对每个状态使用因子化表示（factored representation）来打破黑盒：因子化表示为一组变量，每个变量都有自己的值。当每个变量的值都满足对该变量的所有约束时，问题就解决了。以上述方式描述的问题称为约束满足问题

**6.1 定义CSP（constraint satisfaction problem）**

CSP问题的组件X、D、C：

X是变量集合，{X1,X2,……,Xn}

D是域集合，{D1,D2,……,Dn}，即变量的值域的集合。

域Di，由变量Xi的一组允许的值{v1,……,vk}组成。

C是约束集合，用来规定允许的值组合，每个约束Cj=<scope,rel>,scope是该约束中的变 量元组，rel定义了这些值应该满足的关系。

CSP处理变量赋值问题，是NP-Complete问题

一致/合法赋值：不违反任何约束

完整赋值、部分赋值

部分解：一致部分赋值

CSP可视化为约束图：

**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**

CSP的优势：

1. 问题形式化简便

2．方法成熟

3．相比于**原子的**（状态是一个个小黑盒）状态空间搜索器，CSP求解器可以（通过约束条件）快速消除大面积搜索空间

**CSP形式体系的变体**

**按变量类型**

离散变量

有限域：O(d^n) 地图着色问题、带有时间限制的调度

无限域（整数、字符串、无时间限制的调度）：需要约束语言（算术约束和析取约束）

**线性约束问题可解且有通用算法**，非线性约束问题可解性无法判定

连续变量

线性规划（linear programming）

约束必须为线性等式或不等式

问题可以在关于变量个数的多项式时间内求解。

**按约束类型**

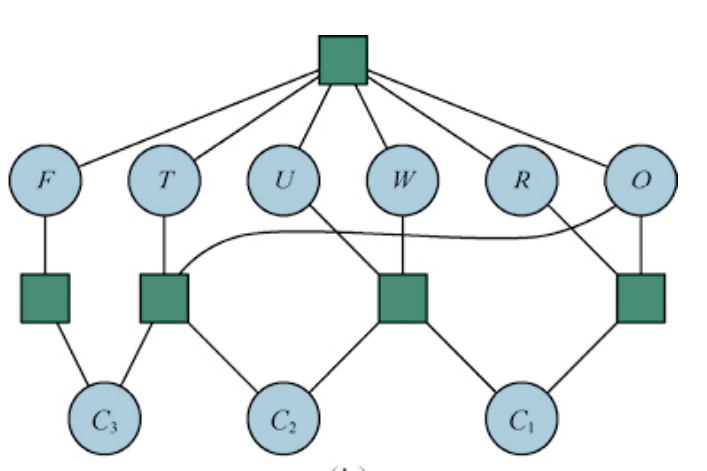
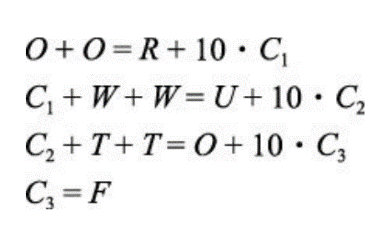
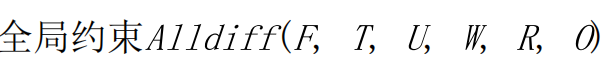
一元约束（unary constraint），它限制单个变量的值

二元约束（binary constraint）关系到两个变量

全局约束：包含任意个数变量的约束（如Alldiff，它表示约束中涉及的所有变

量必须具有不同的值）

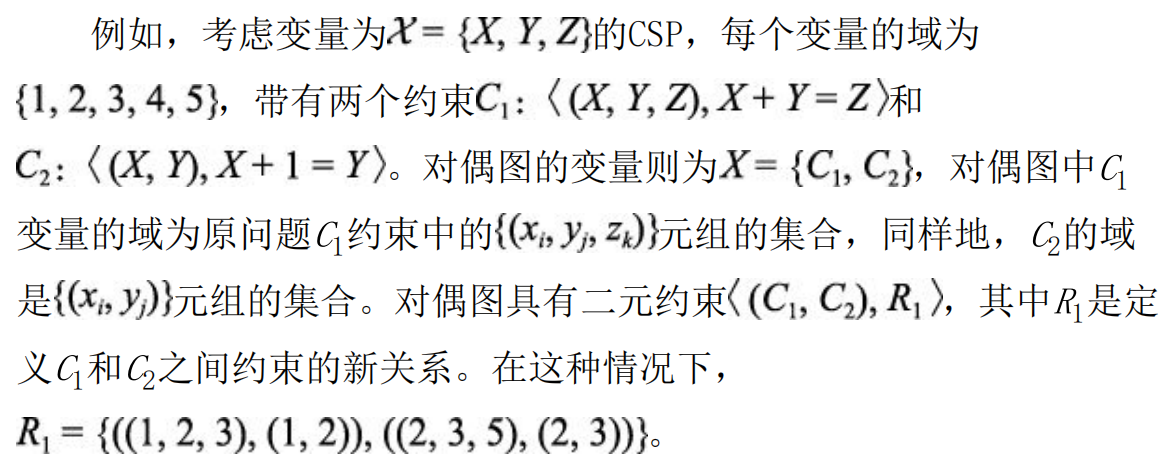
约束超图（constraint hypergraph）：表示变量和过渡变量的普通节点（圆圈）和表示n元约束的超节点（正方形）组成



二元CSP（binary CSP）只存在一元约束和二元约束，可用约束图表示

如果**引入足够多的辅助变量**，每个有限域约束都可以简化为一组二元约束。这意味着我们**可以将任意一个CSP转换为只有二元约束的CSP**

**对偶图（dual graph）变换**：创建一个新图，原图中的每个约束用新图中的一个变量表示，原图中的每对共享变量的约束用新图中的一个二元约束表示。



**偏好约束**

绝对约束规定哪些解是合法的（寻找可行解）

偏好约束规定哪些解是首选的（寻找最优解）

偏好约束通常可以编码为个别变量赋值的代价

带偏好约束的CSP称为**约束优化问题**（constrained optimization problem，COP）

**6.2约束传播：CSP中的推断**

使用约束减少一个变量的合法值的数量，这反过来又可以减少另一个变量的合法值

可以与搜索交通进行，也可当作搜索前的预处理操作

核心思想：**局部一致性**

**6.2.1节点一致性**

单个变量的**域中的所有值**都**满足**该变量的**一元约束**

（利用一元约束删除值域中的冗余部分）

**6.2.2弧一致性**

某一变量的域内的所有值都满足**该变量**的**二元约束**

**增强弧一致性的算法：AC-3算法**

维护一个弧队列：

初始时，队列中包含CSP中的所有弧（每个二元约束都有两条弧，每个方向各一条）

#从队列中任意弹出一条弧（Xi，Xj）并使Xi相对于Xj弧一致（修正Xi的域Di）

如果Di保持不变，继续处理下一条弧

但是如果Di得以修正（域变小），那么将所有的弧(Xk, Xi)（**Xi的所有邻居**）添加到队 列中

如果**Di变为空集**，那么表示整个CSP不存在一致解，**返回失败**

否则回到#，**直到队列中没有弧**

**可以得到一个与原始CSP等价的CSP**（它们的解相同，后者域更小）

弧一致CSP搜索起来会更快，可以完全求解问题（通过将每个域的大小缩减为1），或可以证明解不存在（通过将某些域的大小缩减为0）

假设设CSP有n个变量，每个变量的域大小不超过d，带有c个二元约束（弧）

每个弧(Xk, Xi)最多只能插入队列d次，因为Xi最多有d个值要删除

时间复杂度：O(cd3) 其中对每条弧一致性的检查O(d2)，检查弧的次数最多为cd

**6.2.3路径一致性**

{Xi，Xj}相对于Xm是路径一致的（Xi-->Xm🡪Xj）

对于每个满足Xi与Xj之间约束的的赋值{(xi,xj)}，都存在Xm的一个赋值满足{Xi，Xm}和{Xj，Xm}上的约束

**6.2.4 k一致性**

CSP是k一致的：

对于CSP的**任意(k −1)个变量**的集合以及这些变量的任意一致赋值，**任意第k个变量**都存在一个一致赋值

强k一致的：

一个CSP是k一致的，也是(k −1)一致的，(k −2)一致的……一直到1一致的

n一致性（n为节点数）算法的时间和空间复杂度都是指数级的

常用的是计算2一致性，其次是计算3一致性

**6.2.5全局约束**

全局约束涉及任意个数的变量（**但不一定是所有变量**）

Alldiff约束：

首先，**删除**约束中任意一个**单值变量**（域中只有一个值的变量），并且从**其余变量的域**中**删除该变量的值**。

只要还存在单值变量，就重复上述过程。

如果在任一点上**产生了空集**，或者**存在比剩余取值数更多的变量**，则检测到了**不一致性（误解）**

Atmost约束（资源约束）：

Atmost（10，P1，P2，P3，P4）相当于P1+P2+P3+P4<=10

通过检验当前域的**最小值之和**可以检测不一致性

如果当前**某个变量**的域中的**最大值**加上所有**其他变量**的域的**最小值**超过约束，则可以通过**删除该最大值**来保持一致性

对于大规模的、具有整数值的资源有限问题：

采用**边界传播**处理

**边界一致的：**如果对于**任意变量X**和它的**上下界值**，**任意变量Y**，都存在满足**X和**

**Y之间**约束的**Y的值**（DY中存在y1、y2使得(α,y1)、(β,y2)满足约束）

**6.3 CSP的回溯搜索**

搜索约束传播后仍存在的多种取值组合

状态树分支因子：n个变量、域大小为d

第一层：nd （选择一个变量给他赋上d个值中的一个）

第二层：（n-1）d

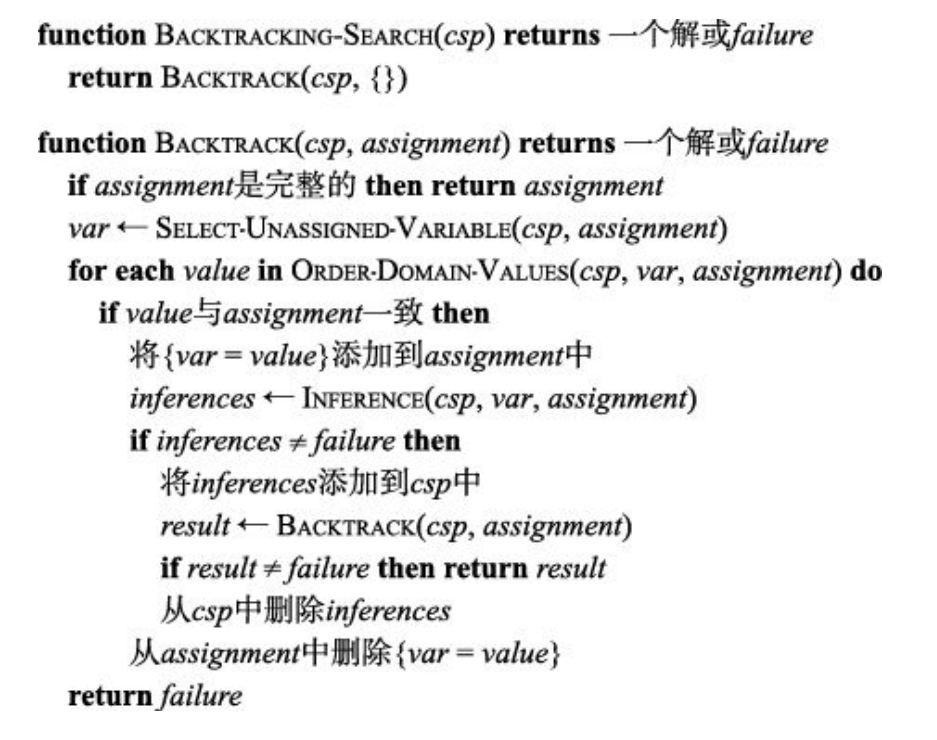
树总共有n! · d^n个叶节点

可交换性（commutativity）：可以消去因子n!，如果任意给定的**动作集合**的**应用顺序**对结果没有影响（如不 ）区别）

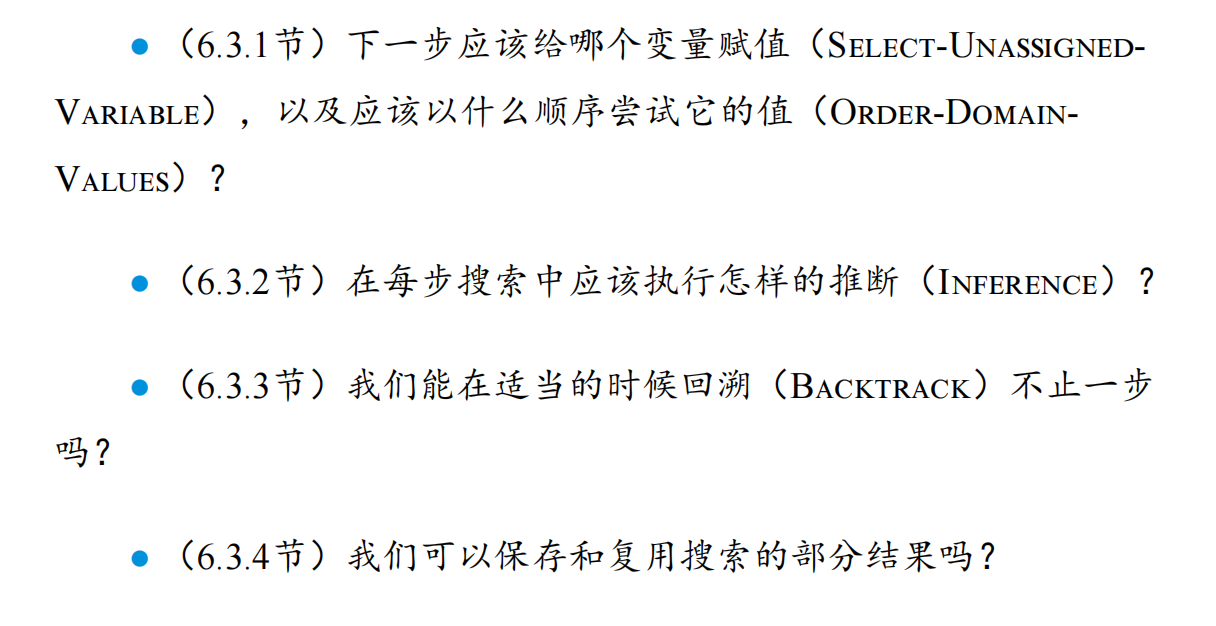
因此，我们只需考虑搜索树中每个节点上的**单个变量**

它不断选择未赋值变量，依次尝试该变量域中的所有值，试图通过递归调用将每个值扩展为一个解。

如果调用成功，则返回解，如果调用失败，则将赋值恢复到前一状态，然后尝试下一个值。如果所有值都不成功，则返回失败。



提高速度的方法：



**6.3.1 变量排序和值排序**

**变量排序：**

**（1）最少剩余值**或 “最受约束变量”或“失败优先”启发式算法：

选择“合法”值最少的变量（最有可能马上导致失败的变量/约束最多的变量），从而可以对搜索树剪枝

**（2）度启发式**算法：

通过选择**与其他未赋值变量**的**约束最多**的变量来降低未来选择的分支因子

**值排序：**

**最少约束值**（least-constraining-value）启发式算法

优先选择那些为约束图中**相邻变量留下最多选择**的值

总结：变量选择是失败优先，值选择是失败延后。如果我们的目标是枚举所有的解而不只是找到一个解，那么值排序就无关紧要了。

**6.3.2 交替进行搜索和推理**

推断：每次我们为某个变量选择某个值时，都有一个全新的机会推断其相邻变量的新的域缩减

前向检查：建立**弧一致性**：对于**每个**通过约束**与X连接**的**未赋值变量Y**，从它的域中删除与X的取值不一致的值

维护弧一致性（maintaining arc consistency，MAC）算法：

当变量Xi被赋值后，Inference程序调用AC-3，

但开始时弧队列只有与Xi相邻的未赋值变量Xj的弧(Xj, Xi)，而不是

CSP中的所有弧

**6.3.3智能回溯：向后看**

时序回溯：当搜索的一个分支失败时，退回到上一个变量，并为其尝试一个不同的值

回跳方法（backjumping）：回溯到冲突集中最近的赋值

冲突集：尝试给X赋值时，发现所有值都违背了约束，记录与X的某些值冲突的赋值集合

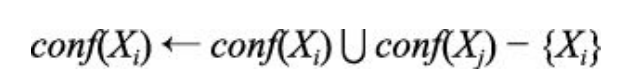
当前向检验根据赋值X = x从Y的域中删除一个值时，它应该将X = x添加到Y的冲突集中。如果Y的域中的最后一个值也被删除，那么Y的冲突集中的赋值也要被添加到X的冲突集中。也就是，我们现在知道X = x导致了（Y中的）矛盾，因此应该为X尝试不同赋值

可以证明，每个被回跳剪除的分支也会被前向检验剪枝

因此，在前向检验搜索中，或者在使用更强一致性检验的搜索（如MAC）中，简单的回跳是多余的——你只需执行其中一项。

**冲突导向回跳**（conflict-directed backjumping）：

设Xj表示当前变量，conf(Xj)表示它的冲突集。如果Xj的每个可能值都失败了，则回跳到conf(Xj)中最近的变量Xi，并使用下列公式重新计算Xi的冲突集：



**6.3.4约束学习**

当搜索得出一个矛盾时，我们知道这是冲突集的某个子集引起的。约束学习（constraint learning）的思想是从冲突集中找出**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（no-good）。如果想要记录无用赋值，要么通过向CSP中添加一个新的约束禁止这种赋值组合，要么通过维护一个单独的缓存

**6.4 CSP的局部搜索**

局部搜索算法：使用完整赋值状态形式，搜索一次改变一个变量的值。

最少冲突（min conflict）启发式算法：改变值时选择与其他变量冲突数最少的值

局部搜索可以很容易地扩展到约束优化问题（COP）。在这种情况下，爬山法和模拟退火的所有技术都可以用于优化目标函数。

在n皇后问题上，如果不计入皇后的初始布局，最少冲突法的运行时间**基本上与问题规模无关**。它甚至可以在（初始赋值后）平均50步内求解百万皇后问题。

粗略地说，用局部搜索求解n皇后问题非常简单，因为**解密集地分布在整个状态空间上**

最少冲突启发式算法下的CSP地形图通常存在一系列平台区：

可用4.1中的局部搜索算法

可以添加一种叫作**禁忌搜索**的技术导引：维护一个**最近访问**过的状态的列表，**并禁止算法返回那些状态**。

另一种技术称为**约束加权**：旨在集中搜索重要约束

每个约束都有一个数值权重，初始时都为1。在每步搜索中，算法找出使其所违反的约束的总权重**最低**的变量，并**修改其值（“最低“和”修改“都是为了找出最难满足的约束）**。然后，增加**当前赋值所违反的每个约束的权重**。

**6.5 问题的结构**

**——利用由约束图表示的问题的结构来快速找到解**

分解为独立子问题：

每个**连通分量**对应一个子问题CSPi。如果赋值Si是CSPi 的解，那么Si的并就是CSP解。

假设每个CSPi具有所有n个变量中的c个变量，其中c是一个常数。那么共有n/c个子问题，求解每个子问题最多需要dc工作量，其中d是域的大小。因此，总的工作量为O(dcn/c)

**关于n是线性的**

约束图为树（当任意两个变量都只由一条路径连接）的问题：

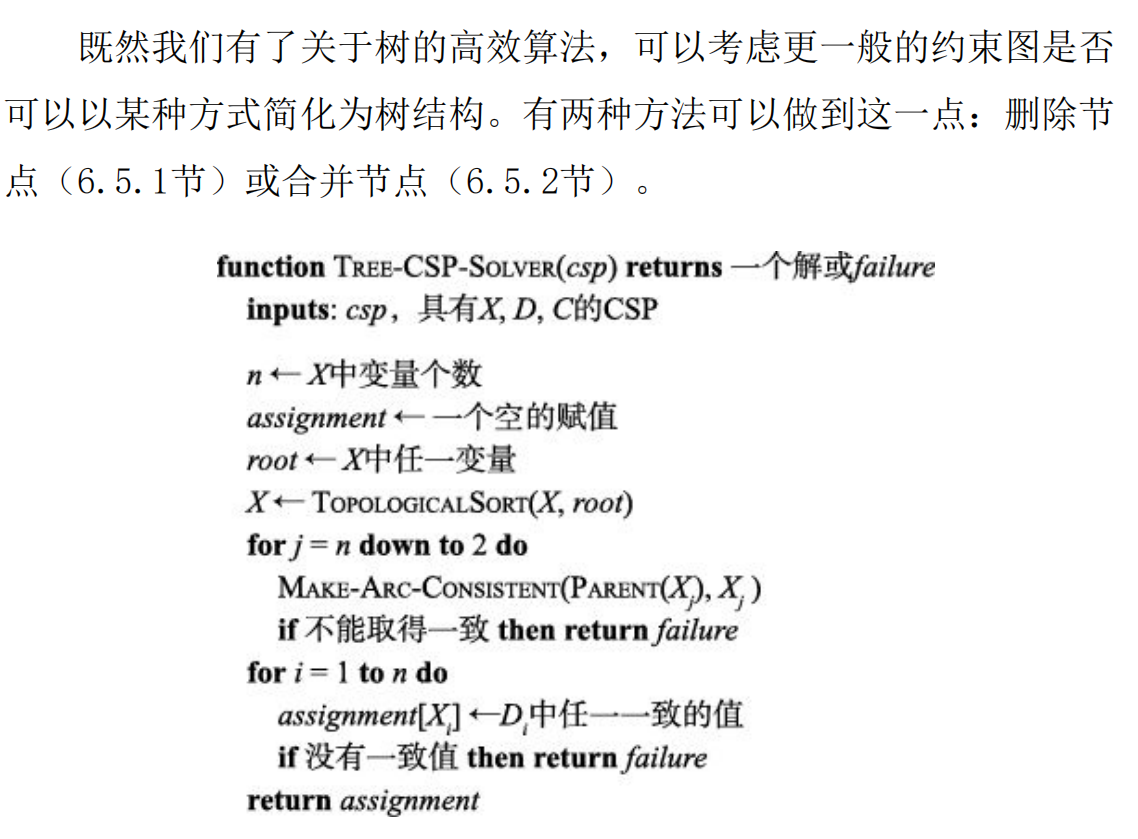
任何树状结构的CSP都可以在**变量个数**的**线性时间内求解**

定向弧一致性（directional arc consistency）DAC

变量顺序为X1, X2, …, Xn的CSP称为定向弧一致的，当且仅当，j>i时，每个Xi相对于每个Xj都是弧一致的

求解树结构的约束图，需要进行**拓扑排序**

任何有n个节点的树都有n−1条边，所以可以在O(n)步内使得该图具有定向弧一致性，每一步都必须比较两个变量的最多d个可能的值，总时间为**O(nd2)**。



**6.5.1割集调整**

将约束图化为树的第一种方法：为部分变量赋值

（1）选择CSP变量的一个子集S，使得约束图在删除S后成为一棵树。S称为环割集（cycle cutset）。

（2）对于满足S上所有约束的、S中变量的每种可能赋值，

a. 从剩余变量的域中**删除**任何**与S赋值不一致的值**，并尝试求解剩下的CSP树

b. 如果剩余的CSP存在一个解，那么将其连同S的赋值一起返回。

如果环割集的大小为c，需要尝试S中变量的值的所有dc种组合，对于每种组合，我们需要求解一个大小为(n −c)的树问题，那么总运行时间为O(dc·(n-c)d2)

寻找**最小**环割集是NP-hard的，但有高效的近似算法：割集调整

**6.5.2树分解**

将约束图化为树的第二种方法：树中的**每个节点由一组变量组成**

组合需要满足下列要求：

● 原始问题中的**每个变量**必须**至少**出现在**一个树节点**中。

● 如果两个变量在原始问题中**由一个约束连接**，那么它们**必须同时出现（连同约束）在至少一个树节点中**。

● 如果**一个变量出现在两个（分解后的）树节点**中，那么它必须出现在**连接这两个节点的路径上的所有节点**中。

树中的约束表明一个树节点中的变量必须与其相邻节点中的相应变量具有相同的值。

所以第三个条件但保证了原始问题的任何变量无论在哪出现都具有相同的值

Tree-CSP-Solver可以在O(nd2)时间内得到树状约束图的解，其中n是树节点的个数，d是最大域的大小，注意域是多个变量的值元组集合，一个节点内包含的变量越多d越大。

（相当于一个节点就是一个子问题）

一个给定的图允许多种树分解，在选择分解时，目标是使**子问题尽可能小**。

图的树分解的树宽（tree width）为最大节点的大小减1

图本身的树宽定义为其所有树分解的最小宽度

若一个图的树宽为w，则总的时间为O（ndw+1）

但找出树宽最小的分解是 一个NP-hard问题，

割集分解VS树分解

每当有一个大小为c的环割集时，也会有一个大小为w<c+1的树宽

则O(ndw+1） < O(dc·(n-c)d2)

**6.5.3值对称**

考虑有d种颜色的地图着色问题。对于每个一致解，实际上都有一组通过排列颜色名形成的d!个解

引入**对称性破缺约束：**对变量和域值施加一个任意的排序约束，要求可取的值按顺序赋给排列好的变量。

要**消除所有的对称性**是NP-hard的