中国科学技术大学
University of Science and Technology of China

人工智能讲义

CSP：约束满足问题

March 14, 2022

Outline

地图着色问题

用 k 种色彩对地图着色

- 能否用 k 种色彩对地图染色，一个国家一种颜色，相邻国家颜色不一样
- 四色定理：第一个用计算机证明的数学定理，证明的正文 300 页，“是否存在四色定理的一个简短证明，……使得一个合格的数学家能在（比如说）两个星期里验证其正确性呢？”（摘自汤米 · R · 延森和比雅尼 · 托夫特《图染色问题》）

着色问题思考

- 优化问题？NO! 没有优化的目标函数，仅仅是优化问题的特例：满足优化问题的约束条件即可，是否是最优解，不关心
- 搜索问题？YES! 一个染色过程（序列），不考虑路径耗散，也可以不需要具体路径，给出终态即可，通用搜索问题的特例

CSP 问题

形式化定义 $CSP = (X, D, C)$，是一个三元组，其中：

- X 是一组变量，不妨假设为一个 n 维向量 $X = (X_1, X_2, \ldots, X_n)$
- D 是值域，定义了每一个变量的值域，$D = (D_1, D_2, \ldots, D_n)$
- C 是一个约束组，$C = (C_1, C_2, \ldots, C_m)$，是 m 个计算公式，约束 X 分量之间取值的关系；
- $\forall i, X_i$ 从 D_i 中取值，所有变量都给一个取值，形成一个“赋值” $v = (x_1, x_2, \ldots, x_n)$，或者说是一个“解”，可以参与 $C_1, C_2, \ldots, C_m$ 表示的约束计算公式的计算，判断约束是否被满足。

CSP 的例子

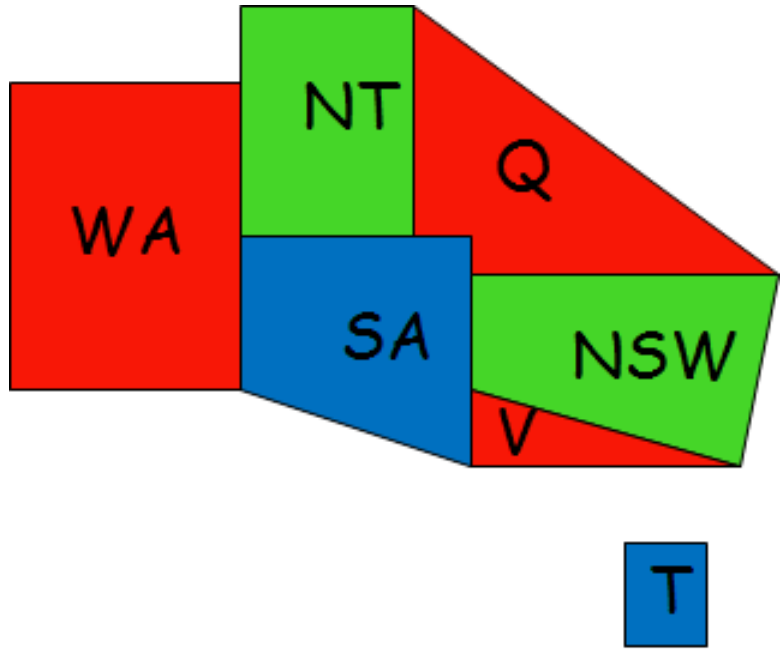
$$\underbrace{(x_1 \vee x_2 \vee x_5) \wedge (\neg x_1 \vee x_3) \ldots\ldots \wedge (x_3 \vee x_n)}_{\text{m个析取式构成一个合取式}}$$

3-SAT

- n 个变量 $x_1, x_2, \ldots, x_n$
- 每个变量的取值域为：$\{0, 1\}$，（另一种说法：变量是布尔变量，此时值域 $\{T, F\}$）
- m 个约束，每个约束是少于等于 3 个变量的析取式，或者将 m 个约束表示为一个合取式约束条件，如上图所示。
- 求一个真值指派 $(x_1, x_2, \ldots, x_n) = (v_1, v_2, \ldots, v_n)$，使得上述每个析取式都被满足（为真），或使得上述合取式为真。

每年的 SAT 竞赛网址：http://www.satcompetition.org/

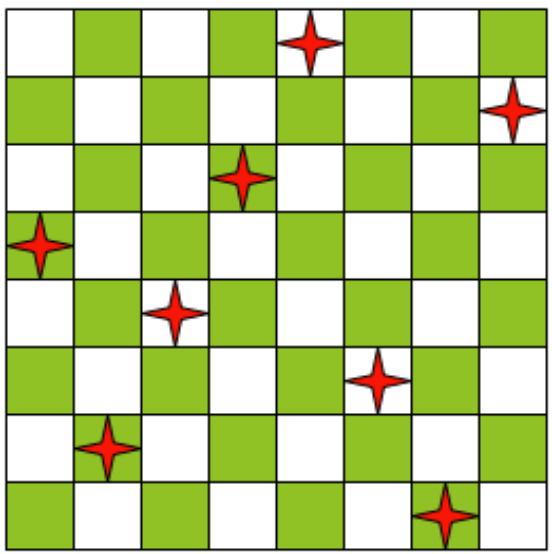
CSP 的例子



地图着色问题，如上图所示

- 7 个变量：$\{WA, NT, SA, Q, NSW, V, T\}$
- 每个变量的取值范围都一样：$\{red, green, blue\}$
- 约束条件：任何两个相邻的变量色彩不一样，即 $WA \neq NT, WA \neq SA, NT \neq SA, NT \neq Q, SA \neq Q, SA \neq NSW, SA \neq V, Q \neq NSW, NSW \neq V$

CSP 的例子



八皇后问题，如图所示

- 8 个变量：$x_1, x_2, \ldots, x_8$
- 每个变量的取值范围:$1, 2, ..., 8$，表示行编号
- 约束条件，包括两类：
  - 若 $x_i = k$, 则 $x_j \neq k, \forall j = 1, 2, ..., 8, j \neq i$, 第 k 行只能放一个皇后
  - 每个对角线只能放置一个皇后，即任意两个皇后 $\forall i, j, x_i = k, x_j = l$, 满足约束 $\frac{l-k}{i-j} \neq \pm 1$

CSP 的例子

N_i = {English, Spaniard, Japanese, Italian, Norwegian}
C_i = {Red, Green, White, Yellow, Blue}
D_i = {Tea, Coffee, Milk, Fruit-juice, Water}
J_i = {Painter, Sculptor, Diplomat, Violinist, Doctor}
A_i = {Dog, Snails, Fox, Horse, Zebra}

推理问题

- 5 个变量表示 5 个人，每个人有 5 种属性，每种属性的取值域如图
- 约束条件如下：
  - The Englishman lives in the Red house. $\rightarrow N_i = English \Leftrightarrow C_i = red$
  - The Spaniard has a Dog. $\rightarrow N_i = Spaniard \Leftrightarrow A_i = Dog$
  - The Japanese is a Painter. $\rightarrow N_i = Japanese \Leftrightarrow J_i = Painter$
  - The Italian drinks Tea. $\rightarrow \ldots$
  - The Norwegian lives in the first house on the left. $\rightarrow \ldots$
  - The owner of the Green house drinks Coffee.
  - The Green house is on the right of the White house.
  - The Sculptor breeds Snails.
  - The Diplomat lives in the Yellow house.
  - The owner of the middle house drinks Milk.
  - The Norwegian lives next door to the Blue house.
  - The Violinist drinks Fruit juice.
  - The Fox is in the house next to the Doctor's.
  - The Horse is next to the Diplomat's.

谁有斑马？

CSP 问题分类

有限和无限 CSP：解的数目是否是有限的

- 判断的方法：变量的取值域是有限还是无限？
- 一旦有一个变量的取值范围是无限的，则 CSP 问题就是无限的（此时解的个数是无限的）
- 若每个变量的取值个数是有限的，则得到有限 CSP 问题，典型问题是 SAT

将 CSP 问题形式化为搜索问题

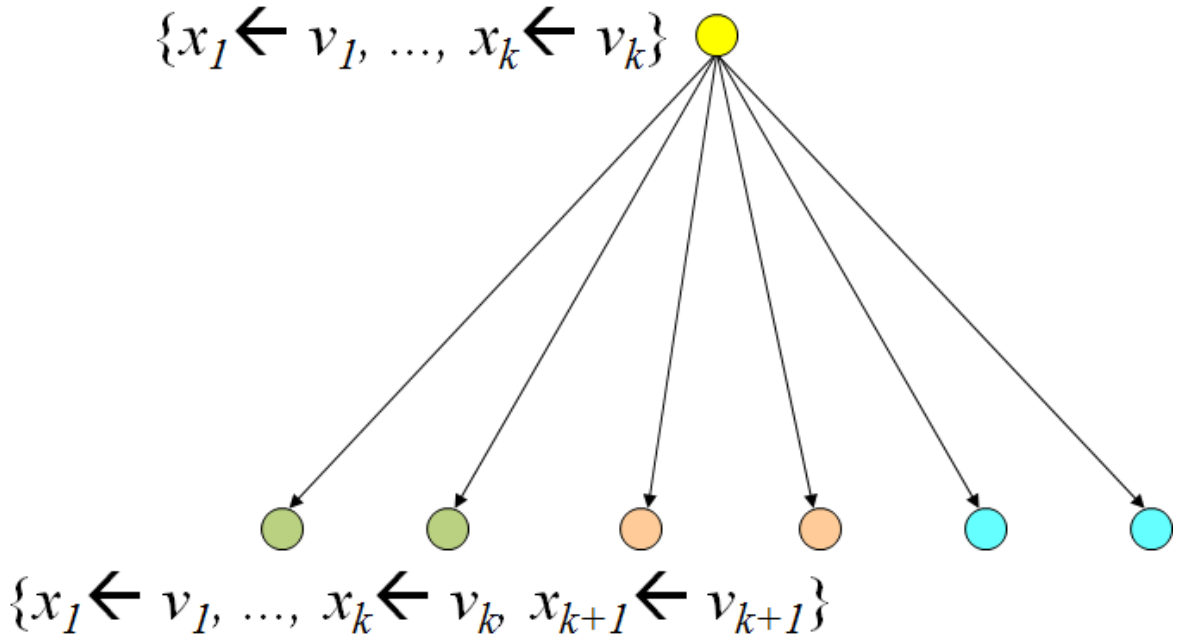
符号和概念定义

- 有效赋值：对 n 个变量 $x_1, x_2, \ldots, x_n$，一个一个地赋值，不妨设前 k 个赋值为：$x_1 \leftarrow v_1, x_2 \leftarrow v_2, \ldots, x_k \leftarrow v_k$，此时的赋值，会使前 k 变量相关的约束条件都满足；
- 完全赋值：$k = n$，也就是 n 个变量都被赋值，使得所有的约束条件都得到了满足。

系统化方法

- 状态/状态空间：有效赋值/所有可能有效赋值
- 初始状态：$\{\}, k = 0$
- 后继函数：$\{x_1 \leftarrow v_1, x_2 \leftarrow v_2, \ldots, x_k \leftarrow v_k\} \rightarrow \{x_1 \leftarrow v_1, x_2 \leftarrow v_2, \ldots, x_k \leftarrow v_k, x_{k+1} \leftarrow v_{k+1}\}$
- 目标测试：$k = n$
- 路径耗散：假设单步路径耗散为 1

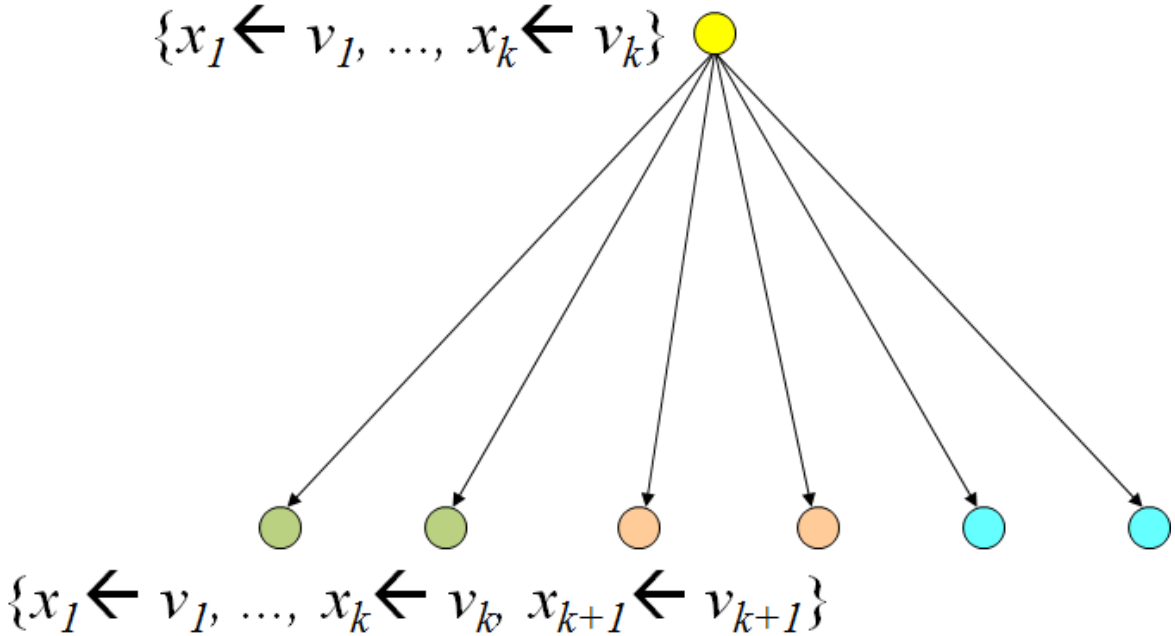
CSP 问题后继函数



图解后继函数，如上图

- 剩下 $r = n - k$ 个变量，不妨设每个变量的取值个数相同，都是 s，则有 $b = s \times r$ 个后继，即分支因子是 b

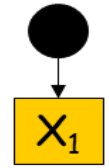
CSP 问题分析

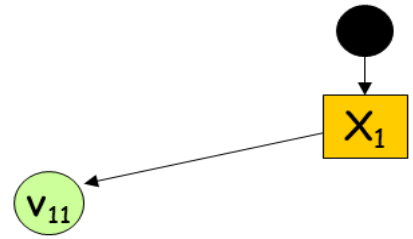


特点/性质：可交换性

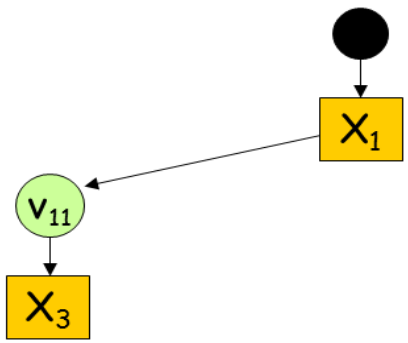
- 变量的赋值次序和完全赋值的可达性无关，CSP 的关键属性之一
- 带来的好处之一：扩展节点的时候，任意选择一个未赋值的变量，考查其所有可能的赋值即可（扩展节点），后继节点的数目 $s \times r \to s$
- 带来的好处之二：不需要存储到达当前节点（已有有效赋值）的路径，所以可用回溯算法（使用递归的简化深度优先算法）

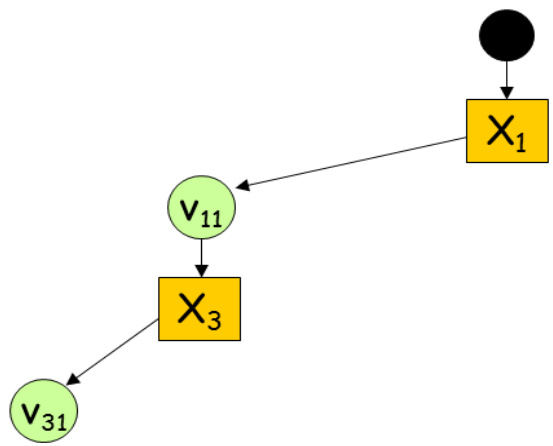
求解 CSP 问题：例子



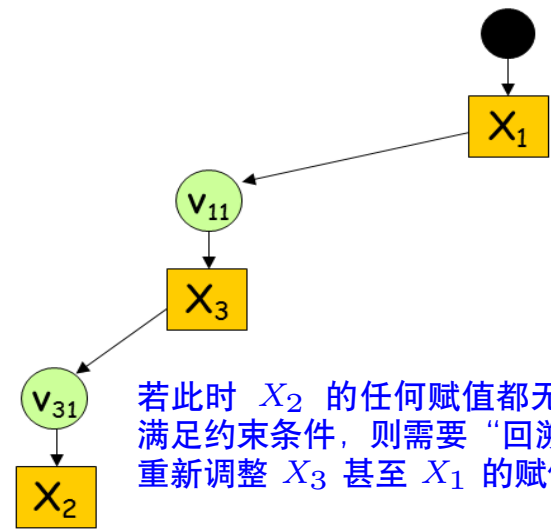


$X_1 \leftarrow v_{11}$



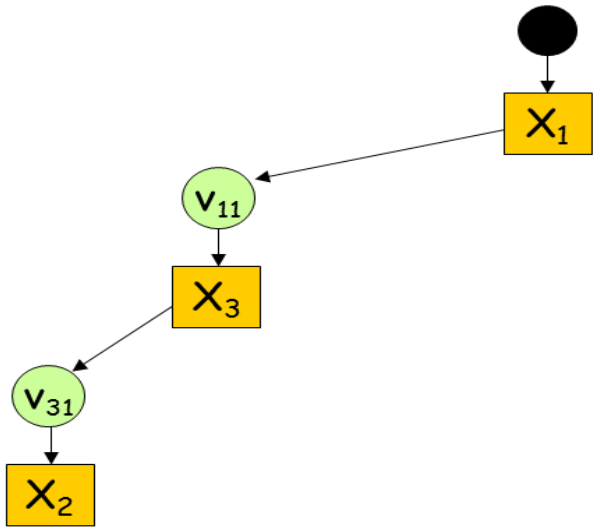


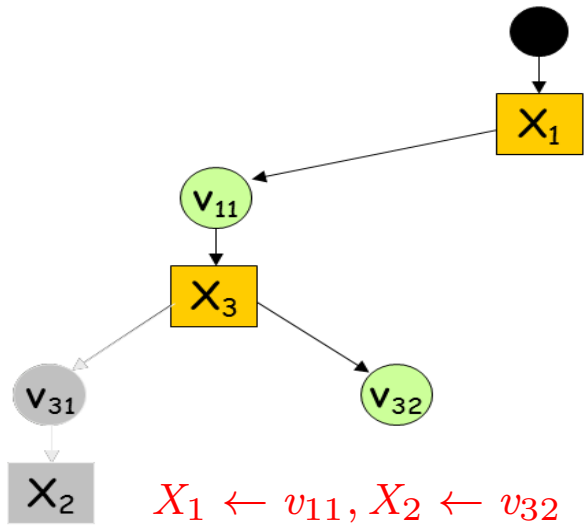
$X_1 \leftarrow v_{11}, X_3 \leftarrow v_{31}$



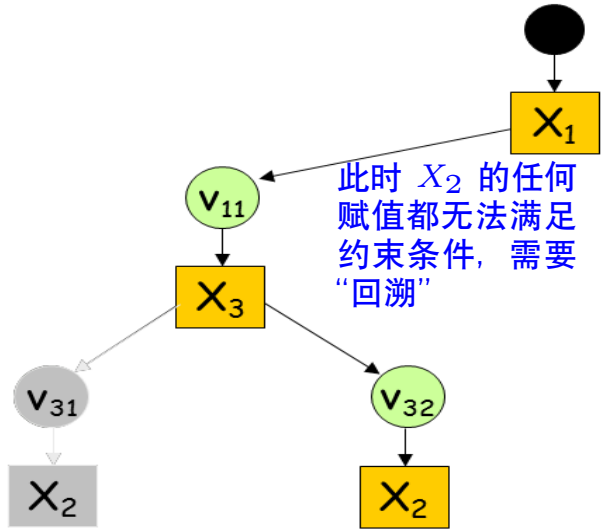
若此时 X_2 的任何赋值都无法满足约束条件，则需要“回溯”，重新调整 X_3 甚至 X_1 的赋值

求解 CSP 问题：例子

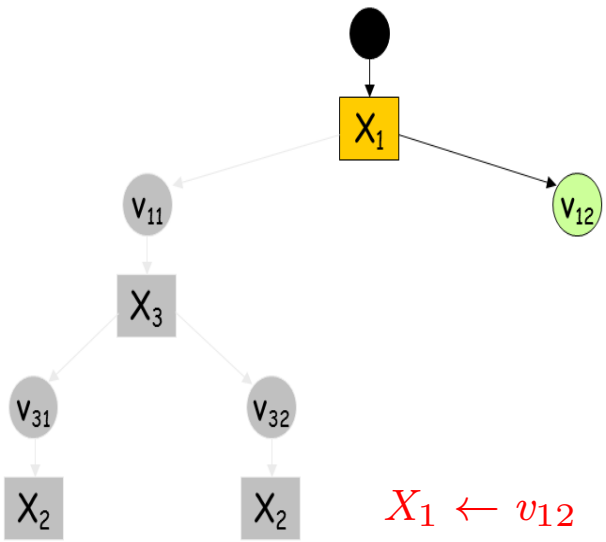




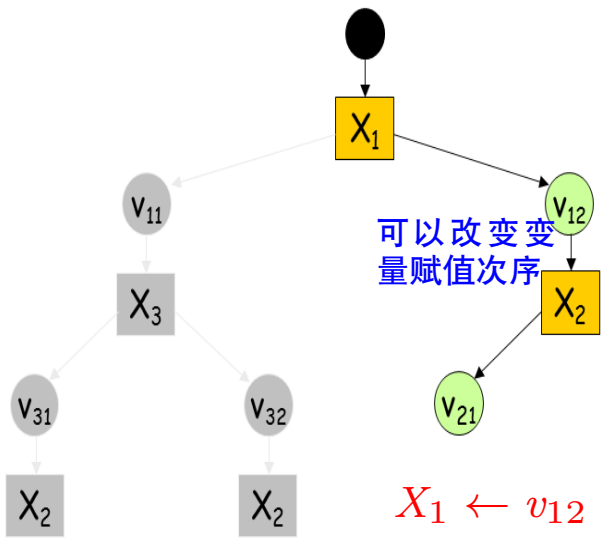
$X_1 \leftarrow v_{11}, X_2 \leftarrow v_{32}$



此时 X_2 的任何赋值都无法满足约束条件，需要“回溯”

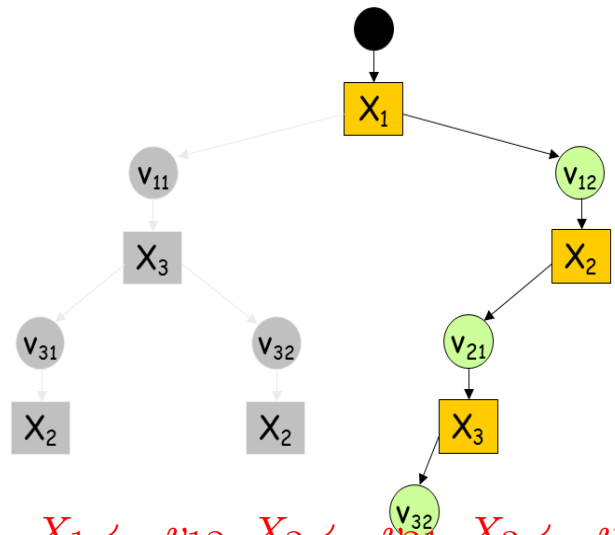


$X_1 \leftarrow v_{12}$



可以改变变量赋值次序

$X_1 \leftarrow v_{12}$
$X_2 \leftarrow v_{21}$



$X_1 \leftarrow v_{12}, X_2 \leftarrow v_{21}, X_3 \leftarrow v_{31}$

回溯算法

算法框架：CSP-BACKTRACKING(A)

Input: 约束条件; 有效赋值 A
Output: A: 完全有效赋值
if 有效赋值 A 是完全的 **then**
    return A
end
$X \leftarrow$ 选择一个未赋值变量;
$D \leftarrow$ 选择一个 X 所有可能取值的次序，例如随机次序;
foreach v *in* D **do**
    $Add(X \leftarrow v)$ to A;
    if A 是有效的 **then**
        $result \leftarrow$CSP-BACKTRACKING(A) /* 递归调用 */;
        if $result \neq failure$ **then**
            return $result$;
        end
    end
    $Remove(X \leftarrow v)$ from A;
end
return $failure$;

算法启动：CSP-BACKTRACKING({})

回溯算法

CSP-BACKTRACKING(A): 减少回溯，提高算法效率

Input: 约束条件; 有效赋值 A
Output: A: 完全有效赋值
if 有效赋值 A 是完全的 **then**
    return A
end
$X \leftarrow$ 选择一个未赋值变量;
$D \leftarrow$ 选择一个 X 所有可能取值的次序，例如随机次序;
foreach v *in* D **do**
    $Add(X \leftarrow v)$ to A;
    if A 是有效的 **then**
        $result \leftarrow$CSP-BACKTRACKING(A) /* 递归调用 */;
        if $result \neq failure$ **then**
            return $result$;
        end
    end
    $Remove(X \leftarrow v)$ from A;
end
return $failure$;

关键之处

改进回溯法

下一步选择哪个变量进行赋值？

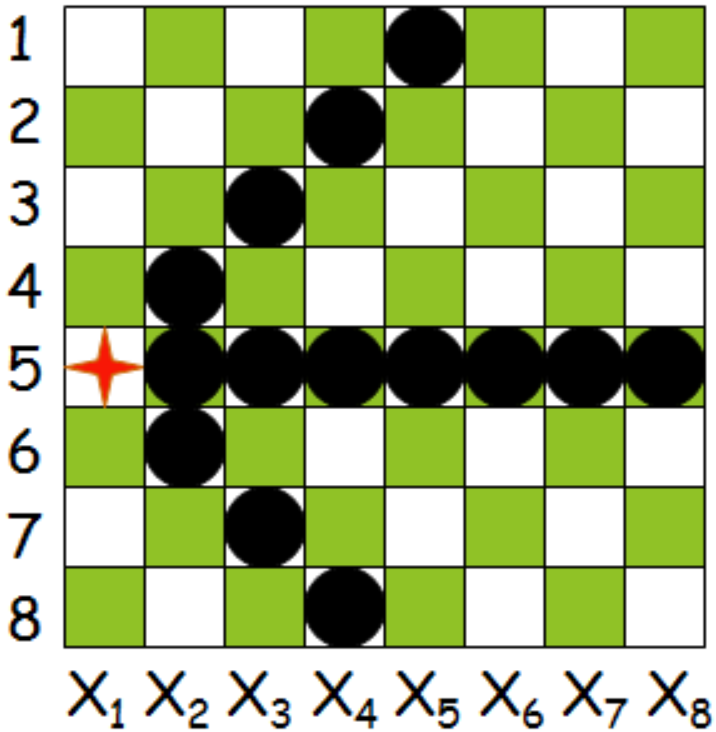
- 为什么会回溯？当前的赋值不一定会最终得到解，继续赋值下去，可能会发现后续变量的赋值总会和以前的某些赋值一起违背某些约束，我们称之为“冲突”，“冲突”导致回溯的产生
- 选择一个变量，让冲突更早来到！可以减少回溯

待赋值变量 x 的 s 个取值，赋值给 x 的次序是什么？

- 理想状态，每次给每个变量的赋值都是“最佳的”，那么就不会有回溯出现，快速地获得了解

具体的技术方法？

前向检验技术

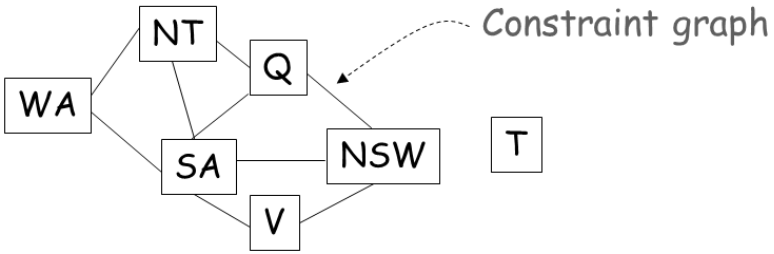


检测冲突的方法

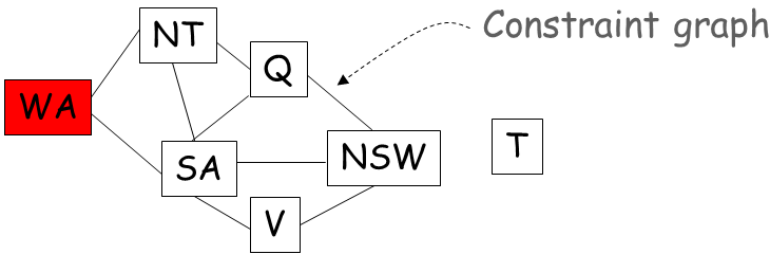
前向检验：以 8 皇后问题为例

- 假设赋值 $x_1 = 5$，导致图中涂黑圆的格子成为其它变量 $x_1, \ldots, x_8$ 等的取值“禁区”，可以将这些格子从这些变量的取值域中“丢弃”；
- 发展这个思想，每增加一个变量的赋值，引发了某些未赋值变量取值范围的缩小（要求满足约束条件而造成的），确定这些变量缩小的变量取值范围，一旦某个变量的取值范围为空集 $\{\}$，则发现了“冲突”，这就是“前向检验”

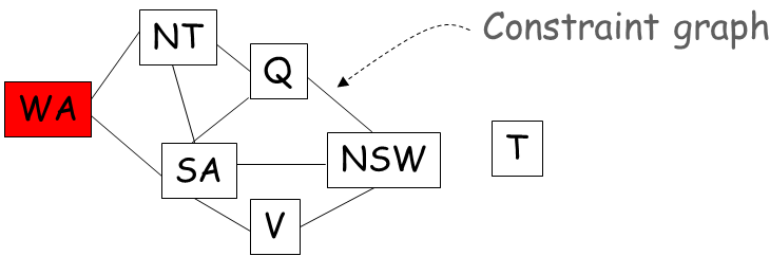
前向检验技术：地图着色的例子



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB

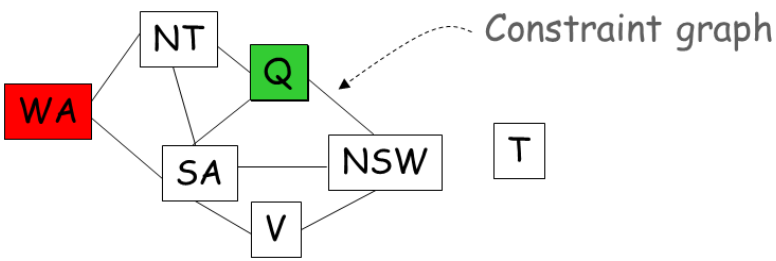


WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	RGB	RGB	RGB	RGB	RGB	RGB



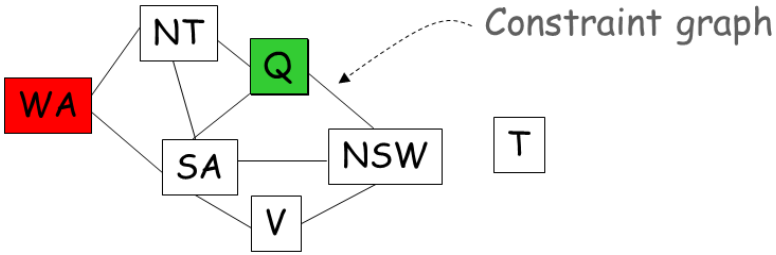
WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	~~R~~GB	RGB	RGB	RGB	~~R~~GB	RGB

前向检验将把红色从NT 和SA的取值域中移除

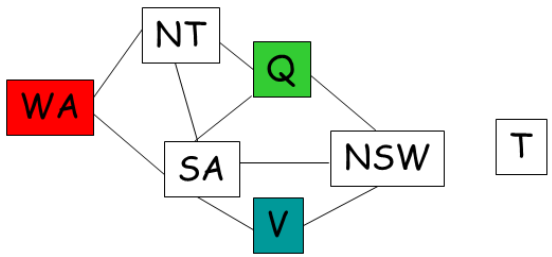


WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	GB	RGB	RGB	RGB	GB	RGB
R	GB	G	RGB	RGB	GB	RGB

前向检验技术：地图着色的例子



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	GB	RGB	RGB	RGB	GB	RGB
R	~~G~~B	G	R~~G~~B	RGB	~~G~~B	RGB



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	GB	RGB	RGB	RGB	GB	RGB
R	B	G	RB	RGB	B	RGB
R	B	G	R~~B~~	B	~~B~~	RGB

产生“冲突”

- SA 的值域为空集：当前赋值 $\{(WA \leftarrow R), (Q \leftarrow G), (V \leftarrow B)\}$ 将不会出现在最终解中

前向检验技术：算法描述

当一个变量赋值 $x \leftarrow v$ 加入到 A 之后，执行下述操作：

forward-checking($D_{\bar{A}}, x, v, A$)

- for each 变量 y not in A do:
  - for each 约束 c in C do:
    - if c 和 y 不相关 then continue;
    - for each u in D_y do:
      if $A \cup \{y\} \leftarrow u$ 使得 c 不被满足 then $remove(u, D_y)$

解释说明

- A 已经赋值的变量集合
- C 约束集合
- D_y 变量 y 的值域,$D_{\bar{A}}$ 不在 A 中的变量的取值域
- $remove(u, D_y)$ 表示从 y 的值域 D_y 中删除 u

改进的回溯算法 1

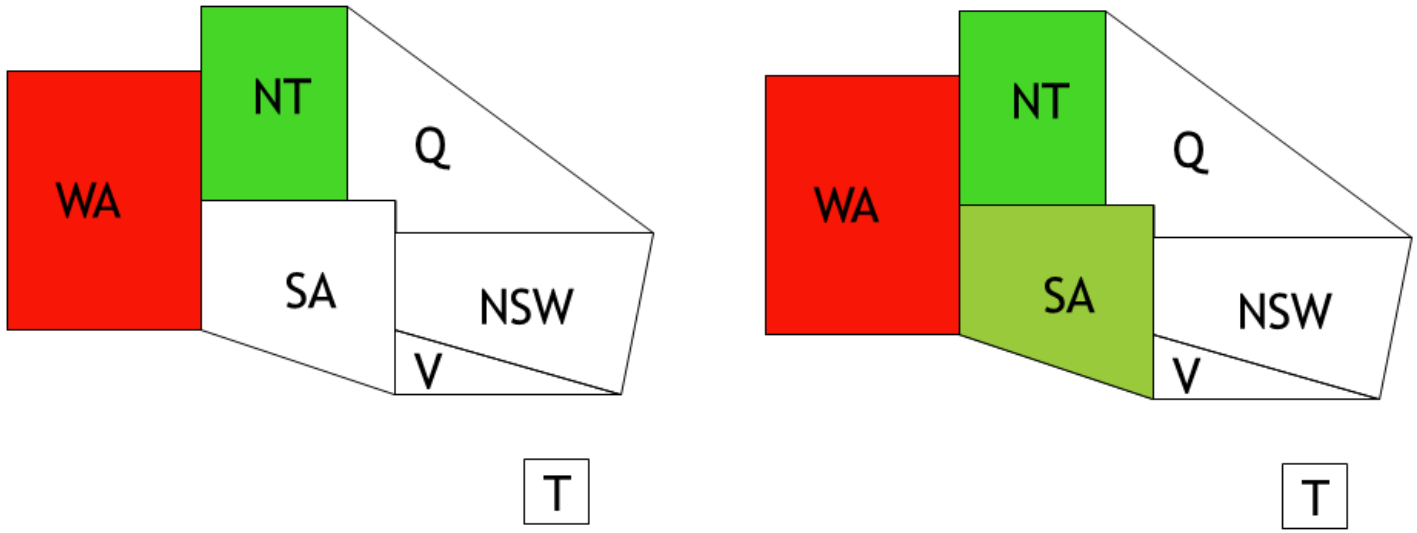
改进的回溯算法
CSP-BACKTRACKING($A, D_{\bar{A}}$)

Input: 约束条件; 有效赋值 A
Output: A: 完全有效赋值
if 有效赋值 A 是完全的 **then**
  return A
end
$X \leftarrow$ 选择一个未赋值变量;
$D_X \leftarrow$ 选择一个 X 所有可能取值的次序，例如随机次序;
foreach v *in* D_X **do**
  $Add(X \leftarrow v)$ to A;
  新值域 $D_{\bar{A}} \leftarrow forward - checking(D_{\bar{A}}, X, v, A)$;
  if $D_{\bar{A}}$ 的每个变量的值域都非空 **then**
    $result \leftarrow$CSP-BACKTRACKING($A, D_{\bar{A}}$) /* 递归调用 */;
    if $result \neq failure$ **then**
      return $result$;
    end
  end
  $Remove(X \leftarrow v)$ from A;
end
return $failure$;

提高回溯算法效率 2

下一步选择哪个变量进行赋值？

- 启发式函数：选择约束最强的变量/残留值域最小的变量
- 理由：分支因子最小，后继最少！



例子如图所示

- SA 的残留值域大小为 1，Q 的残留值域大小为 2，其它三个的残留值域大小都是 3
- 选择 SA 上色

提高回溯算法效率 2

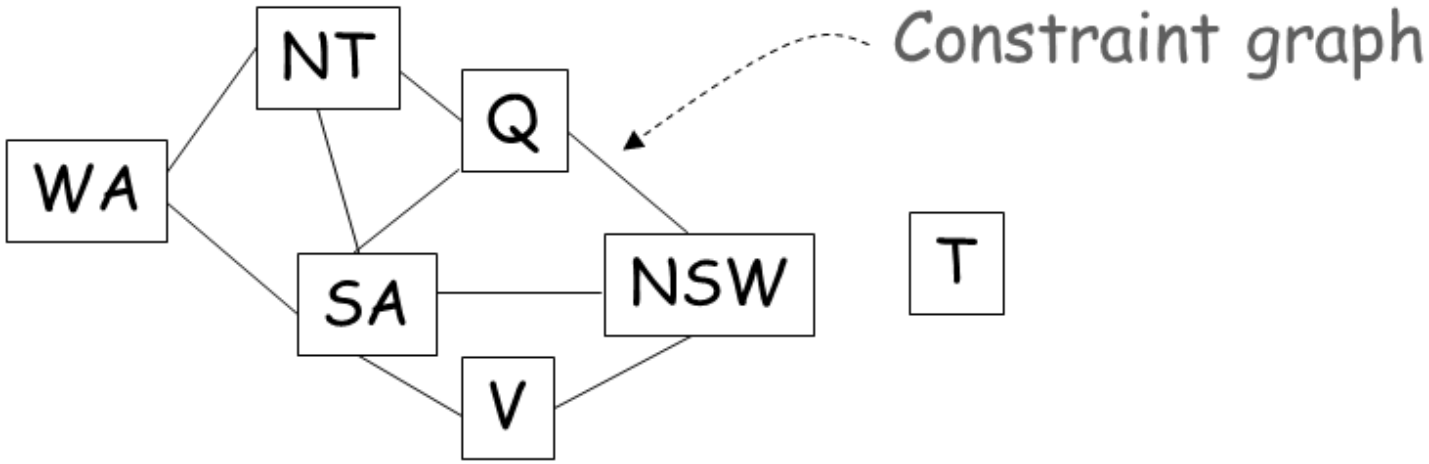
下一步选择哪个变量进行赋值？

- 启发式函数：选择约束最强的变量/残留值域最小的变量
- 理由：分支因子最小，后继最少！

存在问题

- 很多个变量，残留值域大小相同，怎么办？如何选择？
- 新的启发式函数 2：找到一个变量，在未满足约束组（还有变量未被赋值，所以无法判断是否满足）中出现的次数最多，称为“最多被约束着的变量”
- 理由：使将来残留值域缩减得更多！降低分支因子

提高回溯算法效率 2



例子，如上图所示

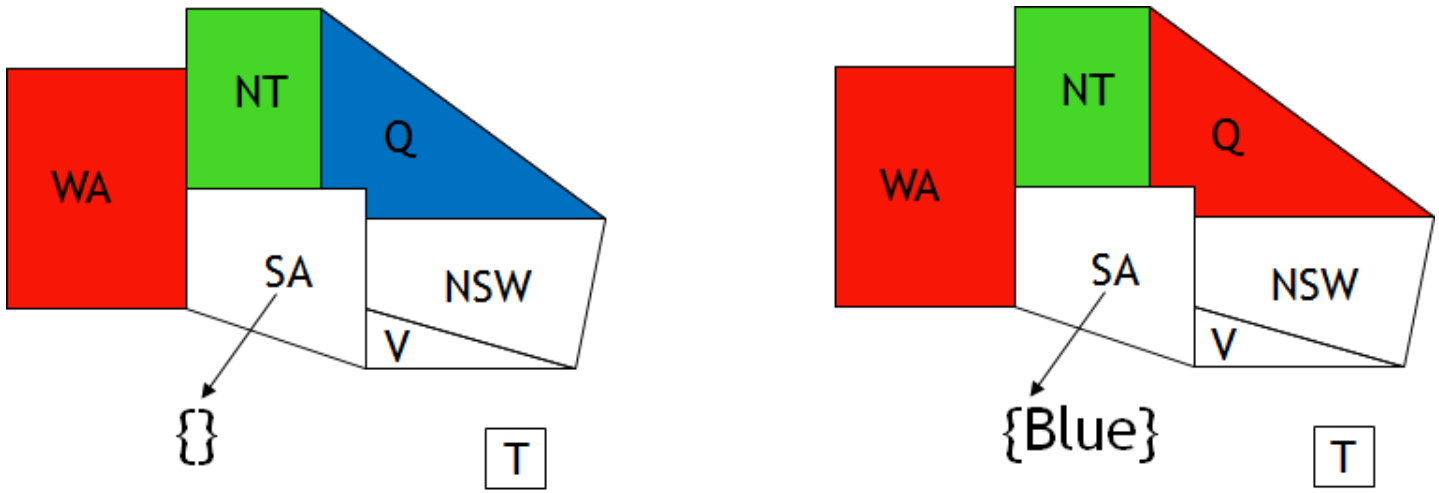
- 启发式函数 1 和启发式函数 2 复合成完整的启发式函数，实现对“下一个变量”的选择第一个变量赋值之前，所有变量的残留值域大小都是 3，选谁？
- SA 同时出现在 5 个未被满足的约束条件中，故选择之赋值。

提高回溯算法效率 3

变量 x 选择出来了，如何给其残留的值域排序？

- 启发式函数 3：选择值 $v \to x$, 满足使得从未赋值的变量的残留值域中删除的值最少。
- 理由：这个 v，被称为“最少被约束着的值”，也就是说这个赋值对未来的影响最小，以防将来没办法给其它变量找到赋值
- 具体做法：对每个不同的 v，都需要进行前向检验，然后检查前向检验的结果，看哪个 v 值对残留值域大小的影响最小！（操作比较复杂，费时间）

提高回溯算法效率 3



地图着色的例子，如上图所示

- 此时，假设 Q 被选出，准备执行上色，Q 的残留值域大小为 2，红色或蓝色
- 若 Q 上蓝色，则 SA 的残留值域大小为 0；
- 若 Q 上红色，则 SA 的残留值域大小为 1，故选择 SA 残留值域最大的着色方式

改进的回溯算法 3

改进的回溯算法

CSP-BACKTRACKING($A, D_{\bar{A}}$)

Input: 约束条件; 有效赋值 A
Output: A: 完全有效赋值
if 有效赋值 A 是完全的 **then**
  return A
end
$X \leftarrow$ 选择一个未赋值变量; $\Longrightarrow$ 用启发式函数 1 和启发式函数 2 来代替
$D_X \leftarrow$ 选择一个 X 所有可能取值的次序，例如随机次序; $\Longrightarrow$ 用启发式函数 3 来代替
foreach v *in* D_X **do**
  $Add(X \leftarrow v)$ to A;
  新值域 $D_{\bar{A}} \leftarrow forward - checking(D_{\bar{A}}, X, v, A)$;
  if $D_{\bar{A}}$ 的每个变量的值域都非空 **then**
    $result \leftarrow$CSP-BACKTRACKING($A, D_{\bar{A}}$) /* 递归调用 */;
    if $result \neq failure$ **then**
      return $result$;
    end
  end
  $Remove(X \leftarrow v)$ from A;
end
return $failure$;

提高回溯算法效率 4

约束传播: 更好的冲突检测机制或方法

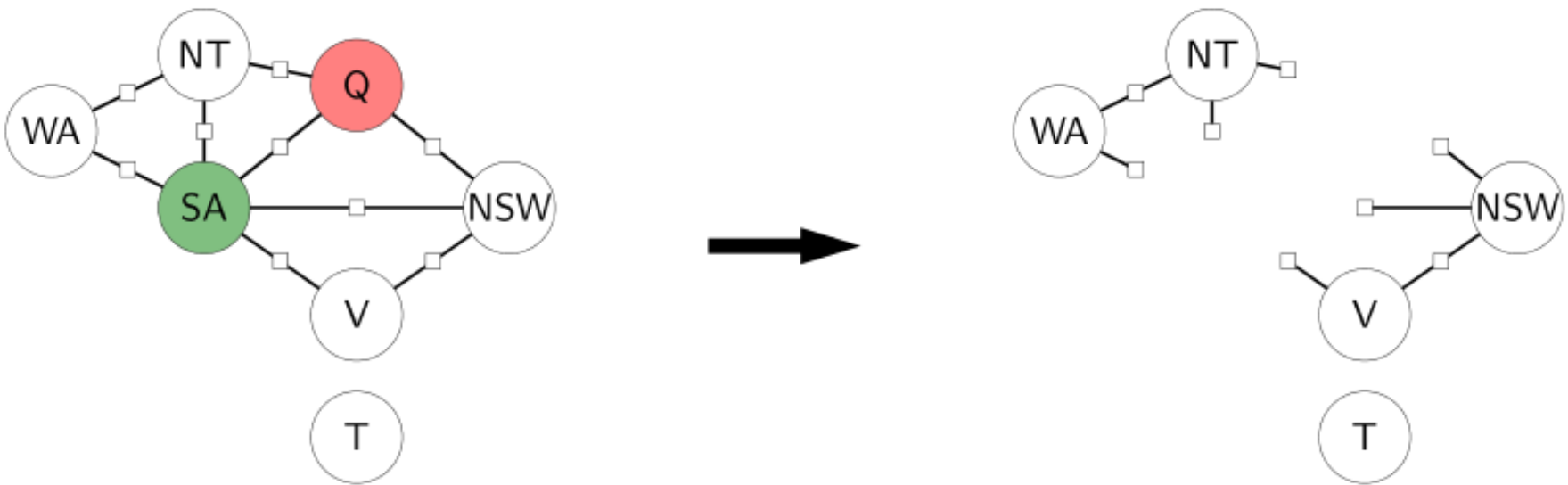
- 一个变量 x 的赋值后，其它未赋值变量中一个变量的赋值可能会影响另一个未赋值变量的残留值域，找到这种变化，实现“约束传播”!
- 相容的概念：一个变量 x 的任何一个赋值，另一个变量 y 至少存在一个赋值，这些赋值不违背任何约束，则称 x, y 是 2 相容的
- 二元约束的传播算法：$AC3$

约束传播方面内容，感兴趣同学课后阅读。

进一步加速 csp 求解算法：约束图

每个变量是一个节点，变量间连边，构成图

- 连边规则：两个变量参与某个约束就连边。“边”意味着边两端的变量在赋值时需要考虑另一个变量的取值。
- 某些顶点/变量赋值后，这些顶点间的连边参与的约束条件被满足，可以移除这些边，图有可能变成不连通的子图；对子图递归求解。



问题：划分子图的方式太多，如何选择，使得每个子图都很容易求解？

CSP 的应用

一些例子

- 教务处排课
- 电路布局设计
- 任务调度
- ...

优化问题 $\Longrightarrow$CSP 问题

- 未知最优解的最优化问题，不知道什么时候搜索停止；
- 实际工程中，往往可以设置一个可接受的解的质量水平，将这个解质量看成是一个约束条件，最优化问题的优化目标就转化成了约束条件，得到了一个 CSP