约束满足问题—回溯法

19211241 王海博

本次阅读笔记，我学习了人工智能教材中的约束满足问题一章节，了解了约束满足问题的定义特点，以及常见的约束问题，并掌握了利用回溯法求解约束问题，进行了代码编写，能实现地图涂色问题的求解。

# 约束满足问题

什么是约束满足问题？

我们通过使用每个状态的分解表示一个问题：一组变量，每个变量都有一个值。当每个变量都有一个满足变量所有约束的值时，问题就解决了。以这种方式描述的问题称为约束满足问题（CSP），全程**Constraint Satisfaction Problem**，求解CSP部分分配问题通常是一个NP完全问题。

约束满足问题在人工智能领域有着广泛的应用。比如新的学期教室的规划分配，数独的解，它们都涉及了约束条件。我们所熟知的经典的皇后问题、SAT问题都属于约束满足问题。约束满足问题可以分为二元约束满足问题和多元约束满足问题。其中，多元约束满足问题可以被划分为等价的二元约束满足问题。因而，研究二元约束满足问题是一个重要的研究方向。

CSP搜索算法利用状态结构，并使用通用而非特定领域的启发式算法来解决复杂问题。其主要思想是通过识别违反约束的变量/值组合，一次性消除大部分搜索空间。

约束满足问题由X、D和C三部分组成：

X是一组变量，{X1，…，Xn}。

D是一组域，{D1，…，Dn}，每个变量对应一个域。

C是一组约束，用于指定允许的值组合。

约束满足问题的经典例子-----地图着色问题：我们的要求是给定三种颜色，对地图进行涂色，要求相邻区域要染成不同的颜色，地图如下图1-2-1所示：

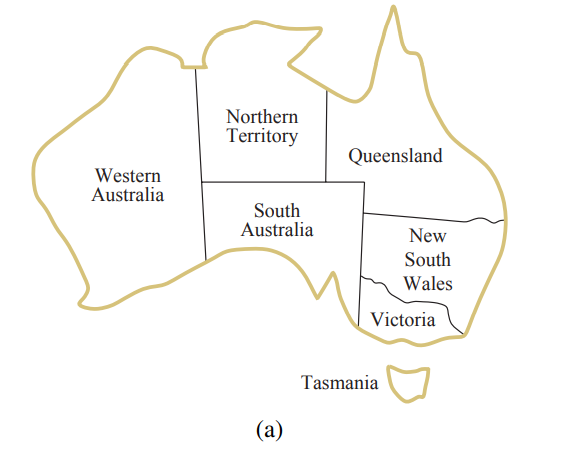


图1-1 地图着色

转化为上面提到的定义的形式：

变量集合X = {Western Australia ，Northern Territory ，South Australia ，Queensland，New South Wales，Victoria}

值域集合D = {red，green，blue}

约束集合C = {要求邻域不能是相同颜色}

那么为什么要把问题形式化成CSP呢？

第一个原因是用CSP表示各种问题会很自然，如果你有一个CSP求解系统，使用它求解问题，比其他搜索方法求解要简单的多。

第二个也是最重要的原因是CSP求解能快速消除庞大的搜索空间。

举个例子：利用约束传播，可以在赋值过程中逐步减少其他未赋值邻域的值域空间，如图1-1-1中，当Western Australia = 红 ，那么Northern Territory 和 South Australia 的值域中红色就可以去掉了。

# 约束传播

在常规的状态空间搜索中，算法只能做了搜索一件事。在CSP中则有了选择和剪枝，算法还可以做一种称为约束传播的特殊推理：使用约束来减少一个变量的合法取值范围，从而影响到跟此变量有约束关系的另一个变量的取值，如此进行。

解决约束问题的核心思想是：局部相容性，具体的种类主要有三种。

## 结点相容

如果单个变量（对应于CSP网络中的结点）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的。

一个通俗的例子，一个人有 {苹果，橘子，香蕉，西瓜}，但是他不喜欢橘子，所以我们把橘子拿走，那剩下的水果就符合了他的条件，即为结点相容的。

通过运行结点相容总能消除CSP中的所有一元约束。可以将所有n元约束转换成二元约束。因为结点约束要求把值域中不满足的值删除，所以在运行结点约束后，CSP中就不存在一元约束了。

## 弧相容

如果CSP中某个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。简单来说就是图中两个有弧连接的结点，它们各自的取值要满足它们两个之间的二元约束关系。例如：

y = 2x 对于（1，2），（2，4），（3，6），（4，8）

y的值域应该是{2，4，6，8}

x的值域应该是{1，2，3，4}

对于弧相容问题的解决方法有弧相容算法：AC-3

具体的操作过程：为了使每个变量都是弧相容，AC-3算法维护一个弧相容的队列。

（1）首先队列中包含CSP中所有的弧，AC-3从队列中弹出弧（,）

（2）要求对 弧相容（意思是的取值得出的结果要属于的值域）

（3）如果D没有发生变化（意思是中的所有值都满足，但也有可能存在不满足的，比如上面的例子中X=5，Y中没有10，就要去掉，那么D就发生了变化），算法则处理下一条弧

（4）如果D发生了变化，那么每个指向的弧（，）都必须重新插入队列中准备检验。之所以这么做是因为D的改变可能引起的缩小。

（5）如果D为空集，则CSP没有解，AC-3返回失败，否则，继续检查。

## 路径相容

弧相容通过弧（二元约束）缩紧值域（一元约束）；路径相容通过观察变量得到隐式约束并以此来加强二元约束。

两个变量的集合{,X3}对于第三个变量是相容的，指的是对{,}的每一个相容的赋值{=a,=b}，都有合适的取值同时使得{,},{,}是相容的。其实简单理解就是 -->-->的传递依赖一样。

## 全局约束

全局约束涉及任意数量的变量。全局约束经常出现在实际问题中，可以通过比目前描述的一般方法更有效的特殊算法来处理。

Alldiff约束要求所有相关变量必须取不同的值。

资源约束（atmost约束）：比如调度问题中，用p1,p2,p3,p4表示执行四项任务的人数，要求总人数不超过10人（资源约束）。我们可以通过检查当前域的最小值之和来检测不一致性；例如，如果每个变量都有域{3,4,5,6}，则无法满足Atmost约束。如果任何域的最大值与其他域的最小值不一致，我们还可以通过删除该域的最大值来强制一致性。因此，如果我们示例中的每个变量都有域{2,3,4,5,6}，那么可以从每个域中删除值5和6。

# 回溯搜索

## 回溯法的基本思路

回溯搜索用于深度优先搜索中，它每次为一个变量选择一个赋值，当没有合法的值可以赋给某变量时就回溯。与穷举的区别在于，回溯会进行一定的剪枝，即根据约束函数在结点处剪去不满足约束的子节点，在不满足约束函数的时候返回上一结点。

具体步骤：它不断选择未赋值变量，轮流尝试变量值域中的每一个值，试图找到一个解。一旦检测到不相容，返回上一次条用尝试另一个值，直到找个一个完整解。

回溯法解决问题时一般要建立解空间树进行分析，如图3-1所示。

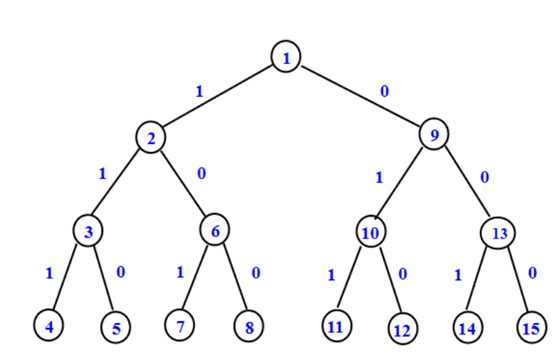


图3-1 解空间树

按照某个要求进行赋值，假设一开始结点都取1，当走到第3号结点时后，若发现在约束条件下无法继续赋值，不再对3号结点继续取值，而回溯到第2号结点，2号结点开始取0。

回溯法的基本思想：

（1）针对所给的问题，定义问题的解空间。要明确多少个结点，解空间树有几层，结点有几个分支。

（2）确定并表示解的约束条件。当不满足约束条件时进行剪枝

（3）结合剪枝进行深度优先搜索。一般采用递归算法

## 变量和取值顺序

（1）最少剩余值启发式(MRV),也称为“最受约束变量”或“失败优先启发式”。

它选择最可能很快导致失败的变量，从而对搜索树进行剪枝。

（2）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。

比如说还是图1-1，我们先选择SA的赋值，一旦SA的值确定了，那么WA,NT,Q,NSW,V的值域就可以进行删减，缩小它们的值域，进而更快的找到结果。

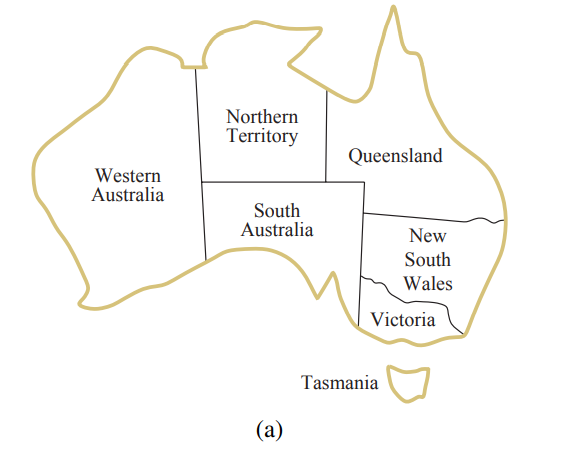
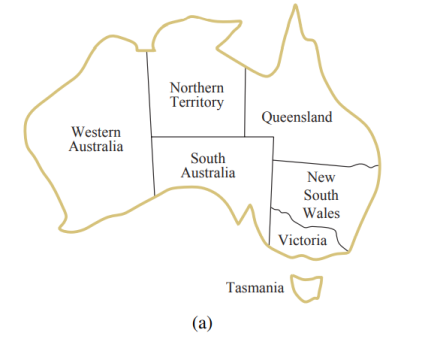


图1-1 地图着色

优先选择的值时给邻居变量留下更多的选择。MRV和度启发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。最少约束值启发式则是帮助变量选取合适的值。

## 回溯法解决着色问题

用红黄蓝三种颜色涂色，相邻地图不能同色。



首先，分析，Tasmania无约束，可随意取颜色。剩下的6个建一个6元的解向量，每个有三个取值。1=红 2=绿 3=蓝

对地区从上到下 从左到右依次编号为 1 2 3 4 5 6 7

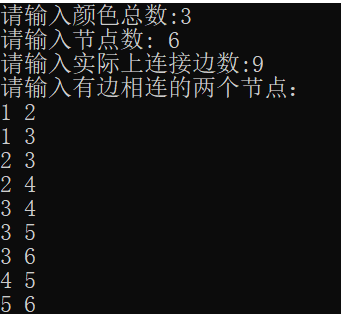
约束条件为相邻的不能共色，建立二维数组邻接矩阵，例：2与3相接 a[2][3]=1

从1号结点开始穷举，不共色继续下下一个结点取值，至第i个，若邻接矩阵第i行中是否有为1的，有则看是否与之共色，共色则回溯。

代码文件见压缩包

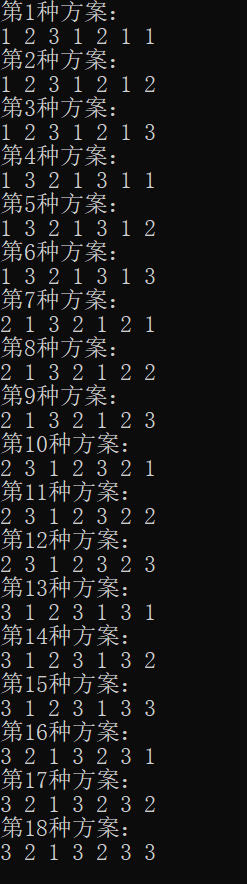
输入和输出结果见下图

输入：



解读一下 输入相连的两点 如 1 2 ；1 3

输出：



输出的是前六个地的涂色方案 第一种方案为 红 绿 兰 红 绿 红 红

# 学习总结

通过本次阅读学习，我学会了约束满足问题的研究方法和解决死了，掌握了核心在于利于局部相容性和约束条件，进行适当的剪枝；并利用回溯法成功编写了工程代码，解决地图着色问题，收获颇丰！