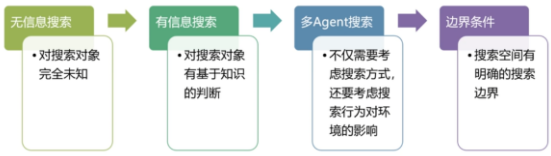
第五章 约束满足问题

目录：

- 约束满足问题简介
- 回溯搜索及改进技术（针对约束满足问题）
  - 变量和取值顺序
  - 通过约束传播信息
- 局部搜索
- 问题的结构

有边界搜索（约束条件搜索）

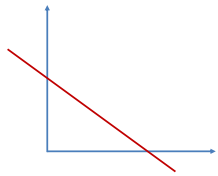


有边界搜索（约束条件搜索）

已知

$$z = x + y$$

求最大的z



有边界搜索（约束条件搜索）

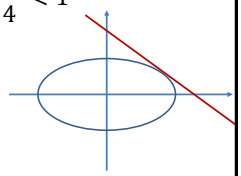
已知

$$z = x + y$$

且x和y满足公式：

$$\frac{x^2}{9} + \frac{y^2}{4} < 1$$

求最大的z



约束问题：几何求解

- 约束满足问题（Constraint satisfaction problems）的通用描述：
- 变量：X Y
- 取值范围：$[-\infty, +\infty]$
- 约束条件：$\frac{x^2}{a} + \frac{y^2}{b} < 1$

约束问题：几何求解

若x,y满足约束条件：$\begin{cases} 2x+y-2\le 0 \\ x-y-1\ge 0 \\ y+1\ge 0 \end{cases}$，则$z=x+7y$的最大值是多少，请给出具体的求解过程

举个例子：猜数字

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

- 变量：$F\ T\ U\ W\ R\ O$
- 取值范围：$\{0,1,2,3,4,5,6,7,8,9\}$
- 约束条件：每个变量不能一样，且要满足计算公式

举个例子：猜数字

$$\begin{array}{r} T\ W\ O \\ +\ T\ W\ O \\ \hline F\ O\ U\ R \end{array} \qquad \begin{array}{r} 7\ 3\ 4 \\ +\ 7\ 3\ 4 \\ \hline 1\ 4\ 6\ 8 \end{array}$$

- 变量：$F\ T\ U\ W\ R\ O$
- 取值范围：$\{0,1,2,3,4,5,6,7,8,9\}$
- 约束条件：每个变量不能一样，且要满足计算公式
- 右边是该问题的一个可行解

地图着色问题

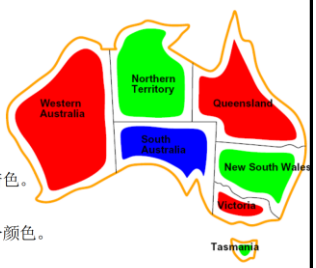
地图着色问题

地图着色问题

变量: WA, NT, Q, SA, NSW, V, T
取值范围：红色，绿色，蓝色
约束：任意两块地图之间不能使用相同的颜色

地图着色问题

- 澳大利亚地图包括7个区域。
- 用红，蓝，绿三个颜色给地图着色。
- 每个区域上一种颜色。
- 相邻的两个区域不能使用同一个颜色。



地图着色问题

- 变量： WA, NT, Q, NSW, V, SA, T
- 值域： D = {red, green, blue}
- 约束： 相邻的区域的颜色不同
  - 隐式表示： $WA \neq NT$
  - 显示表示： $(WA, NT) \in \{(red, green), (red, blue), \ldots\}$
- 问题的解是满足所有约束的一组变量赋值，例如：

  {WA=red, NT=green, Q=red, NSW=green, V=red, SA=blue, T=green}

N皇后问题

- 变量：棋盘某个格子是否有皇后 X_{ij}
- 值域：$\{0,1\}$
- 约束：

$$\forall i,j,k \ (X_{ij}, X_{ik}) \in \{(0,0),(0,1),(1,0)\}$$
$$\forall i,j,k \ (X_{ij}, X_{kj}) \in \{(0,0),(0,1),(1,0)\}$$
$$\forall i,j,k \ (X_{ij}, X_{i+k,j+k}) \in \{(0,0),(0,1),(1,0)\}$$
$$\forall i,j,k \ (X_{ij}, X_{i+k,j-k}) \in \{(0,0),(0,1),(1,0)\}$$
$$\sum_{i,j} X_{ij} = N$$

- 状态空间大小：2^N

真实世界中的约束满足问题

- 课程分配问题：比如，老师和课程的对应
- 课程时间表问题：比如，每一节课在什么时间和什么地点讲
- 硬件配置
- 交通排班
- 流水线安排
- 电路板布局
- 故障排查
- …

约束满足问题(Constraint Satisfaction Problems，CSPs)

- 标准的搜索问题：
  - 状态是原子结构，不可切分。可理解为“黑盒”，没有内部结构。
  - 目标检测是判断某个状态是否在一个目标状态集合中。
- 约束满足问题（CSPs）：
  - 状态被表示为特征向量。
  - 目标检测通过一组约束条件筛选符合条件的特征向量。

约束满足问题

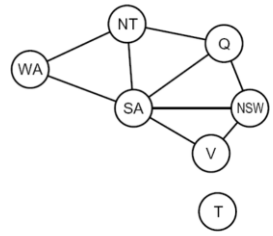
- K 个变量组成特征向量。
- 每个变量都有一个取值范围，也就是值域。
- 一个状态就是特征向量中变量的一个赋值组合。
  - 身高 = {矮，平均，高}
  - 体重 = {瘦，平均，重}
- 当特征向量中只有部分变量有赋值，称为部分赋值。
  - {身高 = 矮，体重=?}
- 特征向量的完整赋值可能是问题的一个可行解。
  - {身高 = 矮，体重 = 重}

约束满足问题和其他搜索问题的区别

- 搜索有明确的边界和约束条件
- 在边界内，满足约束的情况下一般解不唯一
- 解的形式一般是知道的（否则无法定义对解的约束）
- 到达目标解的路径未知，而且一般也无法设置启发函数和评价函数

约束图

- 二元约束满足问题：每一个约束条件至多涉及两个变量
- 二元约束图：节点表示变量，边表示约束



约束图

- 变量：

$$F\ T\ U\ W\ R\ O\ X_1\ X_2\ X_3$$

- 值域：

$$\{0,1,2,3,4,5,6,7,8,9\}$$

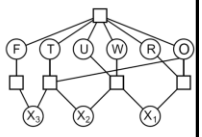
- 约束条件：

$$\text{alldiff}(F,T,U,W,R,O)$$

$$O+O=R+10\cdot X_1$$

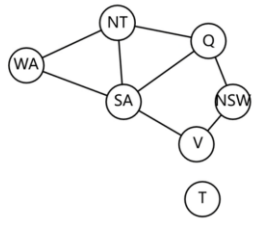
$$\cdots$$

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

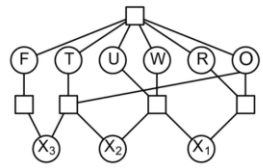


约束图举例

约束图可以简化约束条件的表述



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



Q：着色问题是否可以使用宽度优先或者深度优先搜索算法？

约束满足问题的定义

- 约束满足问题（Constraint Satisfying Problem, CSP）：
  - 由一个变量集合{X_1~X_n}和一个约束集合{C_1~C_m}定义；
  - 每个变量都有一个非空可能值域D_i；
  - 每个约束指定了包含若干变量的一个子集内各变量的赋值范围。
- 例如：地图染色问题，N-皇后问题。
- CSP的一个状态：对一些或全部变量的赋值 {X_i=v_i, X_j=v_j, …}。

23

CSP问题的解

- 一个不违反任何约束的对变量的赋值称为相容赋值或合法赋值。
- 对每个变量都进行赋值称为完全赋值。
- 一个（一组）对变量的赋值，若既是相容赋值又是完全赋值，则这个（组）赋值是CSP问题的解。
  - 某些CSP问题要求问题的解能使目标函数最大化——约束优化。
- CSP问题常常可以可视化，表示为约束图，更直观地显示问题，帮助思考问题的答案。

24

CSP问题的分类

- 根据变量的类型划分：离散值域和连续值域。
- 变量—离散值域
  - 有限值域，如地图染色问题，八皇后问题。
  - 无限值域，如整数集合或者字符串集合。
    - 例如，对于作业规划问题，无法枚举所有可能取值，要使用约束语言(线性约束/非线性约束)描述，如 $StartJob_1+5 \leq StratJob_3$。
- 变量—连续值域
- 最著名的连续值域CSP是线性规划问题。
  - 线性规划中的约束必须是构成一个凸多边形的一组线性不等式。
  - 线性规划问题可以在变量个数的多项式时间内求解。

25

CSP问题的分类

- 根据约束的类型划分：
  - 线性或非线性约束。
  - 一元或多元约束。
    - 一元约束：只限制一个变量的取值
    - 二元约束与2个变量相关
    - 高阶约束：涉及3个或更多变量。
      - 通过引入辅助变量，转为二元约束。
  - 绝对约束 vs 偏好约束。
    - 我们仅讨论绝对约束。

26

约束的不同类型

- 一元约束仅涉及一个变量：
  $SA \neq green$
- 二元约束涉及两个变量，例如：
  $SA \neq WA$
- 更高阶的约束涉及到三个或者更多变量：比如，数独问题的列约束
- 全局约束，比如:数独中的 AllDiff()

- 倾向性(软约束):
  - 比如：苹果比香蕉更好。
  - 变量的赋值往往伴随与倾向性相关的代价。

CSP问题求解的复杂度

- CSP问题的求解目标是找到相容的完全赋值，最朴素的想法是依次取变量的赋值组合并检查其是否满足约束条件。
  - 若CSP问题的任何一个变量的最大值域为d，那么可能的完全赋值数量为$O(d^n)$。
  - 指数级计算量。

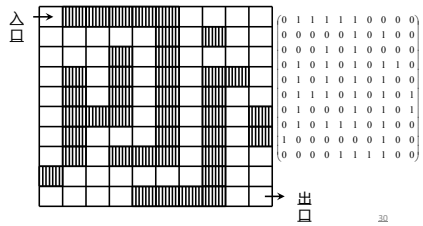
28

目录：

- 约束满足问题简介
- 回溯搜索及改进技术（针对约束满足问题）
  - 变量和取值顺序
  - 通过约束传播信息
- 局部搜索
- 问题的结构

29

引例：迷宫老鼠问题



30

标准的搜索问题设定

- 约束满足问题的标准搜索设定
- 状态对应一个特征向量（变量的赋值组合，部分赋值或者完全赋值）
  - 初始状态：空赋值，{}。
  - 后继函数：给一个未赋值变量赋值。
  - 目标检测：所有变量都被赋值，并且满足所有约束条件。

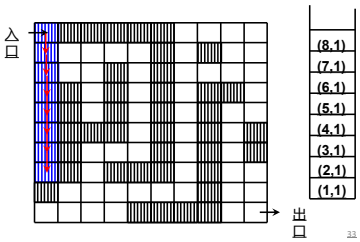
31

搜索方法

- 分支因子?
- 树的深度?
- 树的叶子结点数量?
- 广度优先搜索会做什么?
- 深度优先搜索会做什么?
- 朴素搜索会有哪些问题?
  - 变量排序问题
  - 如何利用约束条件

32

引例：迷宫老鼠问题



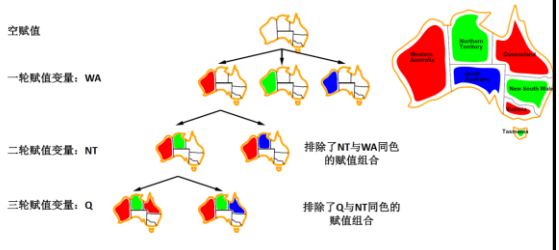
33

回溯搜索

- 回溯搜索是最基本的解决约束满足问题的搜索方法。
- 想法一：变量排序，按照固定顺序依次为变量赋值，每轮一个。
  - 在朴素搜索方法中，变量的赋值顺序可交换，带来重复处理。
  - 如，[WA = red，NT = green] 和 [NT = green，WA = red] 是一样的
- 想法二：冲突停止，变量赋值的过程中，同时检查约束条件。
  - 只考虑和现有赋值不存在冲突的取值，也称“递增式的约束检测”。
  - 可能额外引入检测约束条件的运算。
- 结合想法一和二的深度优先搜索，被称为回溯搜索

34

回溯搜索



35

回溯搜索

- 类似枚举的搜索尝试过程，主要是在搜索尝试过程中寻找问题的解，当探索到某一步时，发现原先选择并不优或达不到目标，就退回一步重新选择，尝试别的路径。
  - “走不通就退回再走”
- 满足回溯条件的某个状态的点称为“回溯点”

回溯搜索

- 回溯法：通常以深度优先的方式系统地搜索问题的解的算法
- 使用场合
  - 对于许多问题，当需要找出它的解的集合或者要求回答什么解是满足某些约束条件的最佳解时，往往要使用回溯法。
  - 这种方法适用于解一些组合数相当大的问题，具有“通用解题法”之称。
- 回溯法的基本做法
  - 是搜索，或是一种组织得井井有条的，能避免不必要搜索的穷举式搜索法。

回溯搜索

具体做法

- 系统性

回溯法在问题的解空间树中，通常按深度优先策略，从根结点出发搜索解空间树。

- 跳跃性

算法搜索至解空间树的任意一点时，先判断该结点是否包含问题的解。如果肯定不包含，则跳过对该结点为根的子树的搜索，逐层向其祖先结点回溯；否则，进入该子树，继续按深度优先策略搜索。

回溯搜索

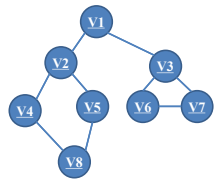
回溯法与穷举查找是一样的吗？

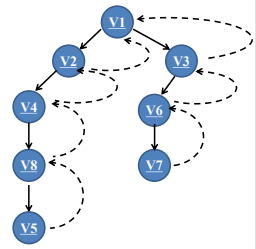
- 可以把回溯和分支限界看成是穷举法的一个改进。

  该方法至少可以对某些组合难题的较大**实例求解**。
- 不同点
  - 每次只构造侯选解的一个部分
  - 然后评估这个部分构造解：如果加上剩余的分量也不可能求得一个解，就绝对不会生成剩下的分量

回溯搜索

回溯算法：**图的深度优先遍历**





回溯法的算法框架

问题的解空间

- 问题的解向量

  回溯法希望一个问题的解能够表示成一个n元式$(x_1,x_2,\ldots,x_n)$的形式。
- 显约束

  对分量x_i的取值限定。
- 隐约束

  为满足问题的解而对不同分量之间施加的约束。

回溯法的算法框架

解空间（Solution Space）

- 对于问题的一个实例，解向量满足显式约束条件的所有多元组，构成了该实例的一个解空间。

注意：同一问题可有多种表示，有些表示更简单，所需状态空间更小（存储量少，搜索方法简单）。

回溯法的算法框架

例如

对于有n种可选物品的0-1背包问题

✓ 解空间由2^n个长度为n的0-1向量组成

✓ n=3时，解空间为{(0,0,0), (0,0,1), (0,1,0), (0,1,1), (1,0,0), (1,0,1), (1,1,0), (1,1,1)}

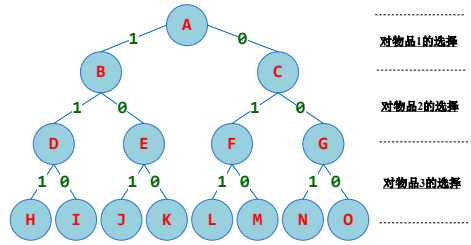
用**完全二叉树**表示的**解空间**

✓ 边上的数字给出了向量x中第i个分量的值x_i

✓ 根节点到叶节点的路径定义了解问题的一个解

回溯法的算法框架

子集树：所给问题是从n个元素的集合S中找出S满足某种性质的子集



回溯法的算法框架

回溯法的基本思想

- 回溯法的基本步骤

(1)针对所给问题，定义问题的解空间；

(2)确定易于搜索的解空间结构；

(3)通常以**深度优先**方式搜索解空间，并在搜索过程中用剪枝函数避免无效搜索。

- 常用剪枝函数

➢ 用约束函数在扩展结点处剪去不满足约束的子树；

➢ 用限界函数剪去得不到最优解的子树。

回溯法的算法框架

空间复杂性

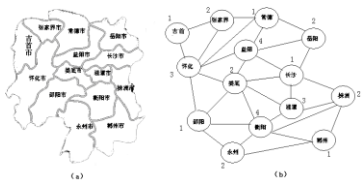
- 用回溯法解题的一个显著特征是在搜索过程中动态产生问题的解空间。在任何时刻，**算法只保存从根结点到当前扩展结点的路径。**
- 如果解空间树中从根结点到叶结点的最长路径的长度为$h(n)$，则回溯法所需的计算空间通常为$O(h(n))$。
- 显式地存储整个解空间则需要$O(2^{h(n)})$或$O(h(n)!)$内存空间。

例：图的m着色问题

图的着色问题是由地图的着色问题引申而来的：用m种颜色为地图着色，使得地图上的每一个区域着一种颜色，且相邻区域颜色不同。

图的m着色问题

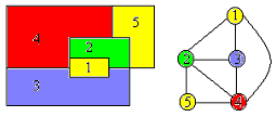
问题建模：如果把每一个区域收缩为一个顶点，把相邻两个区域用一条边相连接，就可以把一个区域图抽象为一个平面图。



例如，图（a）所示的区域图可抽象为（b）所表示的平面图。

图的m着色问题

- 图着色问题描述为：给定无向连通图G=(V, E)和正整数m，求最小的整数m，使得用m种颜色对G中的顶点着色，使得任意两个相邻顶点着色不同。

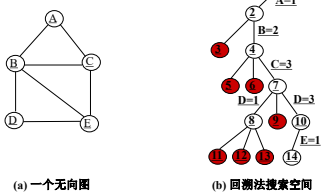


图的m着色问题

- 由于用m种颜色为无向图G=(V, E)着色，其中，V的顶点个数为n，可以用一个n元组C=(c_1, c_2, …, c_n)来描述图的一种可能着色，其中，$c_i \in \{1, 2, \ldots, m\}(1 \le i \le n)$表示赋予顶点$i$的颜色。
- 例如，5元组(1, 2, 2, 3, 1)表示对具有5个顶点的无向图的一种着色，顶点1着颜色1，顶点2着颜色2，顶点3着颜色2，如此等等。
- 如果在n元组C中，所有相邻顶点都不会着相同颜色，就称此n元组为可行解，否则为无效解。

图的m着色问题

回溯法求解图着色问题示例



(a) 一个无向图　　(b) 回溯法搜索空间

图的m着色问题

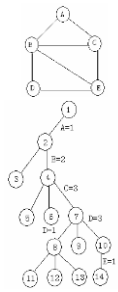
① 把5元组初始化为（0,0,0,0,0），从根结点开始向下搜索，以颜色1为顶点A着色，生成结点2时，产生(1,0,0,0,0)，是个有效着色。

② 以颜色1为顶点B着色生成结点3时，产生(1,1,0,0,0)，是个无效着色，结点3为d_结点。

③ 以颜色2为顶点B着色生成结点4，产生(1,2,0,0,0)，是个有效着色。

④ 分别以颜色1和2为顶点C着色生成结点5和6，产生(1,2,1,0,0)和(1,2,2,0,0)，都是无效着色，因此结点5和6都是d_结点。

⑤ 以颜色3为顶点C着色，产生(1,2,3,0,0)，是个有效着色。重复上述步骤，最后得到有效着色(1,2,3,3,1)。



图的m着色问题

回溯法求解图着色问题：

- 首先把所有顶点的颜色初始化为0，然后依次为每个顶点着色。如果其中i个顶点已经着色，并且相邻两个顶点的颜色都不一样，就称当前的着色是有效的局部着色；否则，就称为无效的着色。
- 如果由根节点到当前节点路径上的着色，对应于一个有效着色，并且路径的长度小于n，那么相应的着色是有效的局部着色。这时，就从当前节点出发，继续探索它的儿子节点，并把儿子结点标记为当前结点。在另一方面，如果在相应路径上搜索不到有效的着色，就把当前结点标记为d_结点，并把控制转移去搜索对应于另一种颜色的兄弟结点。
- 如果对所有m个兄弟结点，都搜索不到一种有效的着色，就回溯到它的父亲结点，并把父亲结点标记为d_结点，转移去搜索父亲结点的兄弟结点。这种搜索过程一直进行，直到根结点变为d_结点，或者搜索路径长度等于n，并找到了一个有效的着色为止。

图的m着色问题

设数组$color[n]$表示顶点的着色情况，回溯法求解m着色问题的算法如下：

1. 将数组$color[n]$初始化为0；
2. k=1；
3. while (k>=1)

3.1 依次考察每一种颜色，若顶点k的着色与其他顶点的着色不发生冲突，则转步骤3.2；否则，搜索下一个颜色；

3.2 若顶点已全部着色，则输出数组color[n]，返回；

3.3 否则，

3.3.1 若顶点k是一个合法着色，则k=k+1，转步骤3处理下一个顶点；

3.3.2 否则，重置顶点k的着色情况，k=k-1，转步骤3。

一个简单的回溯算法（深度优先）

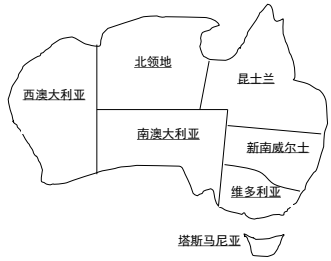
```
1.  function BackTracking-Search( csp ) returns a solution, or failure
2.    return Recursive-BackTracking( {}, csp )

1.  function Recursive-BackTracking( assignment, csp ) returns a solution, or
    failure
2.    if assignment is complete then return assignment
3.    var←Select-Unassigned-Variable(Variables[csp], assignment, csp)
4.    for each value in Order-Domain-Values( var, assignment, csp ) do
5.      if value is consistent with assignment according to
          Constraints[csp] then
6.        add { var=value } to assignment
7.        result ← Recursive-BackTracking( assignment, csp )
8.        if result != failure then return result
9.        remove {var=value } from assignment
10.   return failure
```

55

例1、澳大利亚地图染色问题(1)

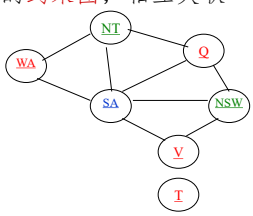
- 澳大利亚地图染色问题：用红绿蓝3色标出各省，相邻者颜色不同。



56

例1、澳大利亚地图染色问题(2)

- 对应于澳大利亚地图的约束图，相互关联的节点用边连接。
  - 西澳大利亚 – WA
  - 北领地 – NT
  - 南澳大利亚 – SA
  - 昆士兰 – Q
  - 新南威尔士 – NSW
  - 维多利亚 – V
  - 塔斯马尼亚 – T

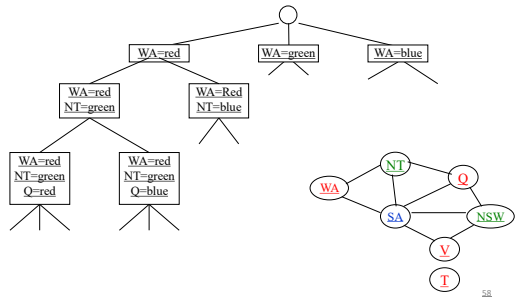


- 一组满足约束的完全赋值：{ WA=R, NT=G, Q=R, SA=B, NSW=G, V=R, T=R }

57

简单回溯法生成的搜索树

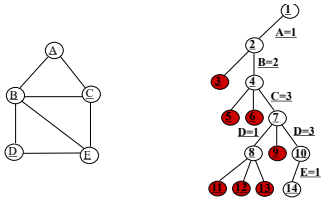
- 澳大利亚地图染色问题的搜索树



58

图的m着色问题

- 如何提升算法效率？
  - 1、接下来应该对哪个变量赋值?
  - 2、应该以什么顺序尝试它的值?
  - 3、我们能在故障出现早期发现吗?
  - 4、我们能利用问题本身的结构来改善搜索吗?



(a) 一个无向图　　(b) 回溯法搜索空间

59

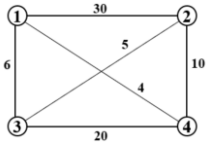
目录：

- 约束满足问题简介
- 回溯搜索及改进技术（针对约束满足问题）
  - 变量和取值顺序
  - 通过约束传播信息
- 局部搜索
- 问题的结构

60

思考题：旅行商问题

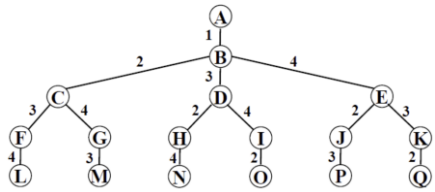
- 某售货员要到若干城市去推销商品，已知各城市之间的路线(或旅费)。要选定一条从驻地出发，经过每个城市一遍，最后回到驻地的路线，使总的路程(或总旅费)最小。
- 举例：4个城市的情况。



来源于网络：https://zhuanlan.zhihu.com/p/141680086

思考题：旅行商问题

- 搜索空间：排列树（所给问题是确定n个元素满足某种性质的排列）
- 思考：解空间是排列树与子集树的区别



CSP的回溯搜索

- CSP问题具有一个性质：可交换性，变量赋值的顺序对结果没有影响。
  - 所有CSP搜索算法生成后继节点时，在搜索树每个节点上只考虑单个变量的可能赋值。
- CSP问题的求解：深度优先的回溯搜索。
  - 每次给一个变量赋值，当没有合法赋值(不满足约束时)就要推翻前一个变量的赋值，重新给其赋值，这就是回溯。

63

回溯搜索的通用算法

- 需改善无信息回溯搜索算法的性能。
- 通用改进方法的思路：
  - 下一步该给哪个变量赋值，按什么顺序给该变量赋值？
  - 每步搜索应该做怎样的推理？当前变量的赋值会对其他未赋值变量产生什么约束，怎样利用这种约束以提高效率。
  - 当遇到某个失败的变量赋值时，怎样避免同样的失败？就是说如何找到对这种失败起到关键作用的某个变量赋值。

64

CSP回溯搜索的改进

- 基于变量和赋值次序的启发式
- 搜索与推理交错进行
- 智能回溯：向后看

65

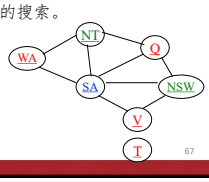
CSP回溯搜索的改进

- 基于变量和赋值次序的启发式
  - MRV（最少剩余值）启发式
  - 最少约束值启发式
  - 度启发式
- 搜索与推理交错进行
- 智能回溯：向后看

66

（1）最少剩余值（MRV）启发式

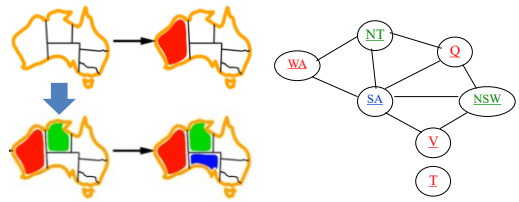
- 随机的变量赋值排序难以产生高效率的搜索。
- 例如：在WA=red/NT=green条件下选取SA赋值比Q要减少赋值次数(1:2)，并且一旦给定SA赋值以后，Q、NSW和V的赋值只有一个选择。
- 因此，应当选择合法取值最少的变量，即最少剩余值(MRV)启发式，也称为最受约束变量启发式或失败优先启发式。
  - 称为失败优先启发式是因为它可以很快找到失败的变量，从而引起搜索的剪枝，避免更多导致同样失败的搜索。



67

（1）最少剩余值（MRV）启发式

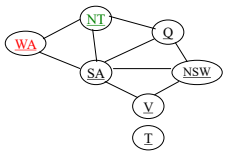
- 因此，应当选择合法取值最少的变量，即最少剩余值(MRV)启发式，也称为最受约束变量启发式或失败优先启发式。
  - 称为失败优先启发式是因为它可以很快找到失败的变量，从而引起搜索的剪枝，避免更多导致同样失败的搜索。



68

（2）最少约束值启发式

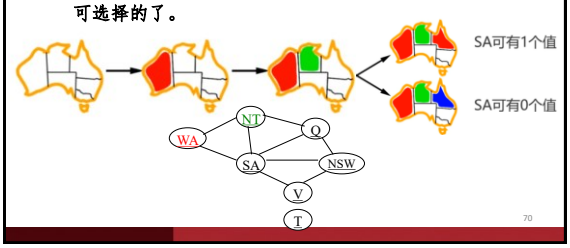
- **MRV启发式：当有多个变量需要选择时，优先选择在当前约束下取值最少的变量。**
  - **问题：一旦变量被选定，算法就要决定它的取值的次序。**
- **最少约束值启发式：当赋值的变量有多个值选择时，优先的值应是约束图中排除邻居变量的可选值最少的，即优先选择为剩余变量的赋值留下最多选择的赋值。**



69

（2）最少约束值启发式

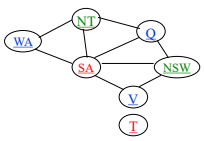
- **最少约束值启发式：当赋值的变量有多个值选择时，优先的值应是约束图中排除邻居变量的可选值最少的，即优先选择为剩余变量的赋值留下最多选择的赋值。**
  - **例如，WA=red且NT=green时，如果给Q赋值，可以为blue或red，而Q=blue的选择不好，因为此时SA没有一个可选择的了。**



70

（3）度启发式

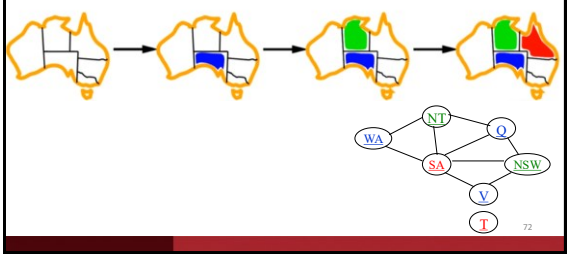
- MRV启发式：当有多个变量需要选择时，优先选择在当前约束下取值最少的变量。
  - 问题：对澳大利亚地图着色问题中选择第一个染色问题根本没有帮助，因为初始的时候每个区域都有3种选择。
- 度启发式：选择涉及对其他未赋值变量的约束数量大（与其他变量关联最多）的变量。



71

（3）度启发式

- MRV启发式：当有多个变量需要选择时，优先选择在当前约束下取值最少的变量。
- 度启发式：选择涉及对其他未赋值变量的约束数量大（与其他变量关联最多）的变量。



72

CSP回溯搜索的改进

- 基于变量和赋值次序的启发式
- 搜索与推理交错进行
- 智能回溯：向后看

73

搜索与推理交错进行

- 一般的回溯算法：只有在选择了一个变量的时候才考虑该变量的约束。
- 变量约束的启发式：在搜索的早些时候，或开始之前就考虑某些约束，从而降低搜索空间。
  - ① 约束传播
  - ② 最简单的推理形式：前向检验

```
  T W O
+ T W O
F O U R
```

74

（1）前向检验

- 前向检验：如果X被赋值，检查与X相连的那些变量Y，看看它们是否满足相关约束，并从Y的值域中删去与X取值矛盾的那些赋值。

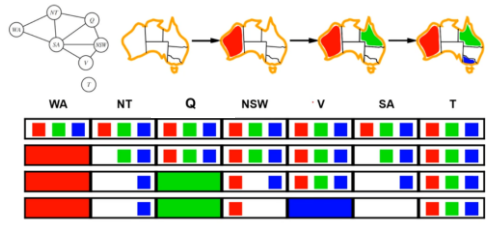
蓝色字体为赋值结果	WA	NT	Q	NSW	V	SA
	RGB	RGB	RGB	RGB	RGB	RGB
WA=red	R	GB	RGB	RGB	RGB	GB
Q=green	R	B	G	R B	RGB	B
V=blue	R	B	G	R	B	---

- 赋值V=blue引起矛盾，此时SA赋值为空，不满足问题约束，此时算法就要立刻回溯。

75

（1）前向检验

- 前向检验：如果X被赋值，检查与X相连的那些变量Y，看看它们是否满足相关约束，并从Y的值域中删去与X取值矛盾的那些赋值。



76

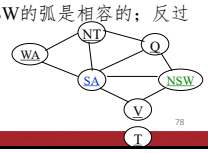
（1）前向检验

- 前向检验可与MRV启发式相结合。
  - 实际上，MRV要做的就是向前找合适的变量。
- 注意：这里只是检验一步，即和当前节点是否矛盾。
  - 前向检验看得不够远。虽然前向检验能检验出许多矛盾，它还是不能检验出所有的矛盾。
  - 被检验节点之间的约束检验还不能进行。
  - 改进：约束传播。

77

（2）约束传播

- 约束传播：使用约束来减小一个变量的合法取值范围，从而影响到与此变量有约束关系的另一变量的取值。
- 弧相容：依次检验约束图中各个相关节点对。注意，弧是有向弧。
  - 例如，给定SA/NSW当前值域，如果对于SA的每个取值x，NSW都有某个y和x相容，则SA到NSW的弧是相容的；反过来是NSW到SA的弧相容。



78

约束传播：弧相容

- 在地图染色约束的赋值过程中：第4行SA={blue}，NSW={red, blue}，则SA的取值有一个NSW=red与之相容；反过来NSW=blue，则SA为空值，即不相容，通过删除NSW值域中的blue可使其相容。
- 弧相容检测也能较早发现矛盾，如第4行SA和NT值域均为{blue}，则必须删去SA=blue，其值域变为空。
  - 用弧相容能够更早地检测到矛盾。

蓝色字体为赋值结果

	WA	NT	Q	NSW	V	SA
	RGB	RGB	RGB	RGB	RGB	RGB
WA=red	R	GB	RGB	RGB	RGB	GB
Q=green	R	B	G	R B	RGB	B
V=blue	……	……	……	……	……	……79

弧相容算法思想

1. 用队列记录需要检验不相容的弧。
2. 每条弧$[X_i, X_j]$依次从队列中删除并被检验。
   - 如果X_i值域中的任何一个值需要删除，则每个指向X_i的弧$[X_k, X_i]$都必须重新插入队列进行检验。
     - 因为指向这个变量的弧可能产生新的不相容（原来可能就是因为这个值产生了它们之间的相容）。

- 时间复杂度：二元CSP约束至多有$O(n^2)$条弧；每条弧至多插入队列d次（d个取值），检验一条弧为$O(d^2)$，因此算法的最坏情况下为$O(n^2d^3)$。

80

弧相容算法AC-3

```
1.  function AC-3(csp) returns the CSP, possibly with reduced domains
2.    inputs: csp, a binary CSP with variables {X1, X2, ..., Xn}
3.    local variables: queue, a queue of arcs, initially all the arcs in csp
4.    while queue is not empty do
5.      (Xi, Xj) ← Remove-First( queue )
6.      if RM-Inconsistent-Values(Xi, Xj) then
7.        for each Xk in Neighbors[Xi] do
8.          add (Xk, Xi) to queue

1.  function RM-Inconsistent-Values(Xi, Xj)
      returns true iff remove a value
1.    removed ← false
2.    for each x in Domain[Xi] do
3.      if no value y in Domain[Xj] allows (x,y) to satisfy constraint(Xi, Xj)
4.        then delete x from Domain[Xi]; removed ← true
5.    return removed
```

81

弧相容的使用

- 弧相容检验可以用作开始搜索之前的预处理。
- 弧相容检验也可以在搜索过程中每次赋值后用作一个约束传播步骤。
  - 即反复检测某个变量值域中的不相容弧，进行值删除，直到不再有矛盾。
  - 称为保持弧相容（MAC）。

82

从弧相容推广到k相容

- 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与前$k-1$个变量相容的值，那么该CSP问题是k相容的。
- 1相容是指每个单独的变量自己是不矛盾的，也称为节点相容。
- 弧相容＝2相容。
- 3相容是指任何一对相邻的变量总可以扩展到第三个邻居变量，也称为路径相容。
- 如果一个图是k相容的，也是k-1相容的、k-2相容的，……，直到1相容，那么这个图是强k相容的。

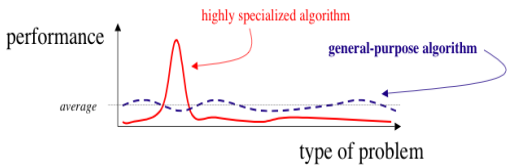
83

从弧相容推广到k相容

- 如果一个CSP问题有n个结点，而且它是强n相容的，则不需要回溯就能求解这个问题。
  - 可以在$O(nd)$时间内找到解。
- No free Lunch：任何建立n相容的算法在最坏情况下必须花费n的指数级时间。
- 从弧相容到n相容：执行较强的相容性检验会花费更多的时间，但是会更有效地降低分支因子和检测出矛盾的不完全赋值。

84

约束传播：特殊约束



- 实际问题中出现的特殊约束，用专用算法处理其效率要比通用的约束处理算法高很多。

85

特殊约束-ALLDiff约束

- 例如，AllDiff约束，变量取值各不相同
  - 假设约束涉及m个变量，所有变量共有n个取值，如果$m>n$，则此约束不能被满足。
- 引出相应的算法：
  - 删除约束中只有单值值域的变量，将这些变量的取值从其余变量值域中删去；
  - 对单值变量重复此过程；
  - 如果得到空的值域、或剩下的变量数大于取值数，则产生矛盾。

86

特殊约束-资源约束

- 资源约束，有时称为atmost约束。
- 例如，用$PA_1, \ldots, PA_4$表示分配给四项任务的人员个数，人员数总计不超过10人的约束记为atmost(10, $PA_1, \ldots, PA_4$)。
  - 如果每个变量的赋值为{3, 4, 5, 6}，就不能满足atmost约束。
  - 通过检验当前值域中的最小值之和就能检测出矛盾。

87

特殊约束

- 对于大型的整数值的资源限制问题，用整数集合来表示每个变量的值域然后通过相容性检验方法逐步削减集合，通常是不可能的。
  - 取代办法：值域用上界和下界来表示，并通过边界传播来管理。
- 例、假设有2次航班，271和272，它们分别有165和385个座位。每次航班可承载的初始值域为

  Flight271∈[0, 165]，Flight272∈[0, 385]。

  设又增加一个约束，两次航班所承载的总乘客数必须是420。通过边界传播约束，可以把这两个值域削减到

  Flight271∈[35, 165]，Flight272∈[255, 385]。
- 如果对于每个变量X和它取值的上下界，每个变量Y都存在某个取值满足X和Y之间的约束，我们称该CSP问题是边界相容的。

88

CSP回溯搜索的改进

- 基于变量和赋值次序的启发式
- 搜索与推理交错进行
- 智能回溯：向后看

89

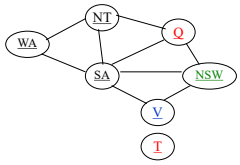
向后看—智能回溯

- 在回溯算法中，当发现不满足约束即搜索失败时，则回到上一个变量并尝试下一个取值，这称为历时回溯。
  - 在很多情况下这样做是效率很低的，因为问题并不决定于上一个（甚至几个）变量的取值。
- 回溯应倒退到导致失败的变量的集合中的一个变量。
  - 该集合称为冲突集。
- 变量X的冲突集是通过约束与X相连接的、先前已赋值变量的集合。

90

冲突集

- 对于地图染色问题，设有不完全赋值
  {Q=red, NSW=green, V=blue, T=red} 。



- 此时，给SA赋值将发现无法满足任何约束，SA的冲突集={Q, NSW, V}。

91

后向跳转

- 后向跳转：回溯到冲突集中时间最近（最后赋值）的变量。
- 对于前向检验算法，可以很容易得到冲突集。
  - 基于X赋值的前向检验从变量Y的值域中删除一个值时，说明X和Y存在冲突，则显然X是Y的冲突集中的一个变量。
  - 当到达Y时，可知回溯到哪个变量。
- 对于弧相容检验，因为都是做取值相容的检测，只要在弧相容检验时增加一个变量集合记录即可。

92

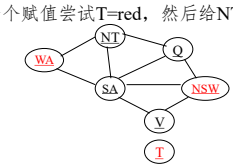
后向跳转

- 问题：后向跳转只出现在值域中的每个值都和当前的赋值有冲突的情况下，但是前向检验能检测到这个事件并且一旦到达这样的结点就阻止搜索。
  - 可以证明：每个被后向跳转剪枝的分支在前向检验算法中也被剪枝。
  - 因此，简单的后向跳转在前向检验搜索中，或者在诸如MAC（保持弧相容）这样使用更强的相容性检验的搜索中是多余的。

93

冲突指导的后向跳转

- 很多情况下，一个分支发生很久以前就已经注定要失败了。
- 例，考虑不完全赋值{WA=red, NSW=red}。
  - 假设下一个赋值尝试T=red，然后给NT，Q，V，SA赋值。



  - 因为最后的四个变量NT，Q，V，SA没有相容的赋值，因此最终我们尝试了NT的所有可能取值。
  - 现在的问题是向哪里回溯？
  - 后向跳转行不通：NT确实有和已赋值变量相容的值，但NT没有完整的由前面能导致失败的变量组成的冲突集。

94

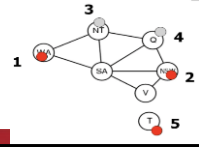
冲突指导的后向跳转

- 变量冲突集更一般的情况：前面的变量集合使得当前变量连同任何后继变量一起没有相容解。
- 冲突指导的后向跳转处理：
  - 令X_j是当前变量，conf(X_j)是其冲突集。
  - 如果X_j每个可能取值都失败了，则后向跳转到conf(X_j)中最近的一个变量X_i，令
    conf(X_i)=conf(X_i)∪conf(X_j) - {X_i}。
  - 从X_i向前是无解的，从X_i回到某个以前的变量赋值。

95

冲突指导的后向跳转

- 例，考虑不完全赋值{WA=red, NSW=red}。
  - 假设下一个赋值尝试T=red，然后给NT，Q，V，SA赋值。
  - 因为最后四个变量NT，Q，V，SA没有相容的赋值，回溯。
  - SA的冲突集是{WA, NT, Q}，后向跳转到Q。
  - Q将SA的冲突集（减去Q）吸收到自己的冲突集里，则Q的新的冲突集为{WA, NT, NSW}。
    - 给定了{WA, NT, NSW}的赋值后，从Q向前是无解的。
  - 回溯到NT，NT的冲突集变为{WA, NT, NSW} - {NT}+ {WA} = {WA, NSW}。
  - 后向跳转到NSW。



96

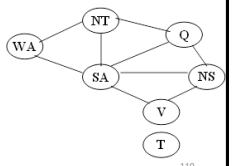
小结

- 约束满足问题（CSP），用图表示它的结构
- CSP问题的回溯搜索；深度优先搜索的一种形式
  - 从三个方面优化搜索
- 约束满足问题的局部搜索
  - 最小冲突启发式
- 问题的结构
  - 割集调整将问题化为树状结构
  - 树分解将问题化为子问题的树

109

课堂练习

- 澳大利亚地图染色问题，是指用红绿蓝3色标出各省，且相邻者颜色不同。下图给出了澳大利亚地图着色问题的约束图。
  ① 假设当前所有变量都还没有赋值，按照度启发式，请问应该选择哪一个变量进行赋值？
  ② 假设已有不完全赋值{WA=red，NT=green}，下面需要在SA或Q这两个变量之间选择一个进行赋值。如果采用MRV（最少剩余值）启发式来选择变量，请问应该选择哪一个？
  ③ 假设已有不完全赋值{Q=red, NSW=green, V=blue, T=red}。此时，给SA赋值将发现无法满足约束条件。请问SA的冲突集是什么？



110