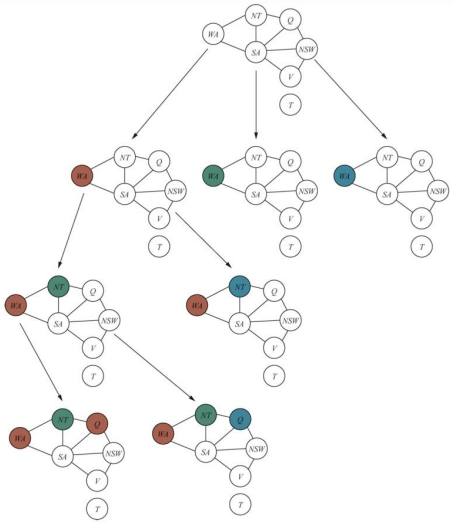


图 6-6　图 6-1 中地图着色问题的部分搜索树

- 定义约束满足问题
- 约束传播：CSP中的推理
- CSP的回溯问题
- CSP局部搜索
- 问题的结构

6.4 局部搜索

- **局部搜索算法**
  - **使用完整状态的形式化：初始状态给每个变量赋一个值，搜索过程是一次改变一个变量的值**
  - 例如，八皇后问题，初始状态是8个皇后在8列的一个随机布局，每步都是选择一个皇后并把它移动到该列的新位置上
  - 初始布局会违反一些约束，局部搜索的目的就是消除这些矛盾

6.4 局部搜索

- 为变量选择新值时，选择与其他变量冲突最少的值——**最少冲突启发式**

function MIN-CONFLICTS(*csp*, *max_steps*) **returns** 一个解或*failure*

  inputs: *csp*, 约束满足问题

    max_steps, 放弃前允许的步数

  current ←*csp*的一个初始完整赋值

  for *i* = 1 **to** *max_steps* **do**

    if *current*是*csp*的一个解 **then return** *current*

    var ←从*csp*.VARIABLES中随机选取的冲突变量

    value ←对于*var*使CONFLICTS(*csp*, *var*, *v*, *current*)取得最小值的*v*值

    在*current*中设置*var* = *value*

  return *failure*

图 6-9　CSP 的 MIN-CONFLICTS 局部搜索算法。初始状态可以随机选择，也可以通过基于贪心法的赋值过程依次为每个变量选择最少冲突值。在给定当前赋值的其余部分后，CONFLICTS 函数统计特定值违反约束的数量

6.4 局部搜索

- 局部搜索实例：八皇后问题

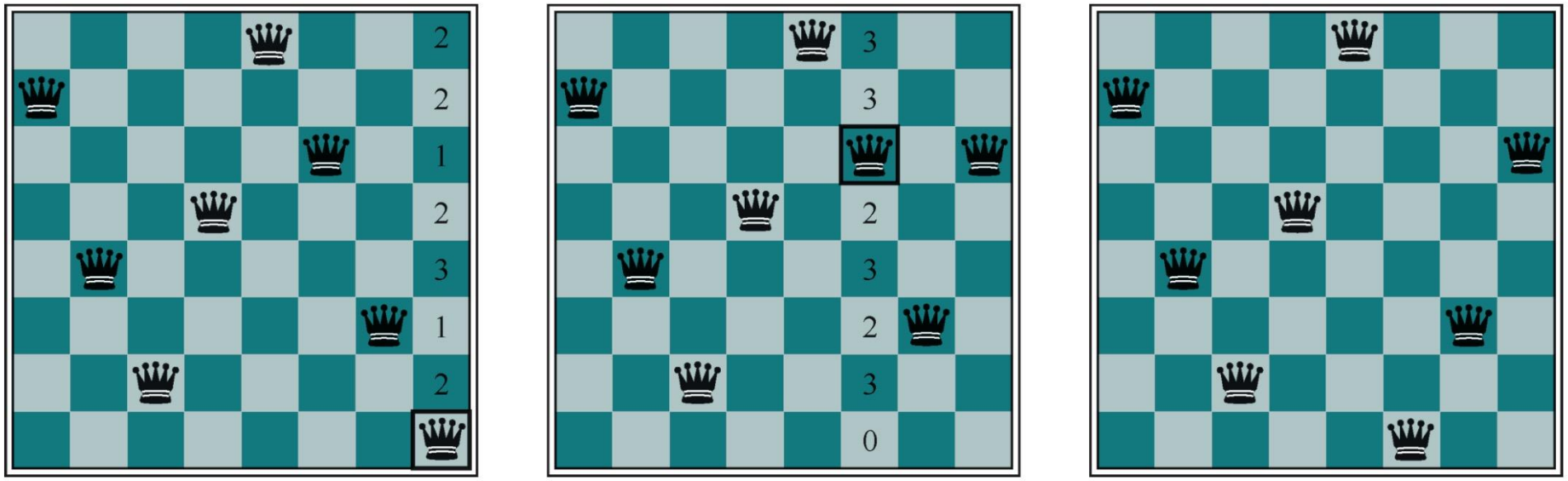


图 6-8　使用最少冲突法求解 8 皇后问题的示例。每步选择一个皇后，在其所在列重新分配位置。每个方格标有冲突数（在本例中是互相攻击的皇后个数）。算法随机选择发生冲突的皇后，将皇后移动到冲突最少的方格

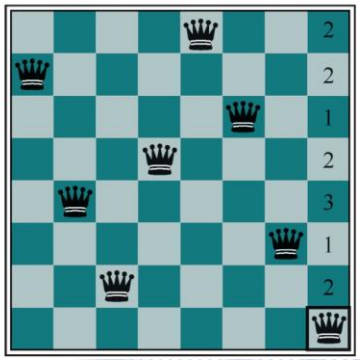
6.4 局部搜索

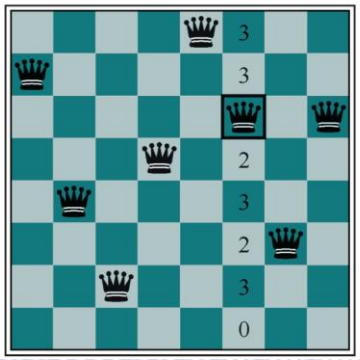
- 局部搜索实例：八皇后问题
  - 最少冲突对许多CSP都有效，比如能在平均（初始赋值后）50步之内求解数百万皇后问题
  - 使用最小冲突启发式的CSP地形通常有一系列高原

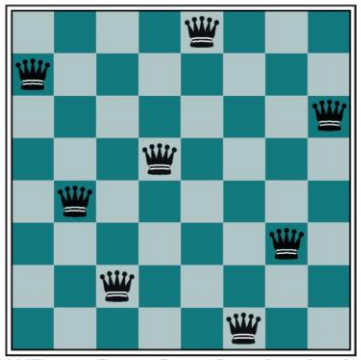
高原搜索：通过走小路移动到另一个分数相同的状态，可以帮助局部搜索离开高原。

这种高原上的流浪由禁忌搜索导引：将最近访问过的状态记录在表上，禁止算法再回到那些状态。

模拟退火也可以用于逃离高原







6.4 局部搜索

局部搜索算法

- **约束加权**：每个约束都有一个数字权重Wi，初始为1，搜索每一步选择使得违反约束权重和最小变量/值对来修改，增加当前赋值违反的约束的权重值
- 局部搜索适合应用在联机设置上。如在调度问题中，一周航班日程表涉及上千次航班和上万次赋值，但是恶劣天气可能会打乱原来的机场日常安排，可以从当前日程开始使用局部搜索算法，以最小改动修改日程

- 定义约束满足问题
- 约束传播：CSP中的推理
- CSP的回溯问题
- CSP局部搜索
- 问题的结构

6.5 问题的结构-引入

- **如何以结构化方式表示问题，如约束图所表示的，帮助快速找到解**
  - 将实际世界问题分解为很多子问题，即独立子问题，如T区域着色和对大陆着色
  - 独立性可以简单通过在约束图中寻找连通子图确定，每个连通子图对应一个子问题CSP_i，如果赋值S_i是CSP_i的一个解，那么U_iS_i是U_iCSP_i的一个解
  - 如果每个CSP_i包含所有n个变量中的c个变量，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为值域大小，因此，总工作量是$O(d^cn/c)$， 不分解的工作量$O(d^n)$

6.5 问题的结构-引入

- **如何以结构化方式表示问题，如约束图所表示的，帮助快速找到解**
  - 将$n = 80$的布尔CSP分解成4个$c = 20$的子问题，会使最坏情况下的时间复杂度从宇宙寿命减少到不到一秒
  - 有些图结构很容易求解，例如当约束图形成一棵**树**时，即任何两个变量间最多通过一条路径连通
  - 可以证明树结构CSP可以在变量个数的线性时间内求解
  - 求解数结构CSP时，首先任意选择一个变量为树的根，选择变量顺序，这样每个变量在树中出现在父结点之后，这样的排序称为**拓扑排序**

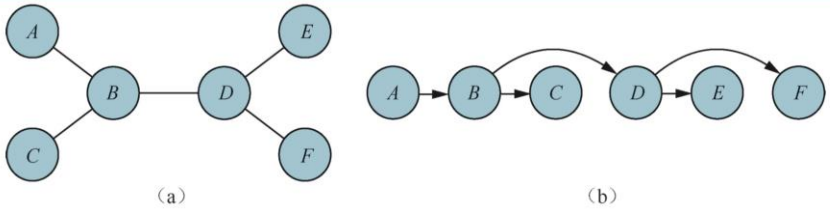


图 6-10 （a）树状结构 CSP 的约束图。（b）与以 A 为根节点的树一致的变量的线性排序。这称为变量的拓扑排序

6.5 问题的结构-引入

- **树结构CSP**
  - n个结点的树有$n-l$条弧，$O(n)$步内改成直接弧相容，每一步需要比较两个变量的可能取值，总时间是$O(nd^2)$
  - 父、子结点弧相容，父结点赋值时，子节点都有值可选，无需回溯，沿着变量线性前进

function TREE-CSP-SOLVER(*csp*) **returns** 一个解或*failure*
  inputs: *csp*，具有*X*, *D*, *C*的CSP

  $n \leftarrow X$中变量个数
  assignment ← 一个空的赋值
  root ← *X*中任一变量
  $X \leftarrow$ TOPOLOGICALSORT(*X*, *root*)
  for *j* = *n* **down to** 2 **do**
    MAKE-ARC-CONSISTENT(PARENT(X_j), X_j)
    if 不能取得一致 **then return** *failure*
  for *i* = 1 **to** *n* **do**
    assignment[X_i] ← D_i中任一一致的值
    if 没有一致值 **then return** *failure*
  return *assignment*

图 6-11　用于求解树状结构 CSP 的 TREE-CSP-SOLVER 算法。如果 CSP 有解，我们可以在线性时间内找到它；如果无解，将检测到矛盾

6.5 问题的结构-割集调整

- **有了求解树结构的高效算法，讨论更一般的约束图是否能化简成树的形式：**
  - 基于删除结点的-**割集调整**
  - 基于合并结点的-**树分解**

6.5 问题的结构-割集调整

基于删除结点的方法-割集调整

- 先对部分变量赋值，使剩下的变量能形成一棵树

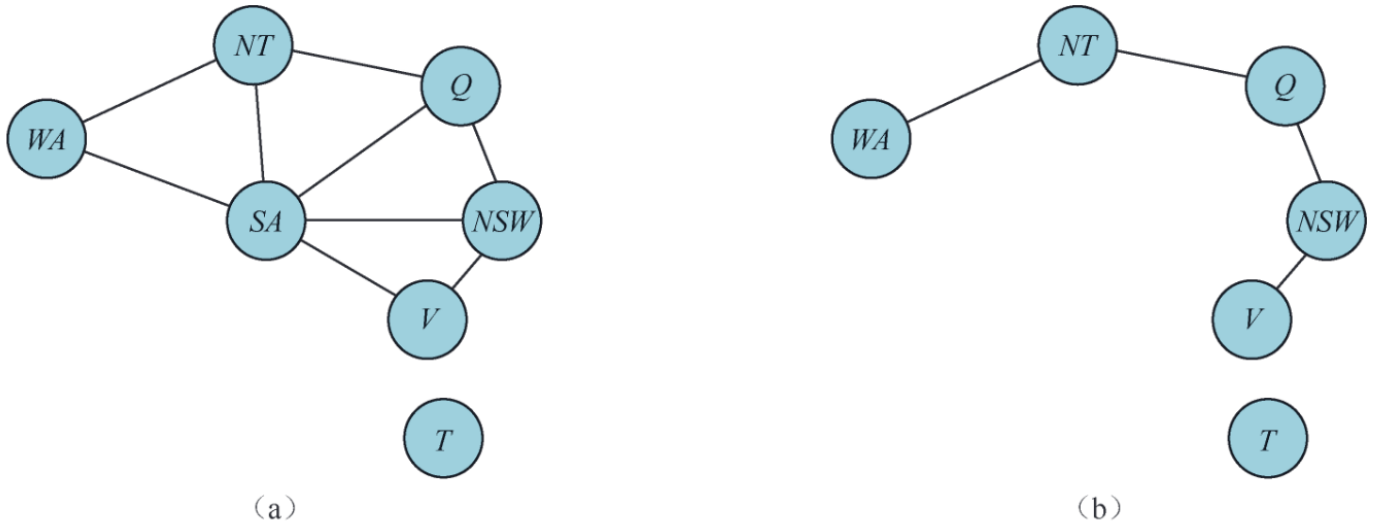


图 6-12 （a）图 6-1 中的原始约束图。（b）除去 *SA* 后，约束图变成由两棵树组成的森林

6.5 问题的结构-割集调整

基于删除结点的方法-割集调整

- 从CSP的变量中选择子集S，使得约束图在删除S后成为一棵树。S称为环割集
- 对于满足S所有约束的S中变量的每个可能赋值：
  - 从CSP剩余变量的值域中删除与S赋值不相容的值，并且
  - 如果去掉S后的剩余CSP有解，把解和S的赋值一起返回
- 时间复杂度
  - 如果环割集S大小为c，总运行时间$O(dc(n-c)d^2)$
  - 最坏情况c取$n-2$，找最小环割集是NP难，有近似算法。总体方法叫割集调整。

6.5 问题的结构-树分解

- **基于合并结点的方法-树分解**
- 以约束图的树分解为基础，把约束图分解为相关联的子问题，树分解必须满足以下3个条件：
  - 原始问题中的每个变量至少在一个子问题中出现
  - 如果两个变量在原问题中由约束相连，那么它们至少同时出现在一个子问题中（连同他们的约束）
  - 如果一个变量出现在树中的两个子问题中，那么它必须出现在连接这两个子问题的路径上的所有子问题里
- 前两个条件保证变量和约束都在分解中有表示，第三个条件反映了任何给定变量在每个子问题中必须取值相同的约束

6.5 问题的结构-树分解

基于合并结点的方法-树分解

- 把每个子问题视为一个“**巨型变量**”，值域是这个子问题的所有解的集合
- 例如，最左侧子问题是有三个变量的地图着色问题，因此有六个解，其中一个是$\{WA = red,\ SA = blue,\ NT = green\}$

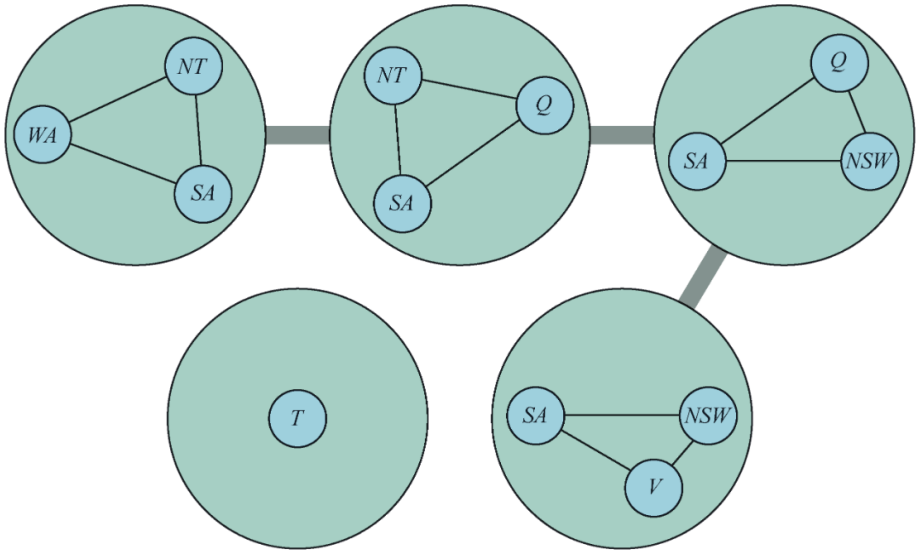


图 6-13　图 6-12a 中约束图的一个树分解

6.5 问题的结构-树分解

- **基于合并结点的方法-树分解**
  - 然后，可以用前面给出的树算法求解连接这些子问题的约束
  - 子问题之间的约束要求它们的共享变量要取相同的值
  - 例如，第一个子问题的解$\{WA = red,\ SA = blue,\ NT = green\}$，下一个子问题的相容解只能是$\{SA = blue,\ NT = green,\ Q = red\}$

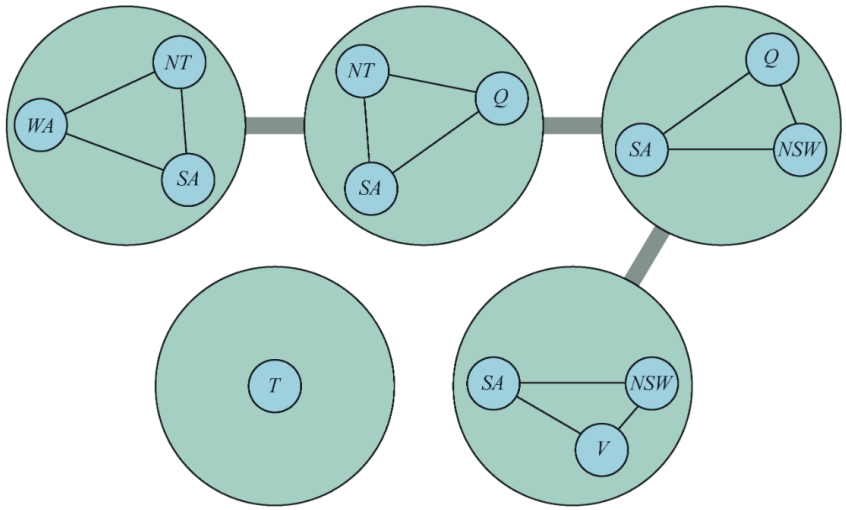


图 6-13　图 6-12a 中约束图的一个树分解

6.5 问题的结构-树分解

- **基于合并结点的方法-树分解**
  - 用树算法求解连接这些子问题的约束，子问题之间的约束要求它们共享变量要取相同的值
  - 例如，下一个子问题的相容解只能是$\{SA = blue,\ NT = green,\ Q = red\}$
  - 一个给定的约束图允许多种分解，原则是分解出来的子问题越小越好
  - 图的树分解的**树宽**是最大的子问题的大小减1；图本身的树宽定义为它**所有树分解的最小树宽**

6.5 问题的结构-树分解

- **基于合并结点的方法-树分解**
  - 图的树分解的树宽为w，给定对应的树分解，此问题的时间复杂度 $O\left(\mathrm{nd}^{w+1}\right)$
  - 如果CSP的约束图树宽有界，该CSP在多项式时间内可解
  - 不幸的是，找到最小树宽的树分解是一个NP难题，不过实际中有一些可行的启发式方法

6.5 问题的结构-值对称

- **值对称**
  - 澳大利亚着色问题：给WA，NT和SA着色，一共有3!=6种方法给变量赋值，这被称为**值对称**
  - 打破对称约束：打破值对称使搜索空间减小n!倍
  - 给变量一个特定的序：$NT < SA < WA$，取值满足字典序，确保n!个选择中只有一个是解：$\{NT = blue,\ SA = green,\ WA = red\}$
  - 多项式时间内找到约束来消除对称性并保留一个对称是可能的，但消除搜索过程中所有中间值集合的对称性是NP难题

6 本章小结

- 约束满足问题（CSP）的状态是变量/值对的集合，解条件通过一组变量上的约束表示。
- 很多推理技术使用约束来推导变量/值对是相容的

  结点相容、弧相容、路径相容、k-相容
- 回溯搜索，深度优先搜索的一种形式，经常用于求解CSP
- 最小剩余值和度启发式
- 求解CSP的复杂度与约束图的结构密切相关。树结构的问题可以在线性时间内求解。

课后作业

- 本章所描述的澳大利亚地图着色问题（三色）共有多少个解?
- 如何通过使用辅助变量把诸如A+B=C这样的三元约束变成三个二元约束。假设值域是有限的。
  - 提示：考虑引进新变量表示变量对，引进约束如“X是Y变量对中的第一个元素”

THANKS