约束传播、弧约束、问题结果与问题分解、局部搜索CSP

约束传播

- 前向检验将信息从已分配的变量传播到未分配的变量，但不能为所有失败情景提供早期检测：

NT和SA不能都是蓝色的！

- **约束传播**在局部重复强制执行约束

弧约束

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，则称此变量是**弧相容**的。更形式地，对于变量 X_i、X_j，若对 D_i 中的每个数值在 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。例如，考虑约束 $Y = X^2$，X 和 Y 都是数字。可以显式地写出约束为：

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

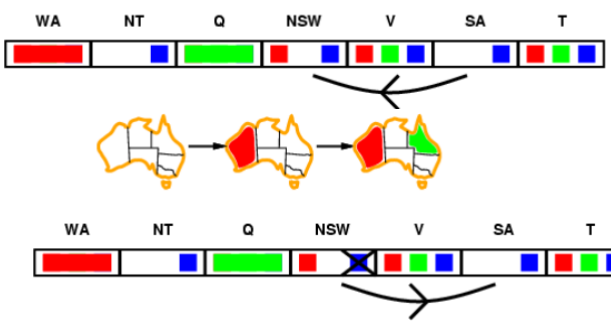
为了使 X 相对于 Y 是弧相容的，将 X 的值域缩小为{0, 1, 2, 3}。如果要使 Y 相对于 X 也是弧相容的，则 Y 的值域应缩小为{0, 1, 4, 9}，此时整个 CSP 就是弧相容的。

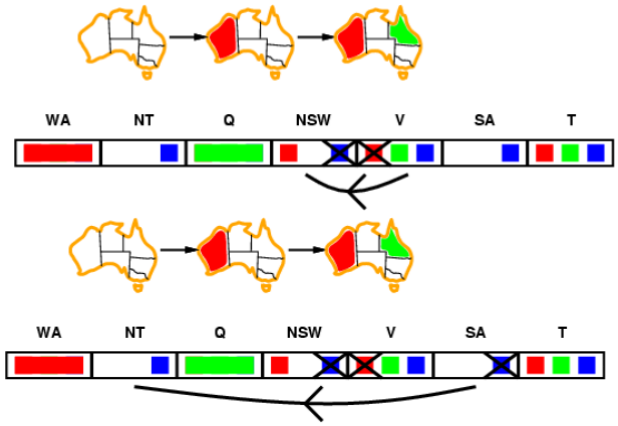
另一方面，弧相容可能对澳大利亚地图着色问题毫无帮助。考虑（SA, WA）之间的不同色约束：

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

不管你如何为 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。

- 能使每个弧相容的最简单形式：
  - **X→Y是可相容的**，当：
  - **对于X的每个值x，Y都存在不与之冲突的取值y**





- 如果X丢失了一个值，则需要重新检查X的邻居
- 弧相容比前向检验更早检测到可能失败的情景
- 弧相容可以作为预处理运行，也可以在每次分配后运行

弧相容算法AC-3

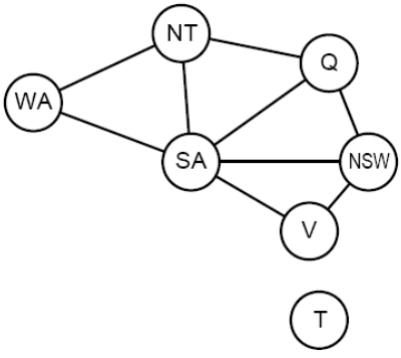
```
function AC-3( csp) returns the CSP, possibly with reduced domains
  inputs: csp, a binary CSP with variables {X_1, X_2, ..., X_n}
  local variables: queue, a queue of arcs, initially all the arcs in csp

  while queue is not empty do
    (X_i, X_j) ← 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 ← 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 ↔ X_j
      then delete x from DOMAIN[X_i]; removed ← true
  return removed
```

- 时间复杂度：$O(n^2d^3)$

  AC-3 的算法复杂度可以分析如下。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（弧）。每条弧(X_k, X_i)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，因此在最坏情况下算法的时间复杂度为 $O(cd^3)$。[1]

问题结构

- T和其它地区是独立的子问题，**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**



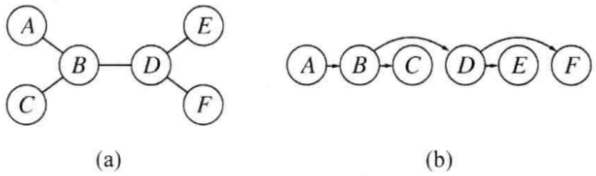
- 假设每个子问题有n个变量中的c个。最坏情况下的解决方案成本是$n/c \cdot d^c$，对n是线性的

每个 CSP_i 包含所有 n 个变量中的 c 个变量，这里 c 是一个常数。那么就会有 n/c 个子问题，解决每个子问题最多花费 d^c 步工作，这里 d 为值域大小。因此总的工作量是 $O(d^c n/c)$，是 n 的线性函数；如果不进行分解，总的工作量是 $O(d^n)$，是 n 的指数函数。看一个更具体的例子：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那么长减少到不到一秒。

- 例如，$n = 80,\ d = 2, c = 20$
  - $2^{80} = 40$亿年，1000万个节点/秒
  - $4 \cdot 2^{20} = 0.4$秒，1000万个节点/秒
- **任何一个树状结构的CSP问题可以在变量个数的线性时间内求解；**

完全独立的子问题是很诱人的，但是很少见。幸运的是，有些图结构也很容易求解。例如当约束图形成一棵**树**时，指的是任何两个变量间最多通过一条路径连通。我们将证明任何一个树状结构的 CSP 可以在变量个数的线性时间内求解。[1]这里的关键是称为直接弧相容或 DAC 的新概念。假设 CSP 的变量顺序为 $X_1, X_2, \cdots, X_n$，该 CSP 是直接弧相容的当且仅当对所有 $j > i$ 每个 X_i 与 X_j 都是弧相容的。

求解树结构 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）中为约束图，（b）为一种可能的排序。n 个结点的树有 $n-1$ 条弧，所以在 $O(n)$步内可以将此图改造成直接弧相容，每一步需要比较两个变量的 d 个可能取值，所以总时间是 $O(nd^2)$。一旦有了直接弧相容的图，就可以沿着变量列表并选择任意剩余值。由于父结点与其子结点的弧是相容的，我们知道无论父结点选择什么值，子结点都有值可选。这意味着无须回溯；可以沿着变量线性前进。



(a) (b)

图 6.10

（a）树状结构 CSP 的约束图。（b）与以 A 为根结点的树相容的变量的线性排序。这是众所周知的变量的**拓扑排序**。

如果约束图没有循环，CSP可以在$O(n \cdot d^2)$时间内解决。

与一般CSP相比，最坏情况下的时间是$O(d^n)$

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

  n ← number of variables in X
  assignment ← an empty assignment
  root ← any variable in X
  X ← 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] ← any consistent value from D_i
    if there is no consistent value then return failure
  return assignment
```

图 6.11 TREE-CSP-SOLVER 求解树结构 CSP。如果 CSP 有解，算法可以在线性时间内求解；如果无解，则会检测到矛盾

化简约束图-树结构

现在已经有了求解树结构的高效算法，下面讨论更一般的约束图是否能化简成树的形式。可以有两种基本方法：一种是基于删除结点的，一种是基于合并结点的。

第一种方法是先对部分变量赋值，使剩下的变量能够形成一棵树。考虑澳大利亚问题的约束图，如图 6.12（a）所示。如果能删除南澳大利亚州，这个图就会变成像（b）中的一棵树。幸运的是，可以这样做（只是在图中删除，而不是真的从大陆上删除），给变量 *SA* 一个固定的值并且从其他变量的值域中删除任何与 *SA* 的取值不相容的值。

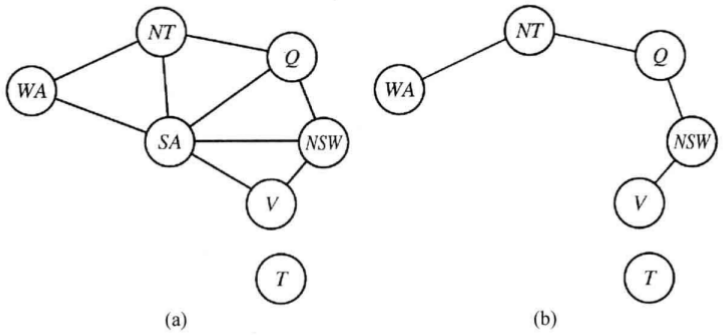


图 6.12
（a）图 6.1 的原始约束图；（b）删除 *SA* 之后的约束图

现在，在删除了 *SA* 和它的约束之后，CSP 的每个解都将与 *SA* 的值相容。（这对二元 CSP 是可行的；在高阶约束问题中情况会更复杂。）因此，用上面给出的算法求解剩余的树，并由此得到整个问题的解。当然，在一般情况下（与地图着色不同），为 *SA* 选择的值可能是错误的，因此将需要尝试所有的可能值。一般算法如下：

（1）从 CSP 的变量中选择子集 *S*，使得约束图在删除 *S* 之后成为一棵树。*S* 称为**环割集**（cycle cutset）。

（2）对于满足 *S* 所有约束的 *S* 中变量的每个可能赋值：

（a）从 CSP 剩余变量的值域中删除与 *S* 的赋值不相容的值，并且

（b）如果去掉 *S* 后的剩余 CSP 有解，把解和 *S* 的赋值一起返回。

如果环割集的大小为 c，那么总的运行时间为 $O(d^c \cdot (n-c)\, d^2)$：我们需要尝试 *S* 中变量的赋值组合共 d^c 种，对其中的每个组合需要求解规模为 $n-c$ 的树问题。如果约束图“近似于一棵树”，那么 c 将会很小，直接回溯将节省巨大开销。然而在最坏情况下，c 可能大到$(n-2)$。虽然找出最小的环割集是 NP 难题，但是已经有一些高效的近似算法。算法的总体方法叫做**割集调整**；将在第 14 章再次讨论用它来进行概率推理。

CSP问题的局部搜索

CSP的迭代算法

- 爬坡法、模拟退火法通常对 "完整 "状态进行工作，即所有变量均被分配

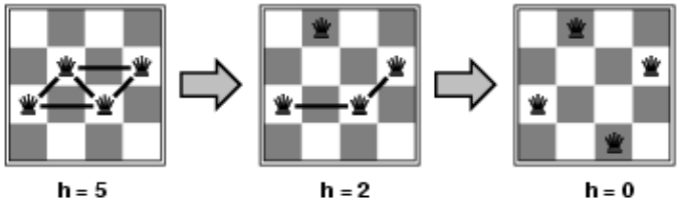
  局部搜索算法（见 4.1 节）对求解许多 CSP 都是很有效的。它们使用完整状态的形式化：初始状态是给每个变量都赋一个值，搜索过程是一次改变一个变量的取值。例如，在八皇后问题（见图 4.3）中，初始状态是 8 个皇后在 8 列上的一个随机布局，然后每步都是选择一个皇后并把它移动到该列的新位置上。典型地，初始布局会违反一些约束。局部搜索的目的就是要消除这些矛盾。[1]

  在为变量选择新值的时候，最明显的启发式是选择与其他变量冲突最少的值——**最少冲突**启发式。

- 为了适用于CSP：
  - **允许有未满足的约束条件的状态**
  - 操作者重新分配变量值
- 变量选择：随机选择任何有冲突的变量
- **通过最小冲突进行价值选择 启发式**：
  - 选择会造成与其它变量的冲突最小的值
  - 在爬山法中，h(n)=被违反的约束总数

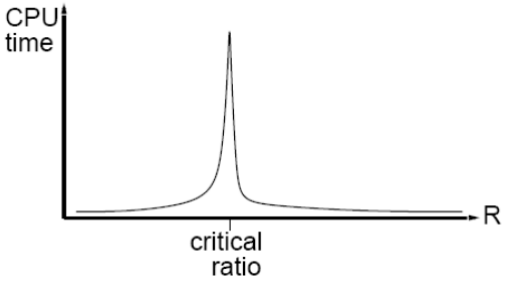
举例：4-Queens

- 状态： 4个皇后在4列（4^4=256个状态）
- 行动：在列中移动皇后
- 目标测试：没有冲突
- 评价：h（n）=冲突次数



- 最小冲突启发式的性能
  - **给定随机初始状态，可以在几乎恒定的时间内解决任意n的高概率问题**（如n=10,000,000）的n-queens。
  - 对于任何随机生成的CSP来说，除了在一个狭窄的比率范围内，似乎也是如此。

$$R = \frac{\text{number of constraints}}{\text{number of variables}}$$



- 在3-SAT问题中也能取得很好的性能：

# vars	Backtrack+tricks	Min-conflicts
50	1.5s	0.5s
100	3m	10s
150	10h	25s
200		2m
250		3m
300		13m
350		20m

加速：模拟退火法

- 思路：通过允许一些 "坏 "的动作来逃避局部最大值，但要逐渐减少其频率

  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

加速：最小最大优化(约束加权法)

另一项技术称为约束加权，可以帮助搜索把精力集中在重要约束上。每个约束都有一个数字权重 W_i，初始都为 1。搜索的每一步，算法都选择使得违反约束权重和最小的变量/值对来修改。接着增加当前赋值违反的约束的权重值。这有两个好处：给高原增加了地形，确保能够从当前状态改善，并且它随着时间的进行不断给难于解决的约束增加权重。

- 局部搜索的另一个优势是当问题改变时它可以用于联机设置。这在调度问题中特别重要。一周的航班日程表可能涉及上千次航班和上万次的赋值，但是一个恶劣天气可能就会打乱原来的机场日程安排。我们希望以在最小的改动来修正日程。从当前日程开始使用局部搜索算法，这项工作很容易地完成。使用新约束集的回溯搜索通常需要更多的时间，找到的解也有可能要对当前日程进行很多改动。

# vars	Backtrack+tricks	Min-conflicts	Minmax
50	1.5s	0.5s	0.001s
100	3m	10s	0.01s
150	10h	25s	0.1s
200		2m	0.25s
250		3m	0.4s
300		13m	1s
350		20m	2.5s

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。许多重要的现实世界问题都可以描述为 CSP。
- 很多推理技术使用约束来推导变量/值对是相容的，哪些不是。这些可以是结点相容、弧相容、路径相容和 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。推理和搜索可以交织进行。
- **最少剩余值**和**度**启发式是在回溯搜索中决定下一步选择哪个变量的方法，都独立于领域。**最少约束值**启发式帮助变量选取适当的值。回溯发生在变量找不到合法取值的时候。**冲突引导的回跳**直接回溯到导致问题的根源。
- 使用**最小冲突**启发式的局部搜索在求解约束满足问题方面也取得了成功。
- 求解 CSP 的复杂度在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。**割集调整**可以把一般 CSP 化简为树结构，如果能找到比较小的割集，算法十分有效。**树分解**技术把 CSP 转变为子问题构成的树，如果约束图的**树宽**不大，则算法很有效。