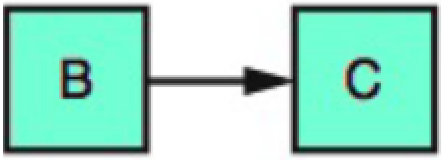
Lecture6 约束满足问题

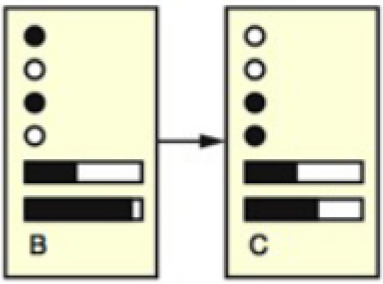
1. CSP 问题定义

搜索问题与 CSP

- 搜索问题定义
  - 状态：一个黑箱，通过一些数据结构实现
  - 目标检测：状态转移函数



- CSP 问题定义
  - 状态：一些变量 X_i，它们的取值范围是 D_i
  - 目标测试：一组约束，指定变量子集允许的值组合



CSP 的定义

约束满足问题由以下三个要素组成

- 一系列**变量**： $X = \{X_1, X_2, \ldots, X_n\}$
- 每个变量的**取值范围**： $D = \{D_1, D_2, \ldots, D_n\}$
- 一组**约束** C：指定允许的值组合

如何解决 CSP 问题

- 找到**满足所有约束条件**的分配值

概念

- 问题规范化
- 回溯搜索 backtracking search（DFS）
- 相容性检测 arc consistency
- 我们称 CSP 问题的解为**一致性分配 consistent assignment**

变量的类型

- 离散变量
  - 有限的取值范围，n 个变量，d 种取值，所以可以赋值的方案有 $O(d^n)$ 种
    - 地图上色问题，八皇后问题
  - 无限的取值范围，需要用约束语言
    - 例如工作调度 $T_1 + d \leq T_2$
- 连续变量
  - 具有线性或非线性等式的线性规划问题

约束的类型

- **一元约束 Unary constraints**：涉及单一变量
  - SA ≠ green
- **二元约束 Binary constraints**：涉及两个变量
  - SA ≠ WA
- **全局约束 Global constraints**：涉及 3 个或以上的约束
  - 密码术谜题，数独
- **偏好（软约束） Soft constraints**
  - 红色比绿色要好
  - 通常用每个变量赋值的成本来表示
  - 是约束优化问题

真实世界的 CSP 问题

- 调度问题：谁教哪个班级
- 规划问题：哪个课程在什么时候安排，安排在哪里
- 硬件配置
- 交通调度
- 工厂调度
- 楼层规划

很多真实世界的问题的变量是实数变量（连续的），而不是整数变量（离散的）

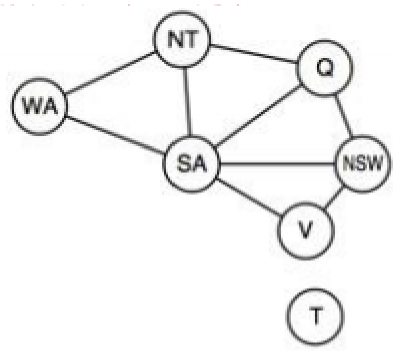
2. 地图上色问题

给上面的地图上色，可以使用的颜色为红色、绿色和蓝色，相邻的州之间的颜色不可以是相同的

问题规范化

- 变量：$X = \{WA, NT, Q, NSW, V, SA, T\}$
- 取值范围：$D_i = \{red, green, blue\}$
- 约束：相邻的区域必须有不同的颜色
  - 例如 $WA \neq NT$

约束图 Constraint graph



- 约束图
  - 每个变量用一个节点表示
  - 有约束的变量用一个边连起来
- 二项 CSP
  - **每一个约束只与最多两个变量相关**

示例解

- 如上图是一种可行解

3. 解决 CSP 问题

- CSP 算法：完成两件事情
  - 搜索：从众多可能性中选择一个新的变量进行赋值
  - 推断：约束传播，使用约束来传播信息：减少一个变量值的数量，这将减少其他变量的合法值
- 作为一个预处理步骤，约束传播有时可以完全不需要搜索就能解决这个问题
- 约束传播可以与搜索相互交织

定义

- **初始状态**：空的赋值集合 { }
- **状态**：部分的参数分配
- **继承者函数 successor function**：为未分配的变量分配一个值
- **目标检测 goal test**：目前的赋值是完整的，并且满足所有的约束

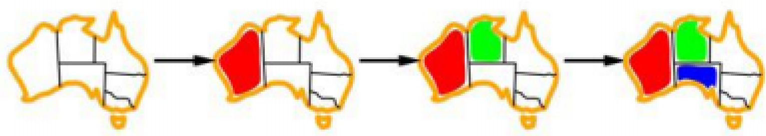
回溯搜索 Backtracking search BTS

- 一次赋值一个变量：赋值是可交换的
- 检查约束然后继续：考虑与以前的赋值不冲突的值

改善 BTS —— 接下来应该挑选哪个变量

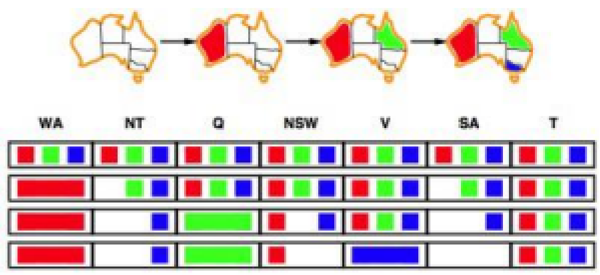
下面是一些启发式的为 BTS 赋值的思路

- **最小剩余价值 Minimum Remaining Value**
  - 选择一个拥有最少合法的取值的变量



- **最小约束值 Least constraining value**
  - 选择一个拥有最少的约束的变量

- **前向检查 Forward checking**
  - 跟踪未分配变量的剩余变量，当某个仍未赋值的变量没有合法的取值时，终止



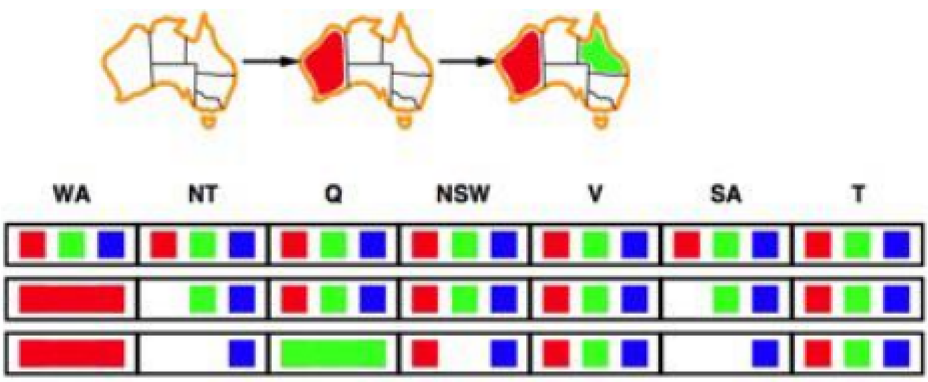
回溯搜索伪代码

```
function BACKTRACKING_SEARCH(csp) returns a solution, or failure
    return BACKTRACK({},csp)

function BACKTRACK(assignment, csp) returns a solution, or failure
    if assignment is complete then return assignment
    val = SELECT_UNASSIGNED_VARIABLES(csp)
    for each value in ORDER_DOMAIN_VALUS(var, assignment, csp)
        if value is consistent with assignment then
            add {var = value} to assignment
            result = BACKTRACK(assignment, csp)
            if result ≠ failure then return result
        remove {var = value} from assignment
    return failure
```

约束传播 Constraint propagation

- 前向检查将信息从分配的变量传播到未分配的变量



- 前向检查不检查未分配变量之间的交互
- 这里是 SA 和 NT（它们都必须是蓝色，但不能同时是蓝色）
- 前向检查改进了回溯搜索，但在未来不会看得太远，因此不能检测到所有的失败

一致性的类型 Type of Consistency

节点一致性 Node-consistency（一元约束）

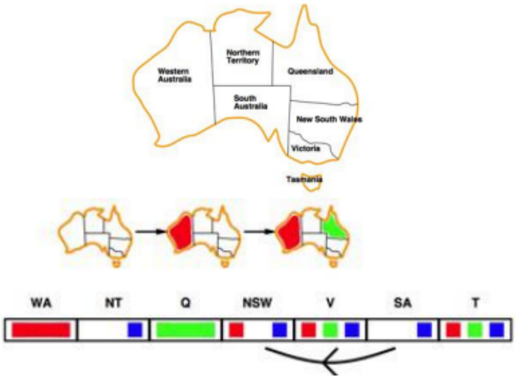
变量 X_i 是节点一致性的如果它的取值范围满足所有的一元约束

弧一致性 Arc-consistency（二元约束）

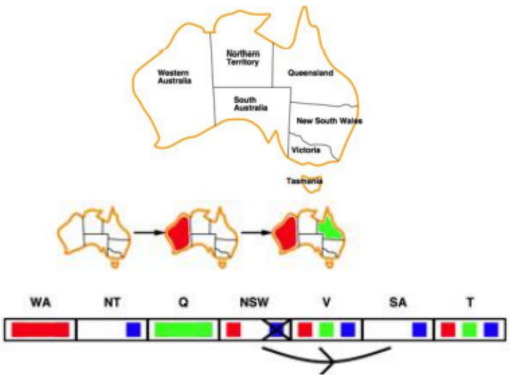
通常，CSP 求解器被设计成处理二元约束

$X \rightarrow Y$ 是弧一致性的当且仅当所有的 X 的取值 x 与一些 Y 的取值 y 是一致的

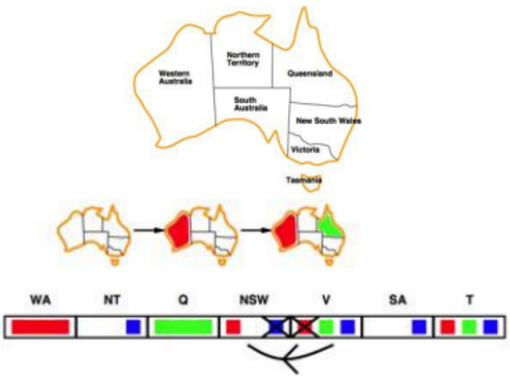
示例：在下面的填色情况中，WA 被分配成了红色，Q 被分配成了绿色，检查约束是否满足



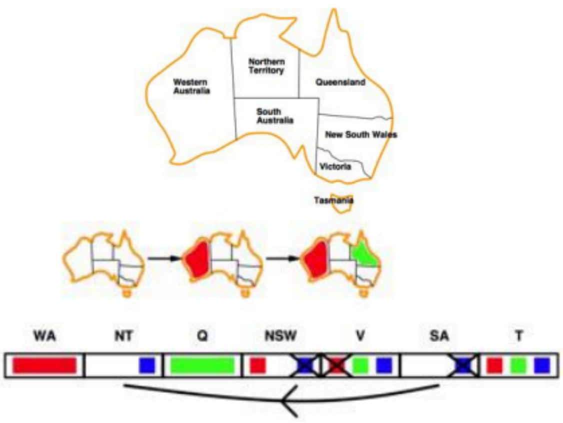
- SA 取蓝色，NSW 取红色，满足条件



- NSW 只能取红色（**删除蓝色选项**），SA 只能取蓝色，满足条件



- NSW 只能取红色，V 不能取红色（**删除红色选项**）



- NT 只能取蓝色，SA 不能取蓝色（**删掉蓝色选项**）

我们发现 SA 没有可选的选项了，但是它还没有被涂色，所以该方案不可以，返回 False

路径一致性 Path-consistency（多元约束）

将弧一致性从二元约束推广到多个约束

可以将所有的多元约束转换为二元约束

弧一致性的检查算法 AC-3

弧一致性的问题可以用如下的算法解决

AC-3

function AC-3(csp)
return **False** if an inconsistency is found, **True** otherwise
inputs: csp, a binary CSB with components(X, D, C)

1. create a **queue**, which contains all arcs
2. **while** queue is not empty **do**
3. (X_i, X_j) = Remove-First(queue);
4. **if** Revise(csp, X_i, X_j) **then**
5. **if** size of $D_i = 0$ **then** **return** False;
6. **for each** X_k in $X_i.Neighbours - \{X_j\}$ **do**
7. add(X_k, X_i) to queue; // 更新后要重新检查一次约束
8. **return** true;

function Revise(csp, X_i, X_j)
return True iff we revise the domain of X_i

1. revised = False
2. **for each** x in D_i **do**
3. **if** no value y in D_j allows (x, y) to satisfy the constraint between X_i and X_j **then**
4. delete x from D_i
5. revised = True
6. **return** revised

AC-3 的时间复杂度

- 设 n 为变量数量， d 是取值范围大小
- 如果每一个变量都链接了其它的所有变量（两两变量都有二元弧约束），那么我们有 $O(n^2)$ 的约束
- 每一个弧都可能有 d 次会被插入到 queue 中
- 检查一个弧的一致性需要 $O(d^2)$ 时间复杂度
- 所以一共需要的时间复杂度为 $O(n^2d^3)$

修改回溯搜索伪代码

```
function BACKTRACKING_SEARCH(csp) returns a solution, or failure
    return BACKTRACK({},csp)

function BACKTRACK(assignment, csp) returns a solution, or failure
    if assignment is complete then return assignment
    val = SELECT_UNASSIGNED_VARIABLES(csp)
```

```
    for each value in ORDER_DOMAIN_VALUES(var, assignment, csp)
        if value is consistent with assignment then
            add {var = value} to assignment
            inferences = AC-3(cap, var, value) // 使用 AC3 推断
            if inferences ≠ failure then
                add inferences to assignment
                result = BACKTRACK(assignment, csp)
                if result ≠ failure then return result
        remove {var = value} ans inferences ffrom assignment
    return failure
```

4. CSP 问题结构化

- 想法：构建问题的结构来提高搜索的效率
- 示例：在涂色游戏中，Tasmania 是一个独立的问题
- 识别约束图的连通组件
- 研究独立的**子问题**

将一个大问题拆解成子问题

- 假设取值范围为 d，变量个数为 n
- BTS 的时间复杂度为 $O(d^n)$
- 假设我们把大问题拆解成了子问题，每个子问题平均有 c 个变量
- 那么我们有 $\frac{n}{c}$ 个子问题
- 完成所有的子问题的计算需要 $O(\frac{n}{c} \times d^c)$ 的时间复杂度

拆解成子问题可以大大缩短计算时间

- 假设 $n = 80, d = 2$
- 假设我们可以拆成 4 个子问题，每个子问题平均有 $c = 20$ 个变量
- 如果只解决大问题，我们需要 $O(2^{80})$ 的时间
- 如果拆分成子问题，我们只需要 $O(4 \times 2^{20})$ 的时间

有向弧一致性 DAC

当然，将一个问题转化为独立的子问题并不总是可能的

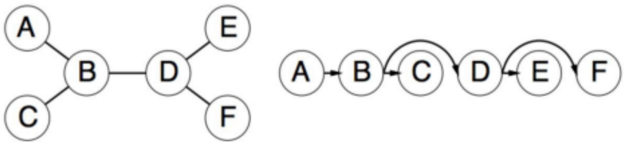
我们能利用其他的图结构吗，当然，如果图是**树状结构**或**接近树状结构**

- 如果任意两个变量只有一条路径相连，那么图就是树

思路：使用 **有向弧一致性 DAC Directed Arc Consistency**

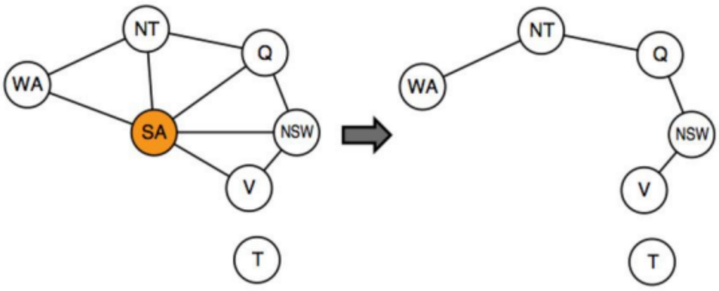
- 一个 CSP 被称为有向弧一致的，如果对于变量的一个排序 $X_1, X_2, ..., X_n$，满足所有的 X_i 都与 X_j 是一致的（且 $j > i$）

树状结构



- 首先选择一个变量作为根（假设是 A）
- 做**拓扑排序**
- 对于 n 个节点，我们有 $n-1$ 条边
- 使得树有向弧一致需要 $O(n)$ 的时间
- 每一次弧一致性的检查需要 $O(d^2)$ 时间
- 整个 CSP 问题可以在 $O(nd^2)$ 的时间复杂度内解决

接近树状结构



- **指定一个变量或一组变量**，并删除所有相邻的边
- 这会将一个约束图剪枝成一个约束树！