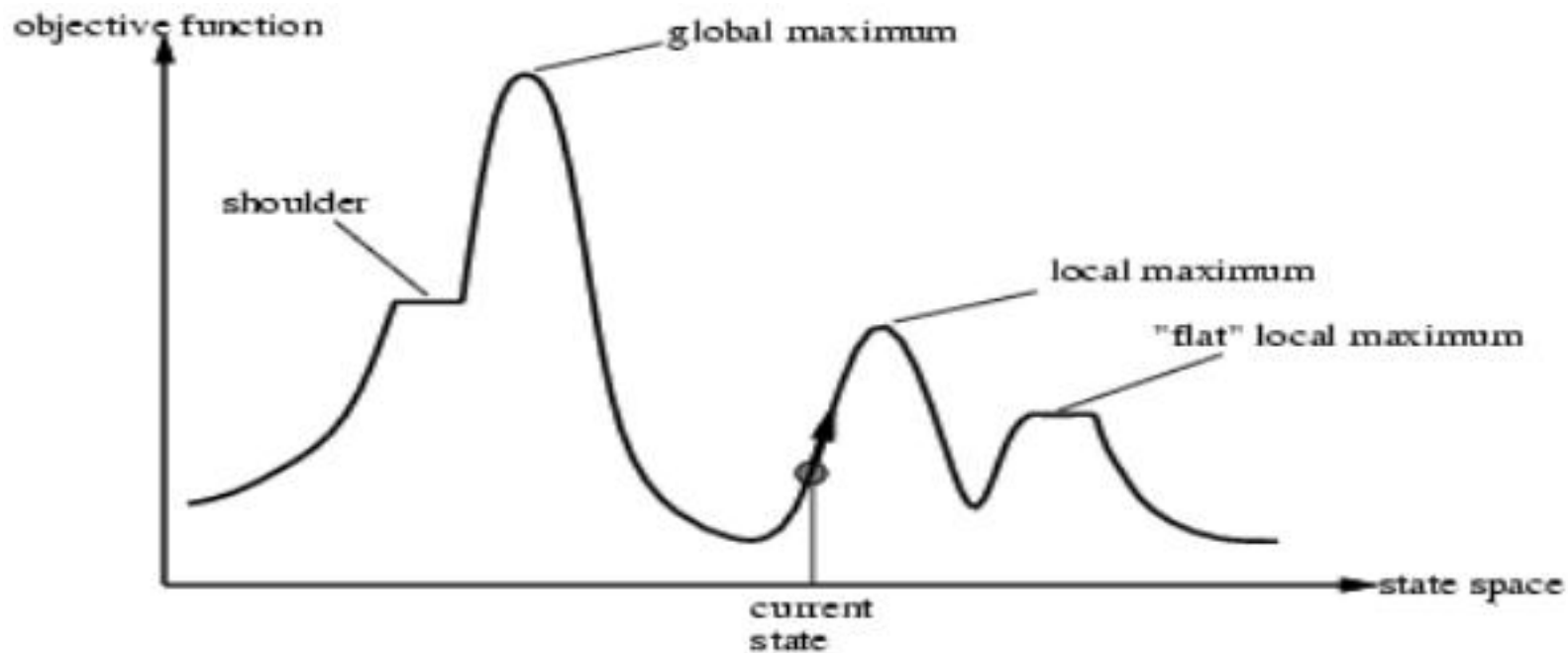


## 爬山搜索

- 几个概念：目标函数、全局最优、局部最优、山肩、平原.....



- 依赖于初始状态，容易陷于局部最优。

## 爬山法的弱点:

如下三种情况经常被困:

- 局部极大值 (Local maxima)

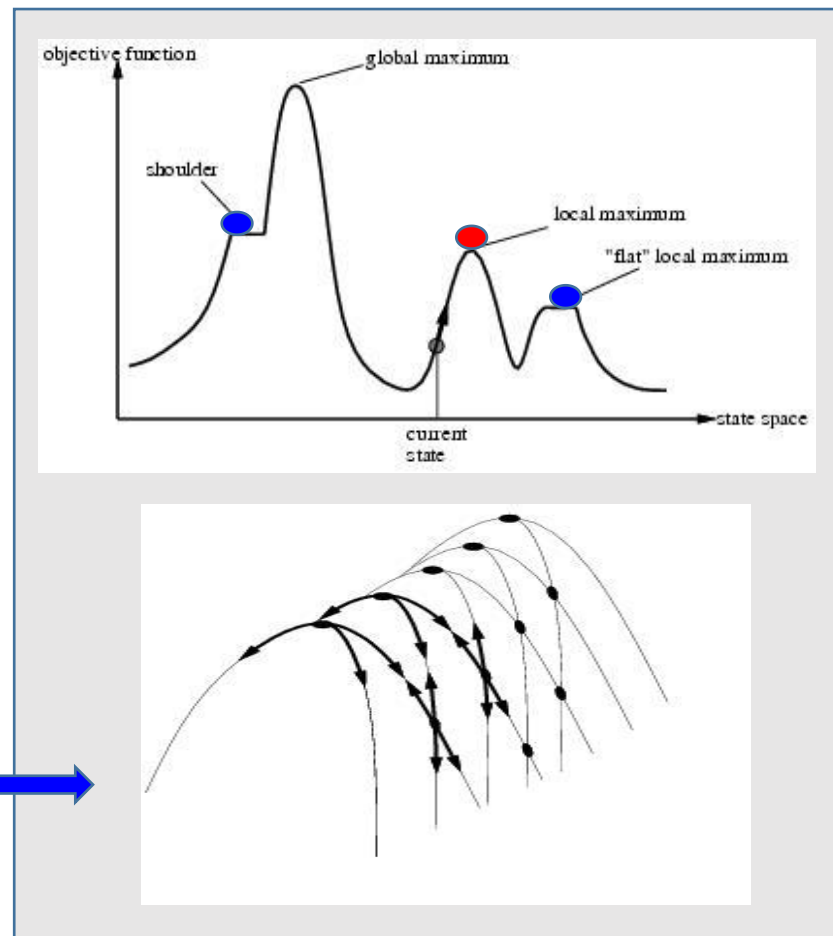
高于相邻节点但低于全局最大值。

- 高原 (Plateaux) :

平坦的局部最大值，或山肩。即一块状态空间区域的评价函数值是相同的。

- 山脊 (Ridges):

一系列连续的局部极大值，对于贪婪算法去引导的搜索是困难的。



## 爬山法变种 (Variants of Hill Climbing) :

- 随机爬山法

- 随机沿上坡选取下一步。
- 选取的可能性随山坡的陡峭程度变化移动。与最陡爬坡相比，收敛速度通常较慢。

- 首选爬山法

- 随机产生后继节点直到有优于当前节点的后继节点出现。

以上二种：试着避开局部最大。

- 随机重启开始爬山法

随机生成的初始状态直到找到目标。它十分完备，重新开始

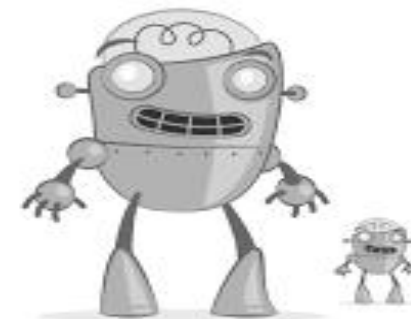
*It adopts the well-known adage: "If at first you do not succeed, try, try again."* 。 **这个算法完备率接近1。**

- 侧向移动：采用限制次数的方法限制侧移次数，避免死循环。

改进后的8皇后问题，侧移设置为100次成功率由14%上升到94%。

## 2.2局部束搜索

- 在内存中仅保存一个节点似乎是对内存限制问题的极端反应。
- **局部束搜索**：保存 $k$ 状态，而不是一个状态
  - 从 $k$ 个随机产生的状态开始
  - 每次迭代,产生 $k$ 个状态的所有后续
  - 如果任何一个新产生的状态是目标状态，则停止。否则从所有后续中选择 $k$ 个最好的后续，重复迭代。



**//同随机重新开始的区分：在 $k$ 条线程中共享信息。**

**缺点：**局部束搜索会很快地集中在状态空间的某个小区域内，使得搜索代价比爬山法还要昂贵。**缺乏多样性。**

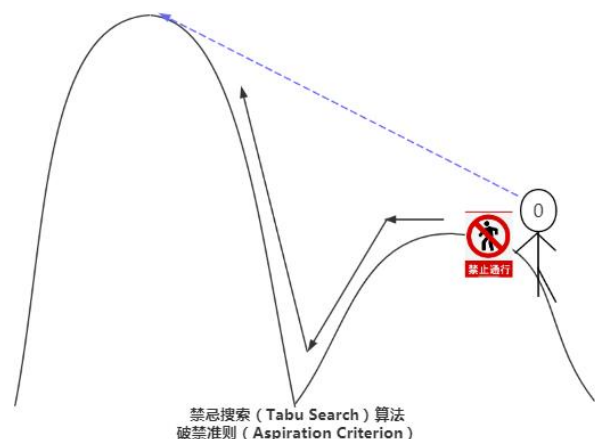
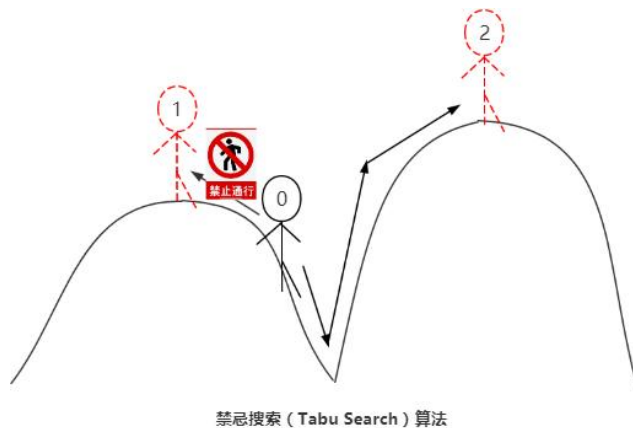
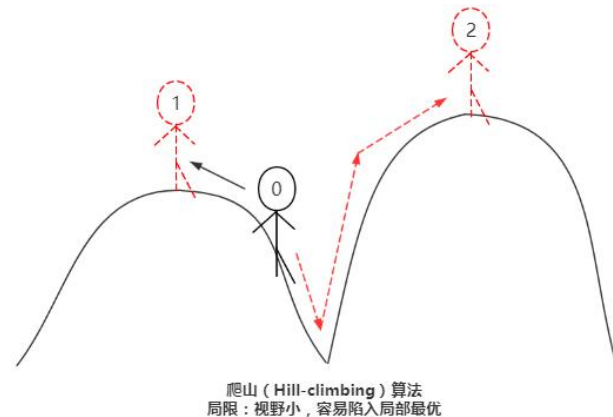
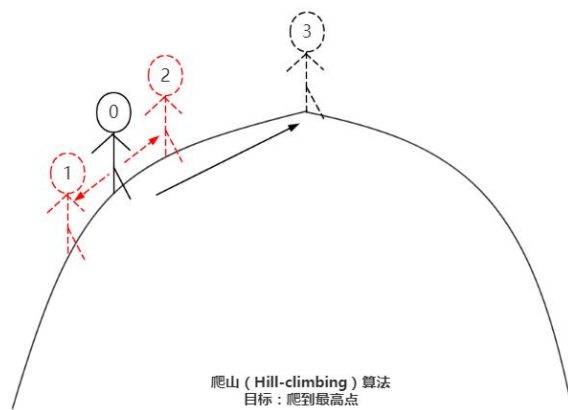
- **改进的随机变种**：它不是选择 $k$ 个最佳后继节点，而是以选择后继节点的概率是其值的递增函数，来随机地选择 $k$ 个后继节点。（随机束搜索有些类似于自然选择的过程）

## 2.3 禁忌搜索

### Tabu Search Algorithm 禁忌搜索算法思想

**禁忌搜索 (Tabu Search, TS)** 模仿人类的记忆功能，在搜索过程中标记已经找到的局部最优解及求解过程，并于之后的搜索中避开它们；

算法通过禁忌策略实现记忆功能，通过特赦（破禁）准则继承局部搜索的强局部搜索能力。使得TS一方面具备高局部搜索能力，同时又能防止算法在优化中陷入局部最优。



# 禁忌搜索

- 禁忌 (prohibited or restricted by social custom) , 指的是不能触及的事物。
- 禁忌搜索是由弗雷德·格洛夫于1986年提出, 1989年加以形式化。
- 它是一种**元启发式(meta-heuristic)** 算法, 用于解决组合优化问题。
- 它使用一种局部搜索或邻域搜索过程, 从一个潜在的解  $x$  到改进的相邻解  $x'$  之间反复移动, 直到满足某些停止条件。
- 用于确定解的数据结构被称为**禁忌表** (tabu list) 。

## 禁忌搜索的三种策略

- Forbidding strategy 禁止策略

control what enters the tabulist. 控制何物进入该禁忌表。

- Freeing strategy 释放策略

control what exits the tabulist and when. 控制何物以及何时退出该禁忌表。

- Short-term strategy 短期策略

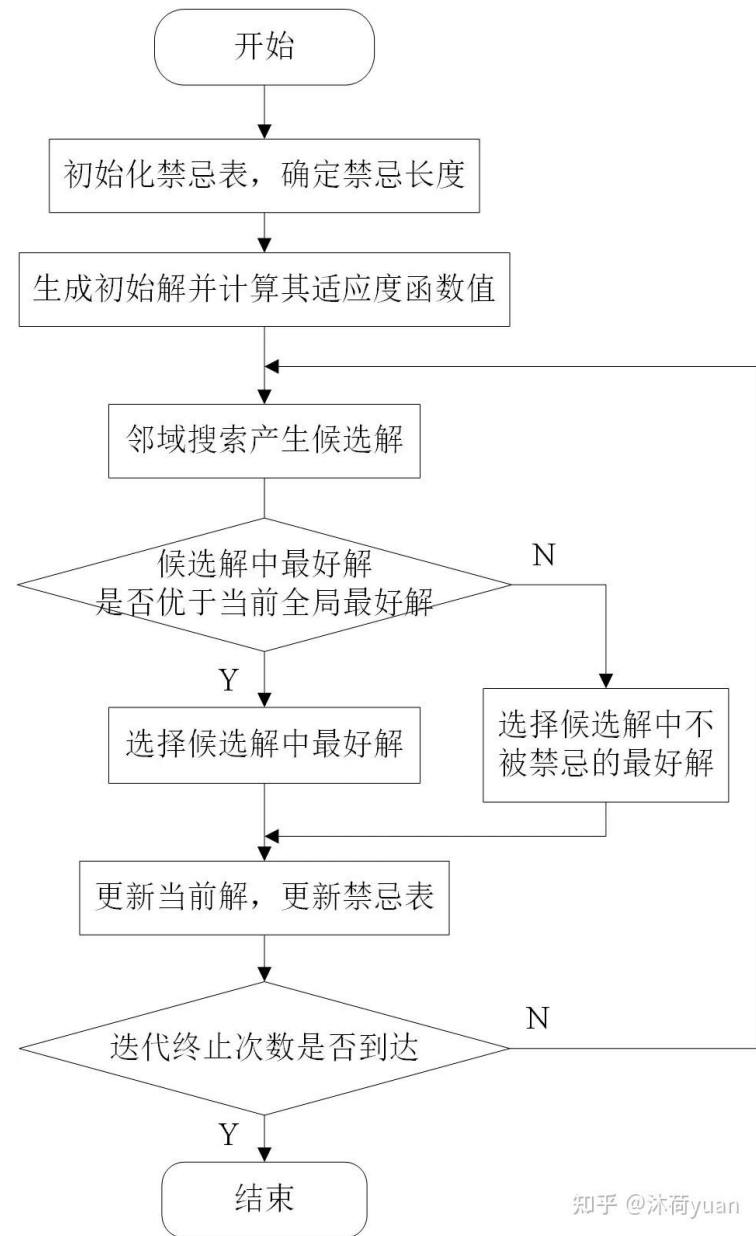
manage interplay between the forbidding strategy and freeing strategy to select trial solutions. 管理禁止策略和释放策略之间的相互作用来选择试验解。

- ✓ aspiration criterion 特赦准则

禁忌搜索算法中，迭代的某一步会出现候选集的某一个元素被禁止搜索，但是若解禁该元素，则会使评价函数有所改善，因此我们需要设置一个特赦规则，当满足该条件时该元素从禁忌表中跳出。

简洁版算法:

- (1) 给定一个禁忌表 ( $tabuList$ ) = null,  
并选定一个初始解  $s'$ 。
- (2) 如果满足停止规则, 则停止计算, 输出结果;  
否则, 在  $s$  的邻域  $sNeighborhoo$ d 中选出满足不受禁忌的  
候选集  $sCandidate$ 。  
在  $sCandidate$  中选择一个评价价值最优的解  $bestCandidate$  ,  
 $bestCandidate : = sCandidate$  ;  
更新历史记录  $tabuList$ 。  
重复步骤 (2)



知乎 @沐荷yuan



**function** TABU-SEARCH( $s$ ) **return** a best candidate

$sBest \leftarrow s \leftarrow s$

$tabuList \leftarrow$  null list

**while**(**not** STOPPING-CONDITION())

$candidateList \leftarrow$  null list

$bestCandidate \leftarrow$  null

**for**( $sCandidate$  in  $sNeighborhood$ )

**if**((**not**  $tabuList.CONTAINS(sCandidate)$ ) **and** ( $FITNESS(sCandidate) > FITNESS(bestCandidate)$ ))

**Then**  $bestCandidate \leftarrow sCandidate$

$s \leftarrow bestCandidate$

**if**( $FITNESS(bestCandidate) > FITNESS(sBest)$ ) **then**  $sBest \leftarrow bestCandidate$

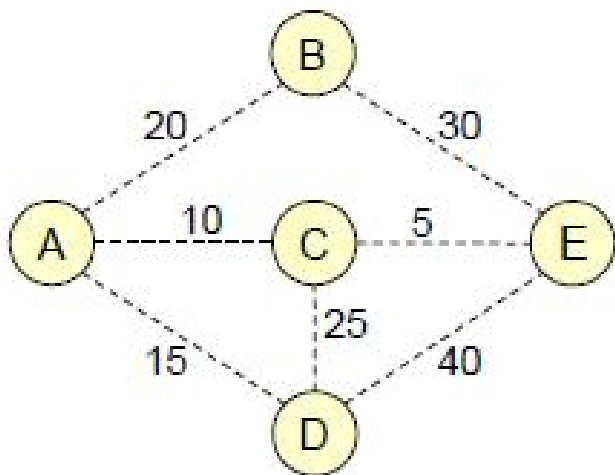
$tabuList.PUSH(bestCandidate)$

**if**( $tabuList.SIZE > maxTabuSize$ ) **then**  $tabuList.REMOVE-FIRST()$

**return**  $sBest$

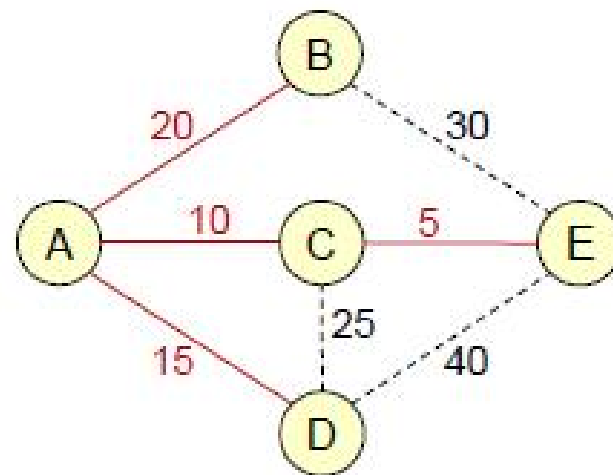
# Minimum Spanning Tree Problem 最小生成树问题

- **Objective 目标:** 用最小代价连接所有节点



Connects all nodes with minimum cost

用最小代价连接所有节点



An optimal solution without constraints

一个无约束的最优解

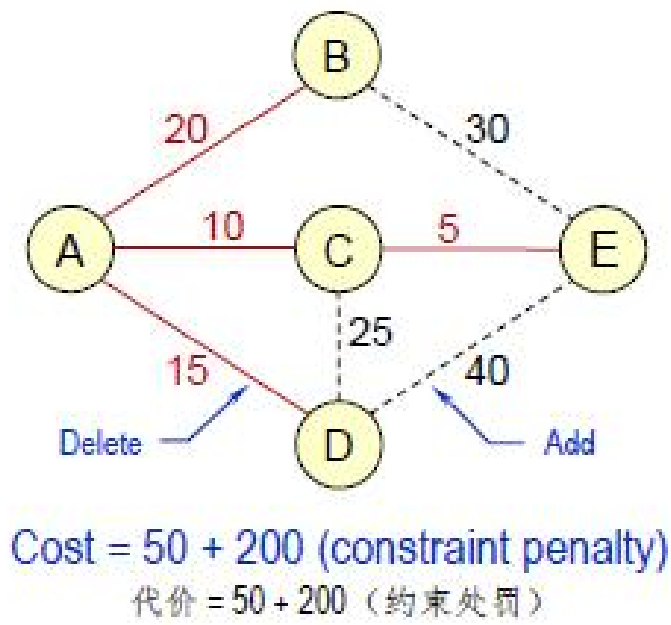
约束1：仅当包含连接DE时，才可以包含连接AD。（处罚：100）

约束2：至多可以包含三个连接（AD，CD和AB）中的一个。

（处罚：若选择了三个中的两个则处罚100，选择了全部三个则罚200）

# Minimum Spanning Tree Problem 最小生成树问题

• Iteration 1 迭代1



Local optimum  
局部最优

Add	Delete	Cost
BE	CE	$75 + 200 = 275$
BE	AC	$70 + 200 = 270$
BE	AB	$60 + 100 = 160$
CD	AD	$60 + 100 = 160$
CD	AC	$65 + 300 = 365$
DE	CE	$85 + 100 = 185$
DE	AC	$80 + 100 = 180$
<b>DE</b>	<b>AD</b>	<b><math>75 + 0 = 75</math></b>

约束1：仅当包含连接DE时，才可以包含连接AD。（处罚：100）

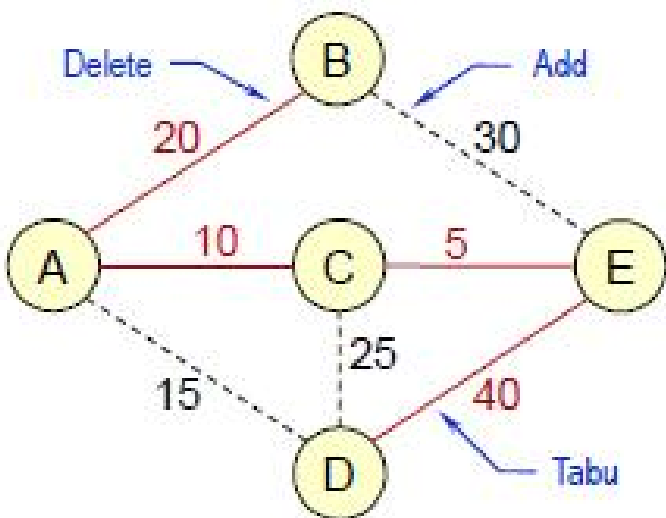
约束2：至多可以包含三个连接（AD，CD和AB）中的一个。

（处罚：若选择了三个中的两个则处罚100，选择了全部三个则罚

New Cost = 75  
新代价=75

# Minimum Spanning Tree Problem 最小生成树问题

## • Iteration 2 迭代2



Cost = 75, Tabu list: DE  
代价 = 75, 禁忌表: DE

## Escape local Optimum 溢出局部最优

Add	Delete	Cost
AD	DE*	Tabu Move
AD	CE	$85 + 100 = 185$
AD	AC	$80 + 100 = 180$
BE	CE	$100 + 0 = 100$
BE	AC	$95 + 0 = 95$
<b>BE</b>	<b>AB</b>	<b><math>85 + 0 = 85</math></b>
CD	DE*	Tabu Move
CD	CE	$95 + 100 = 190$

约束1: 仅当包含连接DE时, 才可以包含连接AD。(处罚: 100)

约束2: 至多可以包含三个连接 (AD, CD和AB) 中的一个。

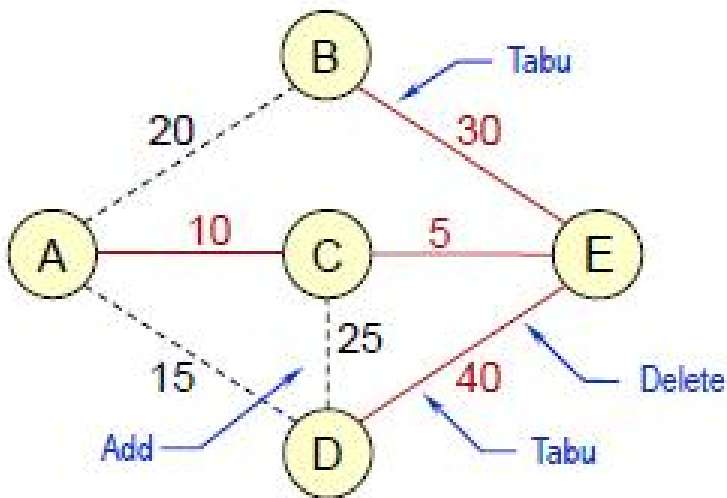
(处罚: 若选择了三个中的两个则处罚100, 选择了全部三个则罚

200)

**New Cost = 85**  
新代价=85

# Minimum Spanning Tree Problem 最小生成树问题

• Iteration 3 迭代3



Cost = 85, Tabu list: DE & BE  
代价 = 85, 禁忌表: DE & BE

Override tabu status  
覆盖禁忌状态

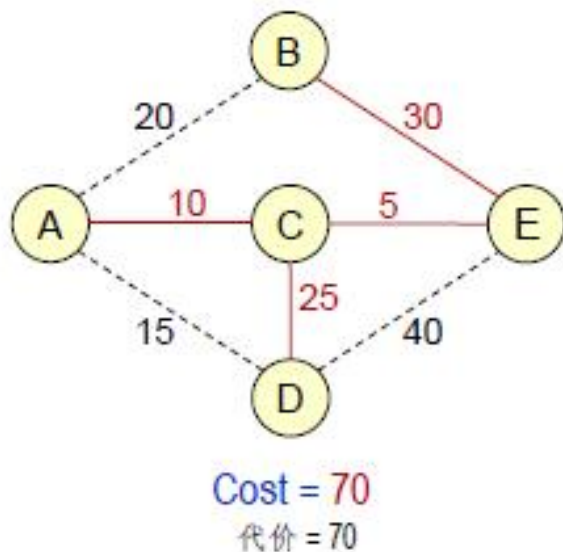
Add	Delete	Cost
AB	BE*	Tabu Move
AB	CE	100 + 0 = 100
AB	AC	95 + 0 = 95
AD	DE *	60 + 100 = 160
AD	CE	95 + 0 = 95
AD	AC	90 + 0 = 90
CD	DE*	70+0 = 70
CD	CE	105 + 0 = 105

约束1: 仅当包含连接DE时, 才可以包含连接AD。(处罚: 100)  
约束2: 至多可以包含三个连接 (AD, CD和AB) 中的一个。  
(处罚: 若选择了三个中的两个则处罚100, 选择了全部三个则罚200)

New Cost = 70  
新代价 = 70

# Minimum Spanning Tree Problem 最小生成树问题

- Iteration 4 迭代4



**Optimal Solution**  
最优解

*Additional iterations only find inferior solutions*  
额外的迭代只会找到较差解

约束1：仅当包含连接DE时，才可以包含连接AD。（处罚：100）

约束2：至多可以包含三个连接（AD，CD和AB）中的一个。

（处罚：若选择了三个中的两个则处罚100，选择了全部三个则

罚200）

## 可用禁忌搜索 解决的问题

- |                                  |          |
|----------------------------------|----------|
| • Travelling Salesperson Problem | 旅行推销员问题  |
| • Traveling Tournament Problem   | 旅行比赛问题   |
| • Job-shop Scheduling Problem    | 车间作业调度问题 |
| • Network Loading Problem        | 网络加载问题   |
| • The Graph Coloring Problem     | 图着色问题    |
| • Hardware/Software Partitioning | 硬件/软件划分  |
| • Minimum Spanning Tree Problem  | 最小生成树 问题 |

## 禁忌搜索的应用领域

- |                              |         |
|------------------------------|---------|
| • Resource planning          | 资源规划    |
| • Telecommunications         | 通讯      |
| • VLSI design                | VLSI 设计 |
| • Financial analysis         | 金融分析    |
| • Scheduling                 | 调度      |
| • Space planning             | 空间规划    |
| • Energy distribution        | 能源分配    |
| • Molecular engineering      | 分子工程    |
| • Logistics                  | 后勤保障    |
| • Flexible manufacturing     | 柔性生产    |
| • Waste management           | 废物管理    |
| • Mineral exploration        | 矿产勘探    |
| • Biomedical analysis        | 生物医药分析  |
| • Environmental conservation | 环境保护    |



1

超越经典概述

2

局部搜索

3

优化算法

模拟退火 Simulated Annealing

4

CSP问题

5

联机搜索

6

小结



# Simulated Annealing 模拟退火搜索

- **引论:** 爬山法的从不下山策略导致其**不完备性**，而纯粹的随机行走，又导致**效率的低下**，把单纯的爬山法和纯粹的随机相结合就是以下的**模拟退火**算法的思想。
- **Idea:** 通过允许向不好的状态移动来避开局部最值点，但允许的频率逐渐降低。

起源：冶金中的退火原理。

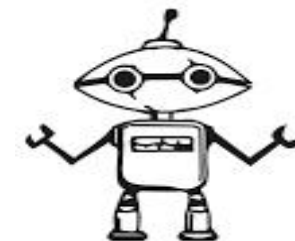
退火用于对金属和玻璃进行回火或硬化。

将一个固体放在高温炉内进行加热，提升温度至最大值。在该温度下，所有的材料都处于液体状态，并且粒子本身随机地排列。随着高温炉内的温度逐渐冷却，该结构的所有粒子将呈现低能状态。

例子：弹球的分析

剧烈的摇动（=高温）

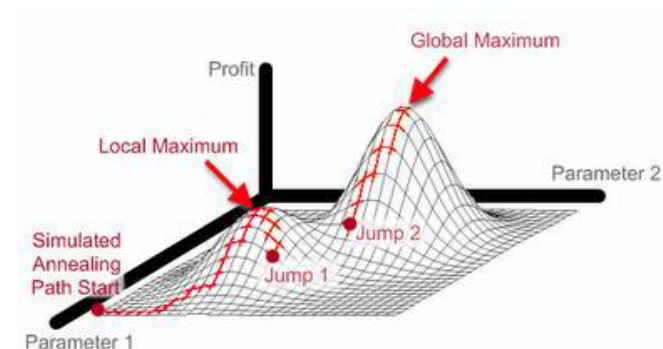
减少晃动（=降温）



# Simulated Annealing 模拟退火搜索

- 模拟退火是一种给定函数逼近全局最优解的概率方法。发表于 1953 年。  
具体来说，是一种在大搜索空间逼近全局最优解的**元启发式**方法。
- Optimization and Thermodynamics 优化与热力学

Objective function	目标函数	$\Leftrightarrow$ Energy level	能量极位
Admissible solution	可接受解	$\Leftrightarrow$ System state	系统状态
Neighbor solution	相邻解	$\Leftrightarrow$ Change of state	状态变化
Control parameter	控制参数	$\Leftrightarrow$ Temperature	温度
Better solution	更优解	$\Leftrightarrow$ Solidification state	凝固状态



# Simulated Annealing 模拟退火算法

- Initial Solution 初始解

Generated using an heuristic Chosen at random.

使用随机选择启发式方法生成。

- Neighborhood 相邻节点

Generated randomly. Mutating the current solution.

随机生成当前解的变异。

- Acceptance 接受条件

Neighbor has lower cost value, higher cost value is accepted with the probability  $p$ .

相邻节点使情况：（1）改善，接受；（2）变坏，则以概率 $P$ 接受。

- Stopping Criteria 停止判据

Solution with a lower value than threshold. Maximum total number of iterations.

解具有比阈值低的值。已达到迭代最大总次数。

## 模拟退火算法

**function** SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state

**inputs:** *problem*, a problem

*schedule*, a mapping from time to “temperature”

*current* ← MAKE-NODE(*problem*.INITIAL-STATE)

**for**  $t = 1$  **to**  $\infty$  **do**

$T \leftarrow \text{schedule}(t)$

**if**  $T == 0$  **then return** *current*

*next* ← a **randomly** selected successor of *current*

$\Delta E \leftarrow \text{next.V} \text{ALUE} - \text{current.V} \text{ALUE}$

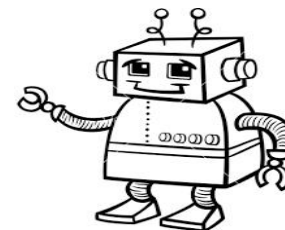
**if**  $\Delta E > 0$  **then** *current* ← *next* // 相邻节点有改善

**else** *current* ← *next* only with probability  $e^{\Delta E/T}$

一种允许部分下山移动的随机爬山法的版本。下山移动在退火调度的早期容易被接受，随着时间的推移逐步降低。

## 模拟退火搜索的性质

- 可以证明: 如果  $T$  降低的足够慢, 则模拟退火能以趋近于1的概率找到最优解。
- 广泛应用于VLSI 布局问题, 航空调度等领域

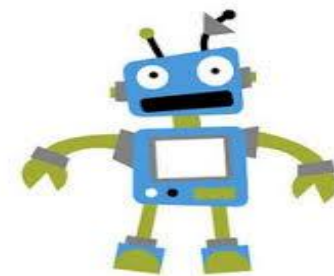


1	超越经典概述	
2	局部搜索	
3	优化算法	
4	CSP问题	<ul style="list-style-type: none"><li>4.1 约束满足问题定义</li><li>4.2 约束满足问题放入标准搜索</li><li>4.3 回溯搜索算法求解CSP</li><li>4.4 局部搜索算法求解CSP</li></ul>
5	联机搜索	
6	小结	



## 4.1 约束满足问题

- 标准搜索问题：
  - 状态是一个黑盒，通过后续函数，启发式函数和目标测试来访问。
- CSP：
  - 状态由多个变量  $X_i$  定义，变量  $X_i$  的值域为  $D_i$
  - 目标测试是由约束集来确定，约束指定变量子集允许的赋值组合。





# 约束满足问题表示

- **CSP形式化:**

- 有限变量集:  $\{X_1, X_2, \dots, X_n\}$
- 有限约束集:  $\{C_1, C_2, \dots, C_m\}$
- 每个变量有非空可能值域:  $D_{x1}, D_{x2}, \dots, D_{xn}$
- 每个约束 $C_i$ 的值可以用变量来表示: 例如  $X_1 \neq X_2$

- 一个状态被定义为所有变量的赋值

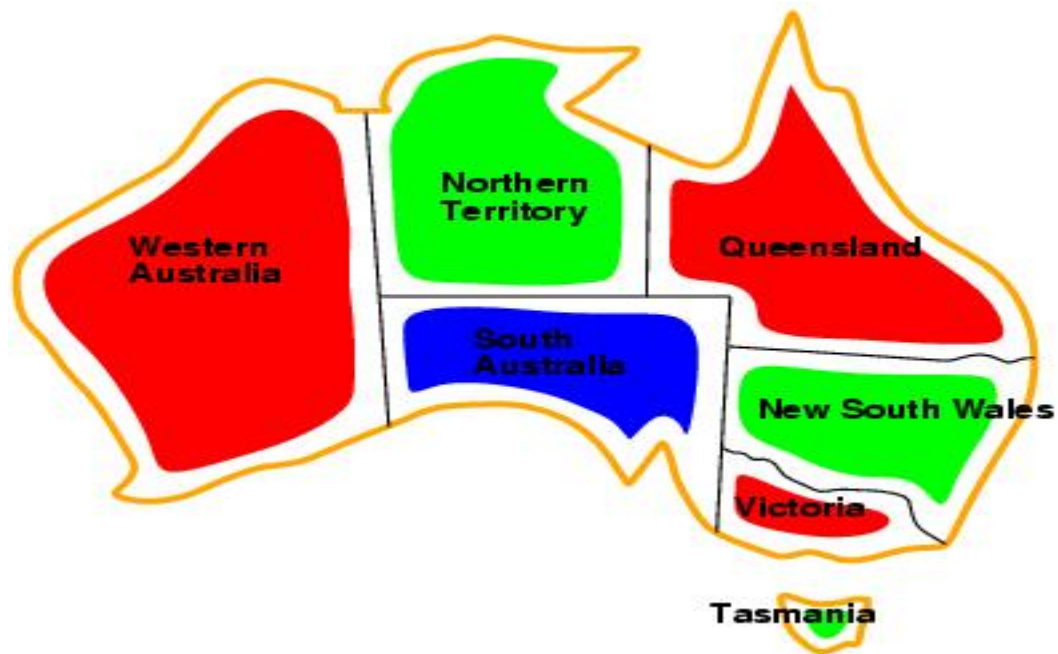
一个相容的赋值: 所有赋值均满足所有约束

## 例子: 地图着色问题



- 变量:  $WA, NT, Q, NSW, V, SA, T$
- 域:  $D_i = \{\text{red, green, blue}\}$
- 约束: 相邻区域不能着相同的颜色

## 例子: 地图着色问题



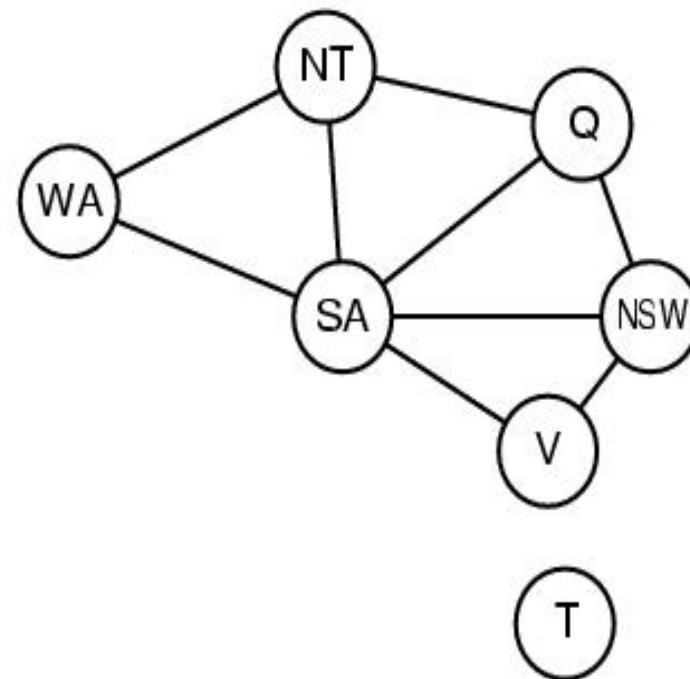
- 解是完整而且符合约束的赋值,

如: WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green

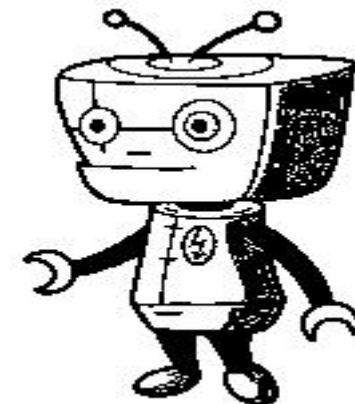
- 一些CSP问题要求一个目标函数的最佳解。

# 约束图：

- 二元 **CSP**：每个约束都关联两个变量
- 约束图：结点表示变量，边表示约束
- **CSP**的特点（优势）
  - 呈现出标准的模式
  - 通用的目标和后继函数
  - 通用的启发函数（无域的特别技术）



# CSP的变量类型



- 离散变量

- 有限域:

- $n$  个变量, 域大小为  $d \rightarrow O(d^n)$

- 无限域:

- 作业调度, 变量是工序的起始和结束日期。

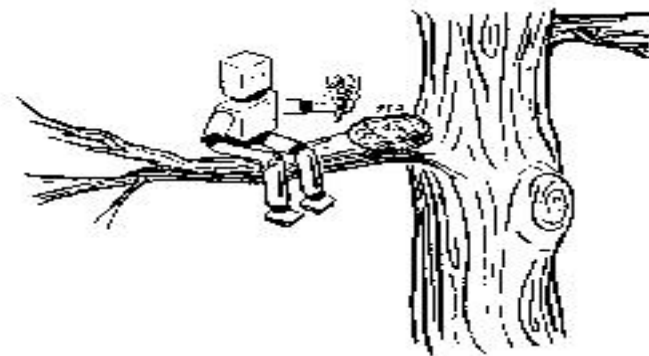
- 约束:  $StartJob_1 + 5 \leq StartJob_3$

- 连续变量

- 哈勃太空望远镜观测的起始和结束时间。

# 变量约束类型

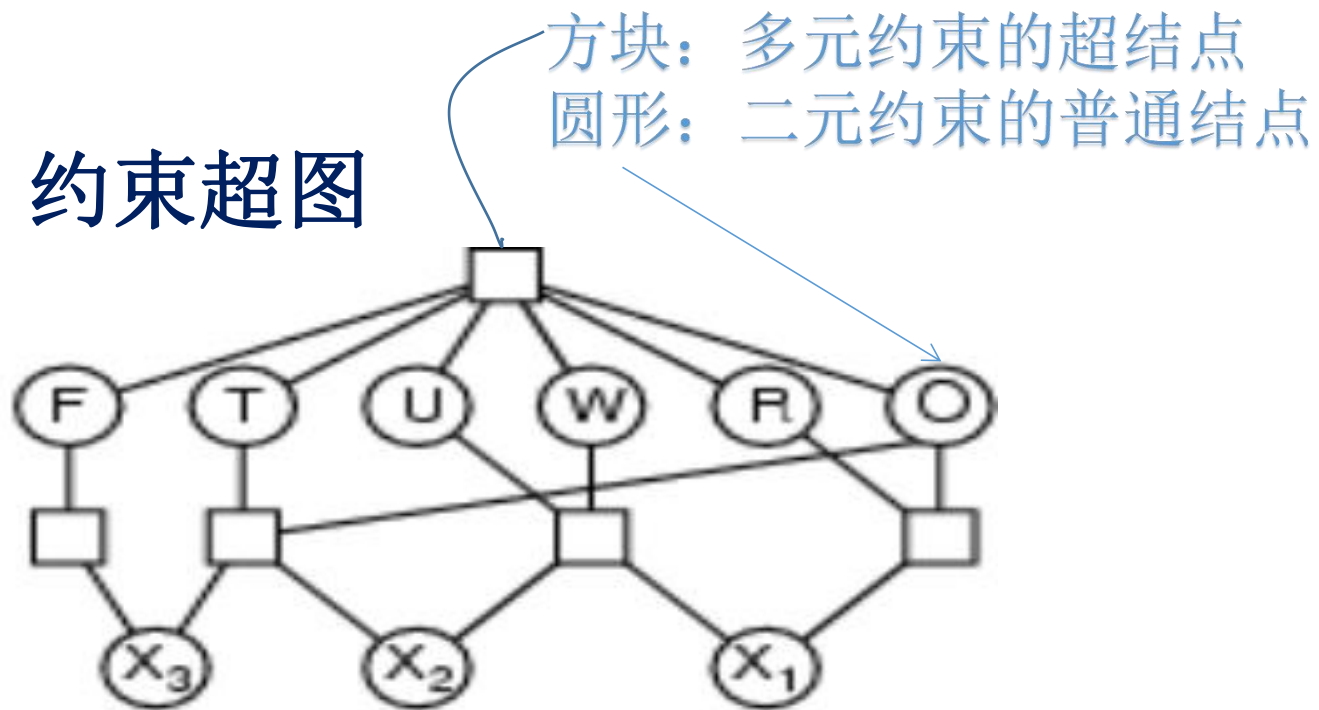
- 一元约束只涉及到一个变量
  - e.g.,  $SA \neq \text{green}$
- 二元约束涉及到两个变量
  - e.g.,  $SA \neq WA$
- 高阶约束涉及到3个以个的变量
  - e.g., 密码算术约束
- 偏好约束



# 例子:密码算术

$$\begin{array}{r}
 \text{TWO} \\
 + \text{TWO} \\
 \hline
 \text{FOUR}
 \end{array}$$

约束超图



- 变量:

$FTUWR O \text{ } X_1 \text{ } X_2 \text{ } X_3$

- 域:

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

- 约束:  $Alldiff(F,T,U,W,R,O)$

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

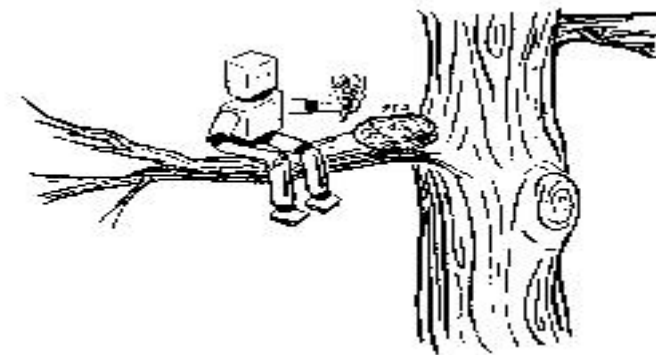
$$X_1 + W + W = U + 10 \cdot X_2$$

$$X_2 + T + T = O + 10 \cdot X_3, T \neq 0$$

$$X_3 = F, F \neq 0$$

# 变量约束类型

- 一元约束只涉及到一个变量
  - e.g.,  $SA \neq \text{green}$
- 二元约束涉及到两个变量
  - e.g.,  $SA \neq WA$
- 高阶约束涉及到3个以个的变量
  - e.g., 密码算术约束



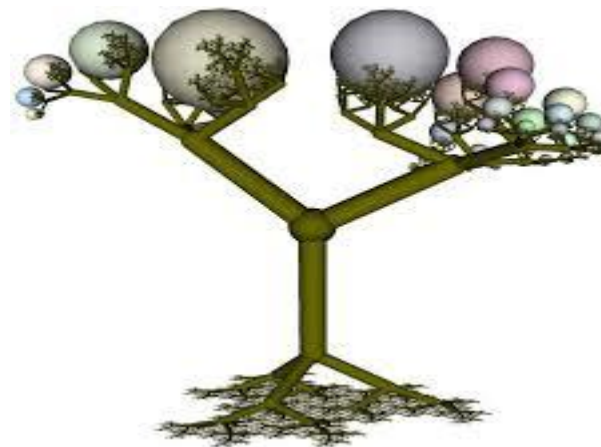
任何有限值域的约束都可以通过加入条件转化为二元约束

- 偏好约束（软约束）（指出哪些解是更喜欢的）
  - e.g., 用一个代价值代表红色比绿色好



# 真实世界 CSPs

- 分配问题
- 排课问题
- 运输调度
- 生产调度



## 4.2 标准搜索形式化CSP(增量形式化)

### CSP的特点（优势）

- 呈现出标准的模式
- 通用的目标和后继函数
- 通用的启发函数（无域的特别技术）

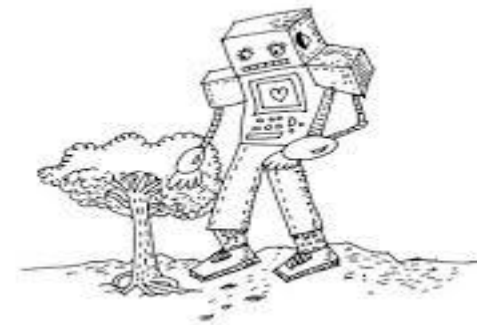
状态用已经给变量赋的值来表示



- **初始状态：** 空  $\{\}$
- **后续函数：** 给一个没有赋值的变量赋值，此值与已经赋的值不能与约束冲突。
- **目标测试：** 当前赋值是完整赋值，即所有变量都有值。

# 标准搜索形式化CSP(增量形式化)

- 有  $n$  个变量的问题，解都在深度为  $n$  的层  
→ 可以使用深度优先搜索
- 在深度为  $l$  时，分支数  $b = (n - l)d$ ，所以搜索树有  $n! \cdot d^n$  个叶子结点

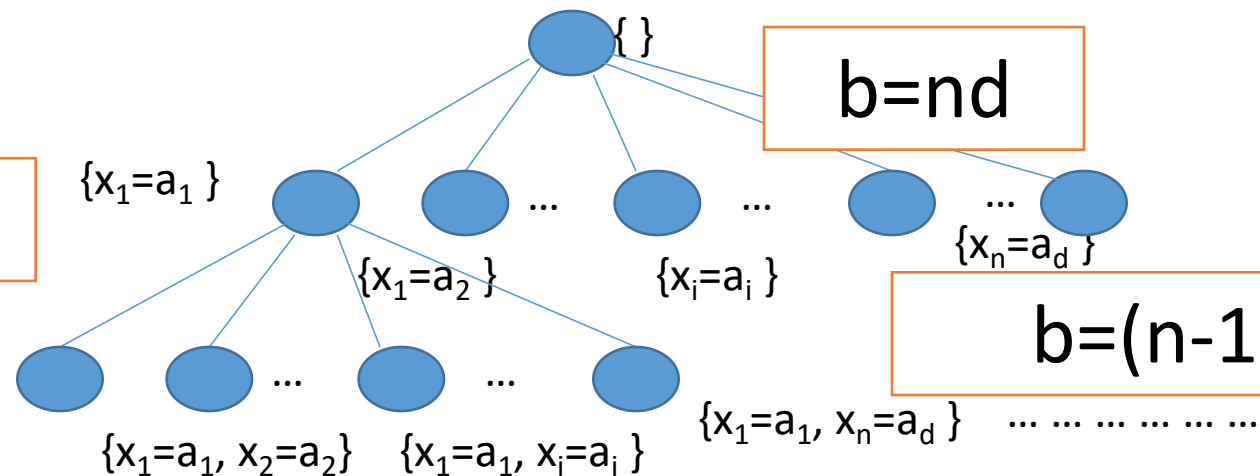


# 标准搜索形式化(增量形式化)

- 状态表示:  $x_1, x_2, \dots, x_n$
- 最大  $|D_i|=d$

$$nd$$

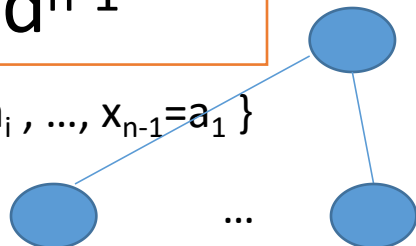
$$n(n-1)d^2$$



$$n(n-1)\dots 2d^{n-1}$$

$$\{x_1=a_1, x_2=a_1, \dots, x_i=a_i, \dots, x_{n-1}=a_1\}$$

$$n!d^n$$



$$b=d$$

$$\{x_1=a_1, x_2=a_1, \dots, x_i=a_i, \dots, x_{n-1}=a_1, x_n=a_d\}$$



## 4.3 回溯搜索

- 变量赋值是可交换的, i.e.,  
[ WA = red then NT = green ] 等同于 [ NT = green then WA = red ]
- 每个结点只需要考虑给一个变量赋值  
→  $b = d$  所以  $d^n$  片叶子
- 采用单变量赋值的深度优先搜索称为回溯搜索



# 回溯搜索算法

```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
  return RECURSIVE-BACKTRACKING( $\{\}$ , csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns a solution, or failure
  if assignment is complete then return assignment
  var  $\leftarrow$  SELECT-UNASSIGNED-VARIABLE(Variables[csp], assignment, csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
    if value is consistent with assignment according to Constraints[csp] then
      add { var = value } to assignment
      result  $\leftarrow$  RECURSIVE-BACKTRACKING(assignment, csp)
      if result  $\neq$  failure then return result
      remove { var = value } from assignment
  return failure
```

# 回溯搜索示例

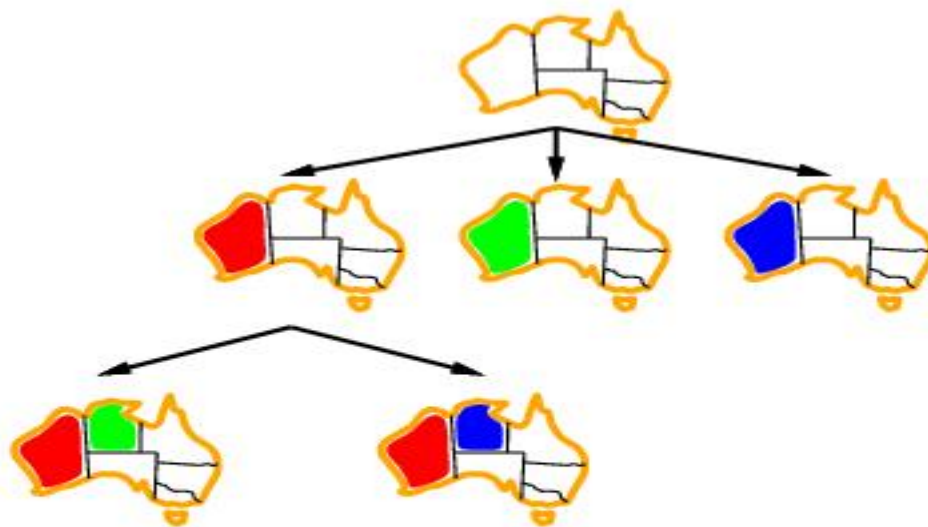


# 回溯搜索示例

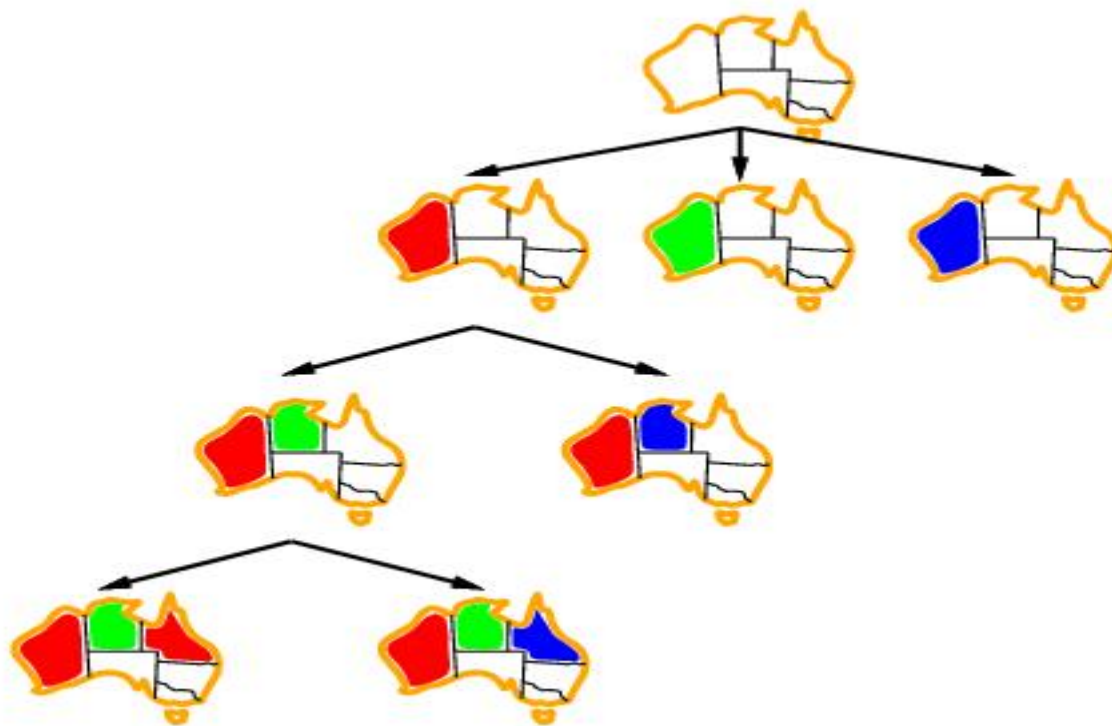




# 回溯搜索示例



# 回溯搜索示例



## 提高回溯搜索的效率

- 通用方法就能巨大提高效率:
  - 下一步应该给哪个变量赋值?
  - 应该以一种什么顺序来试着给变量赋值?
  - 能不能早些检测到不可避免的失败?

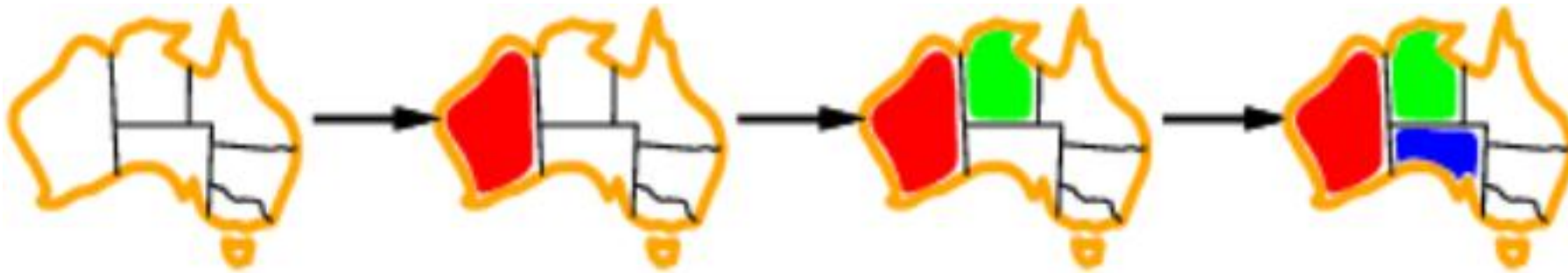


如何较早探测到不可避免的失败，并且避免它

```
if assignment is complete then return assignment
var ← SELECT-UNASSIGNED-VARIABLE(Variables[csp], assignment, csp)
for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
    if value is consistent with assignment according to Constraints[csp] then
        add { var = value } to assignment
```

## 回溯改进——最大受限变量

- 最大受限变量（most constrained variables）：  
具有最少合法赋值的变量

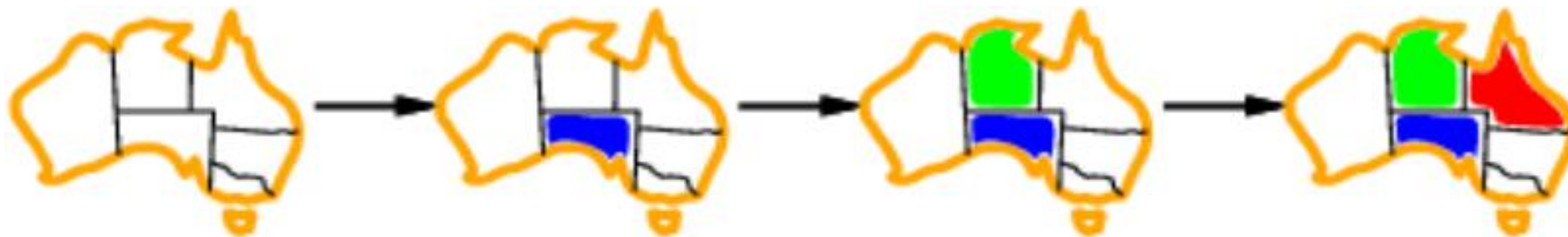


- 也称为最少剩余值**启发式**  
(minimum remaining values , MRV)

## 回溯改进——最大约束变量

- 当多个变量的MRV值相同时，采用最大约束准则  
(Most constraining variable)

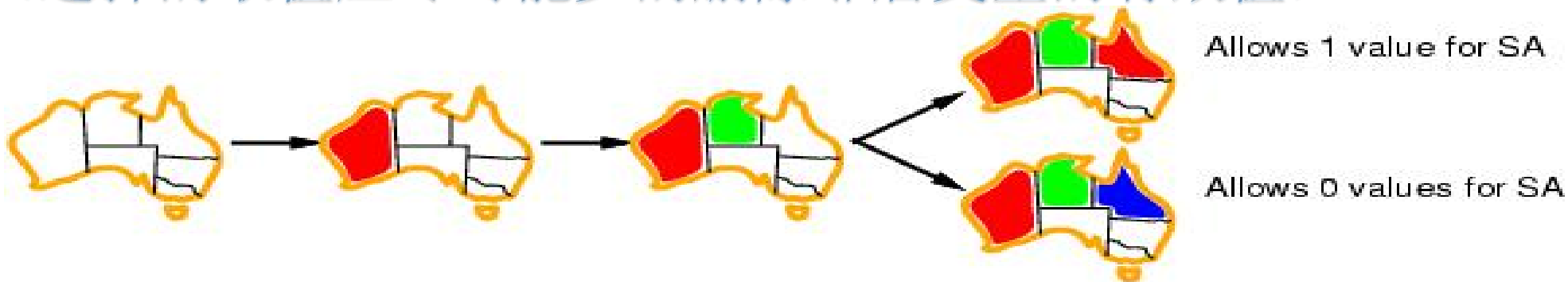
从中选择约束其他未赋值变量最多的变量



## 回溯改进——最少约束值

- 选择好了变量后，选择一个最少约束值来尝试赋值  
即选择一个对其他未赋值变量的合法赋值影响最少的值

(选择的取值应尽可能少的删除邻居变量的有效值)



# 前向检查

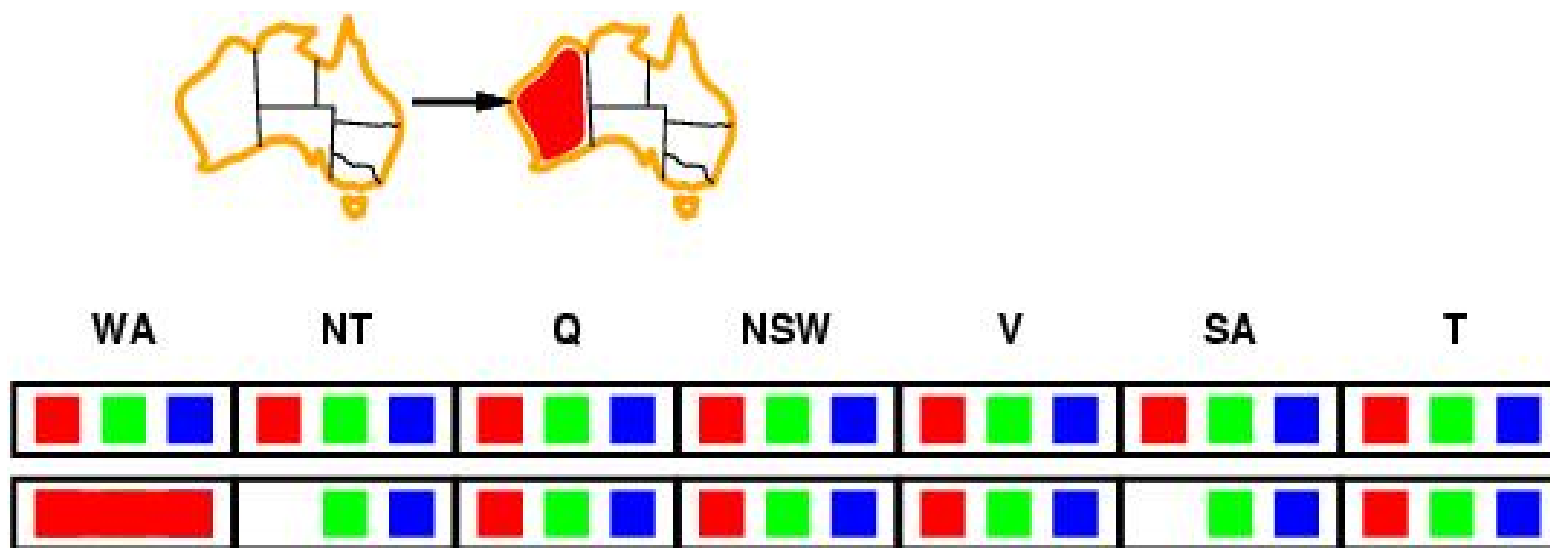
- 我们如何较早探测到不可避免的失败？
  - 并且在后面避免它。
- 向前检查的思想：只保留未赋值的变量的合法值。
- 当任何变量出现没有合法可赋值的情况终止搜索。

跟踪未赋值变量的合法赋值，当发现有变量没有合法赋值时就停止搜索。



# 前向检查

- 跟踪未赋值变量的合法赋值，当发现有变量没有合法赋值时就停止搜索

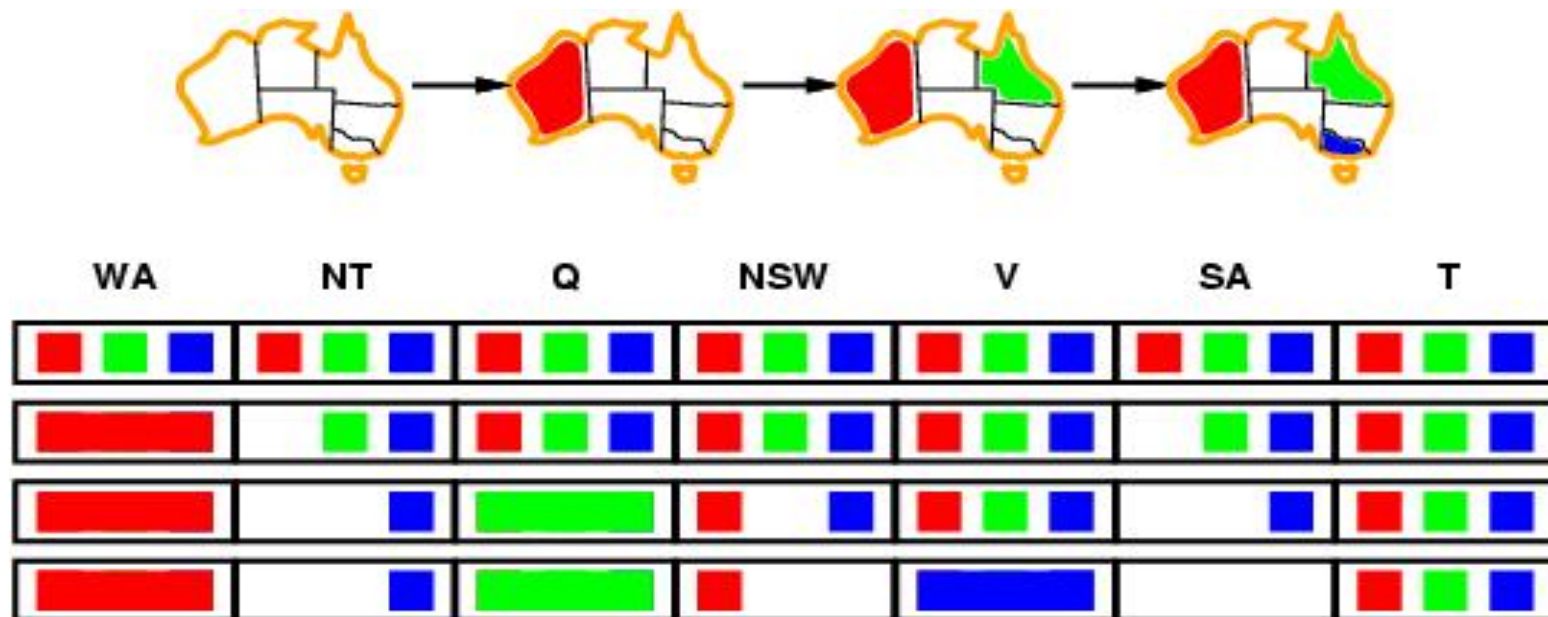






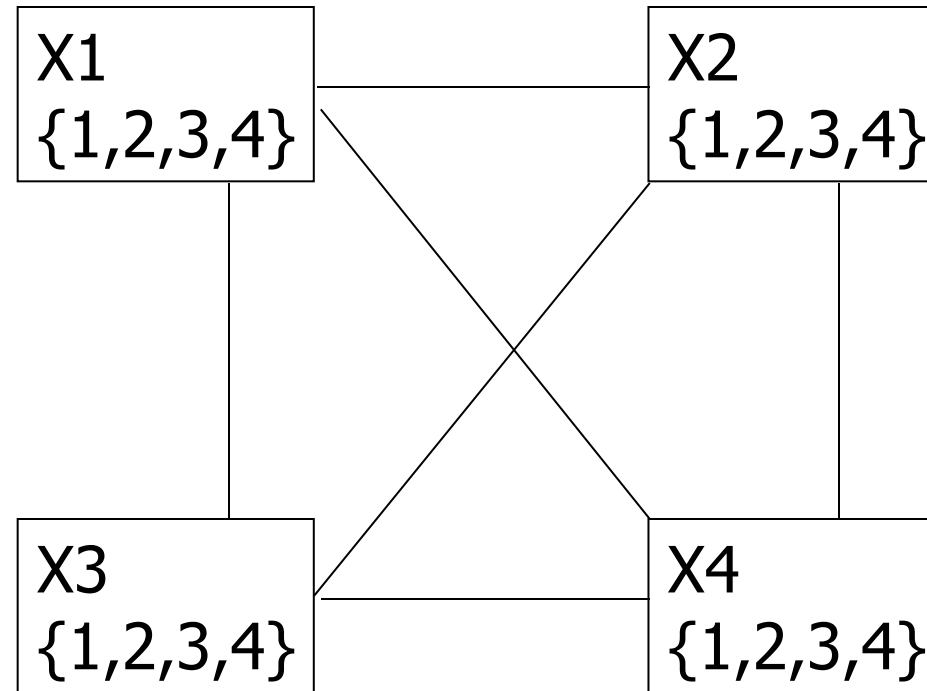
# 前向检查

- 跟踪未赋值变量的合法赋值，当发现有变量没有合法赋值时就停止搜索

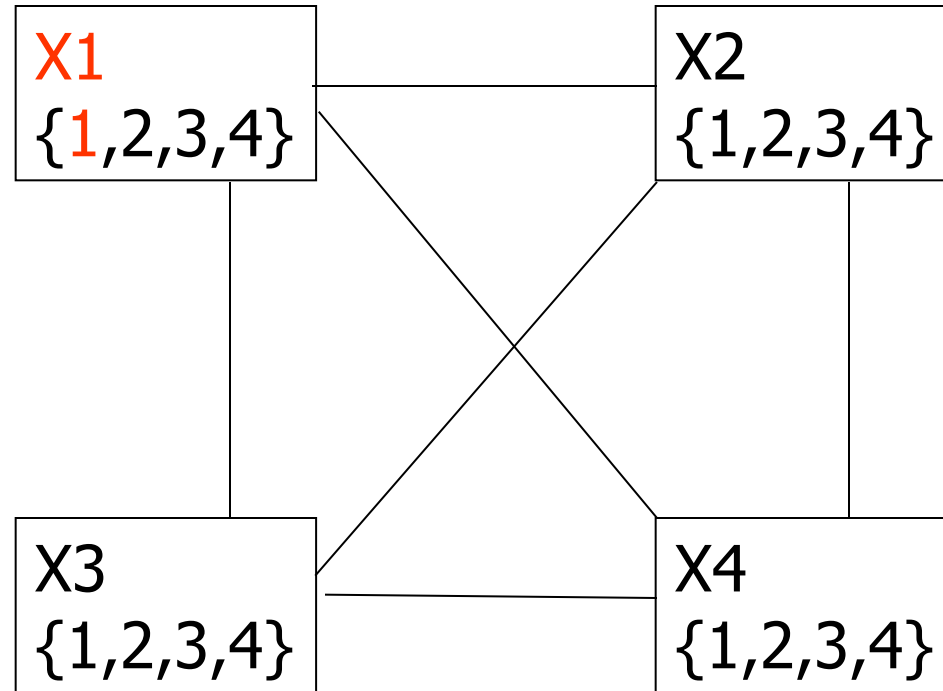
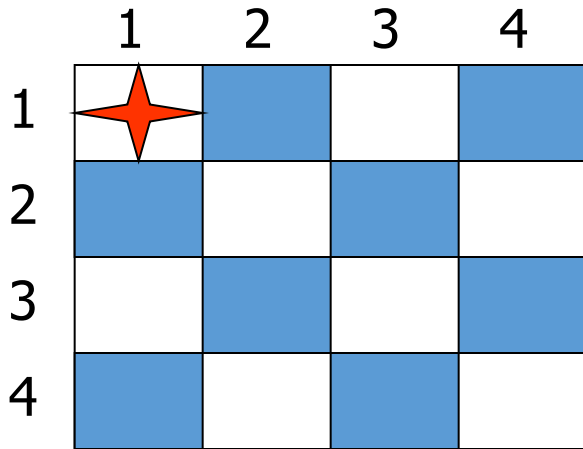


# Example: 4-Queens Problem

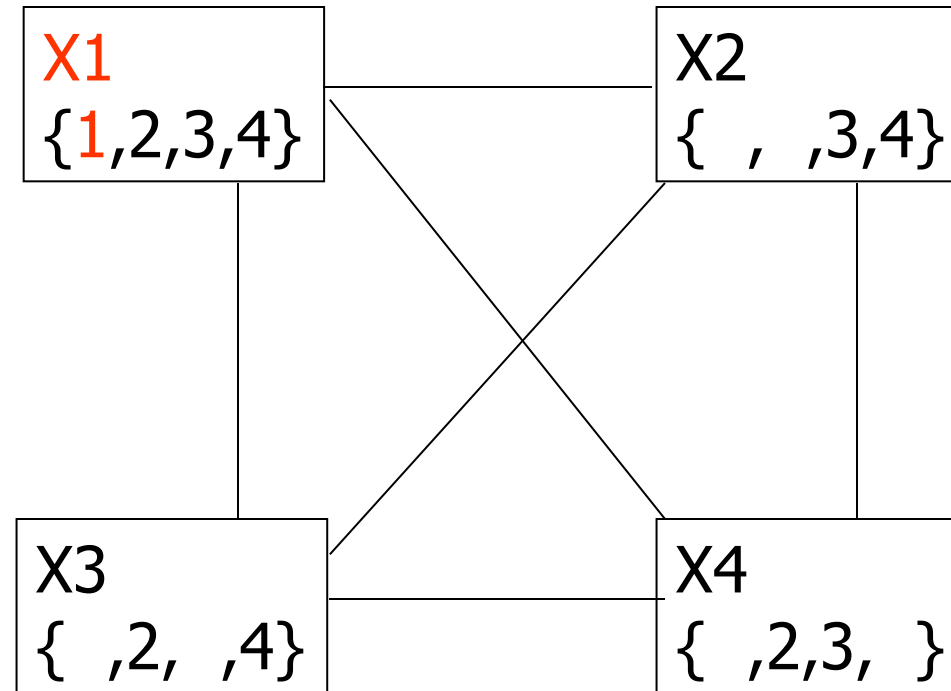
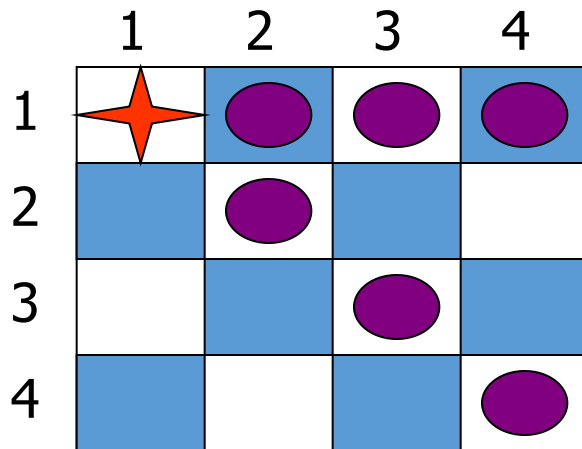
	1	2	3	4
1				
2				
3				
4				



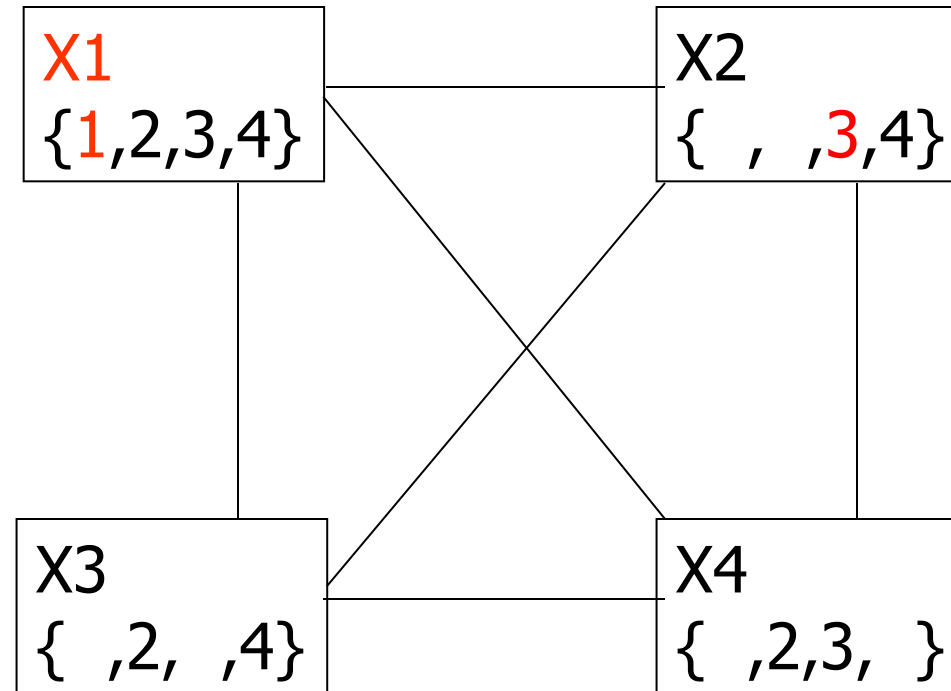
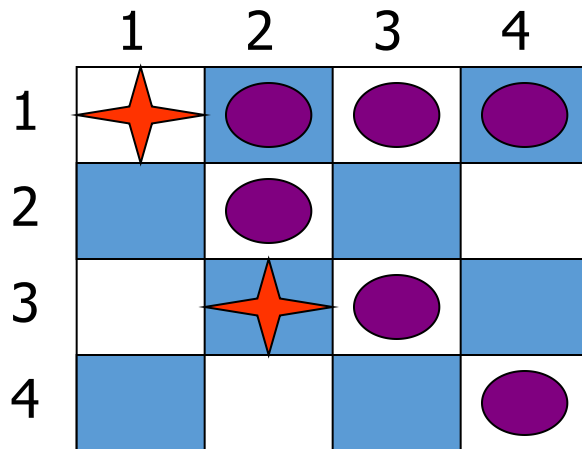
# Example: 4-Queens Problem





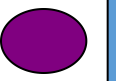
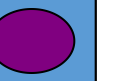
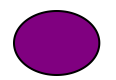
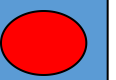

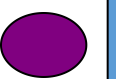
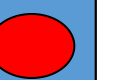
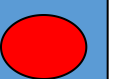
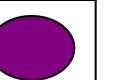
# Example: 4-Queens Problem

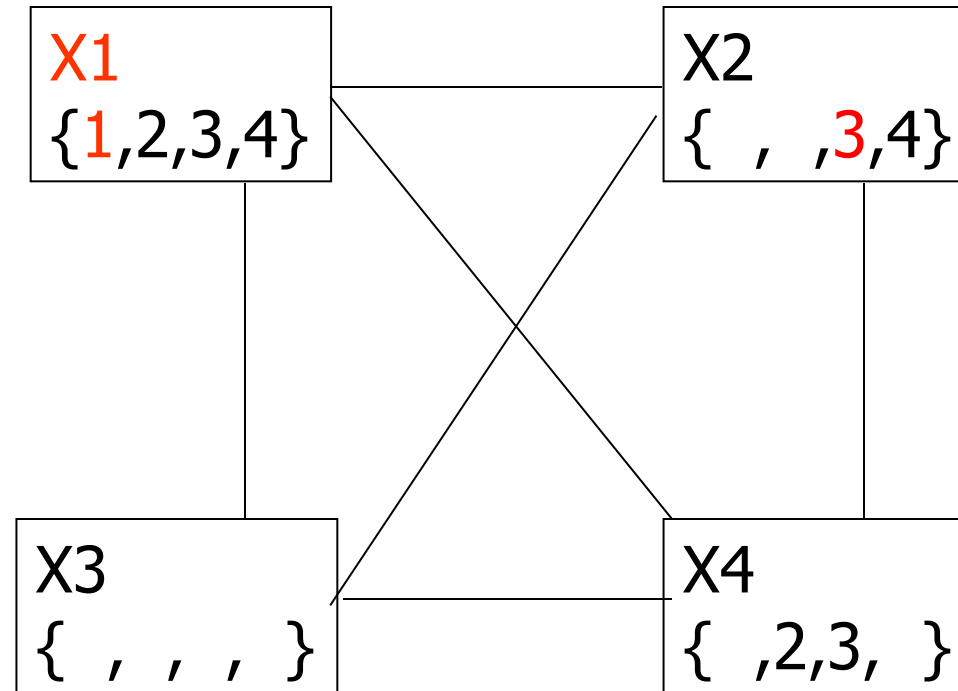


# Example: 4-Queens Problem



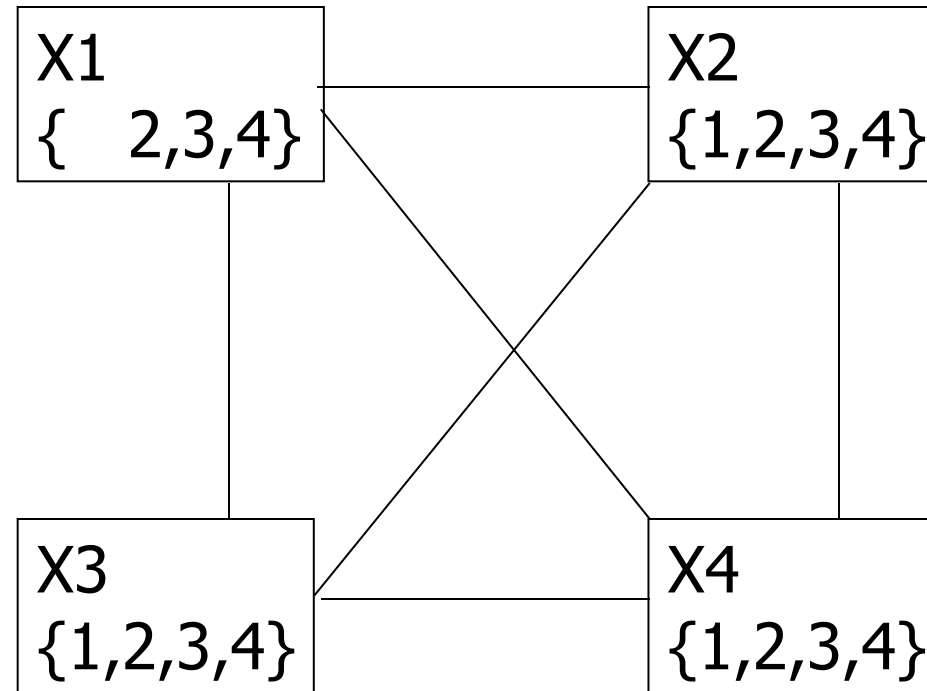
# Example: 4-Queens Problem

	1	2	3	4
1				
2				
3				
4				



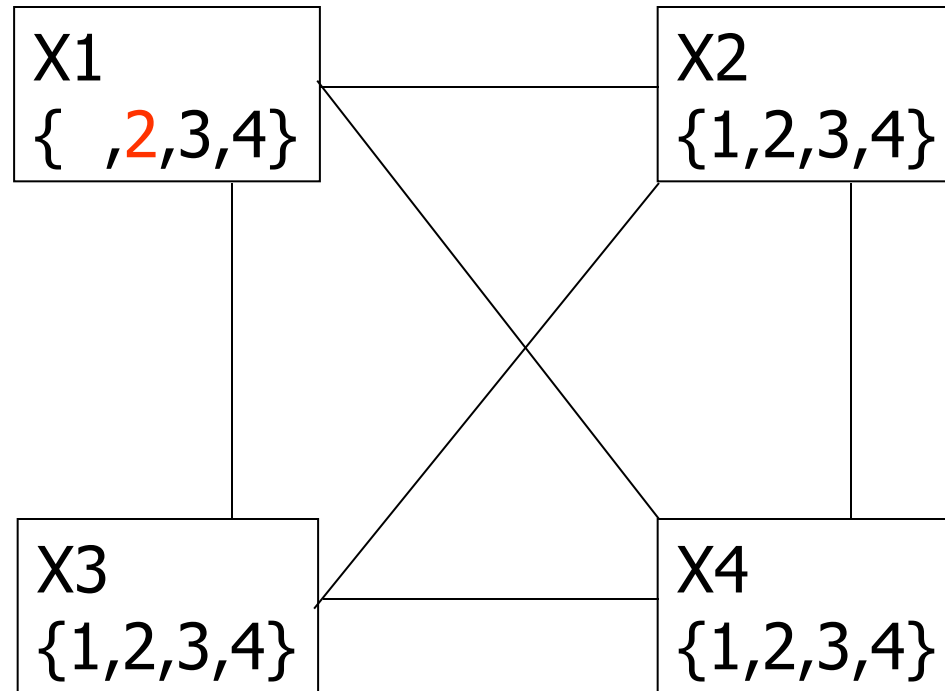
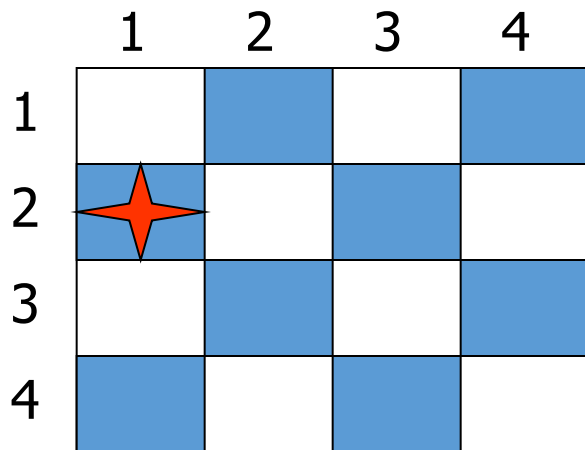
# Example: 4-Queens Problem

	1	2	3	4
1				
2				
3				
4				





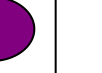




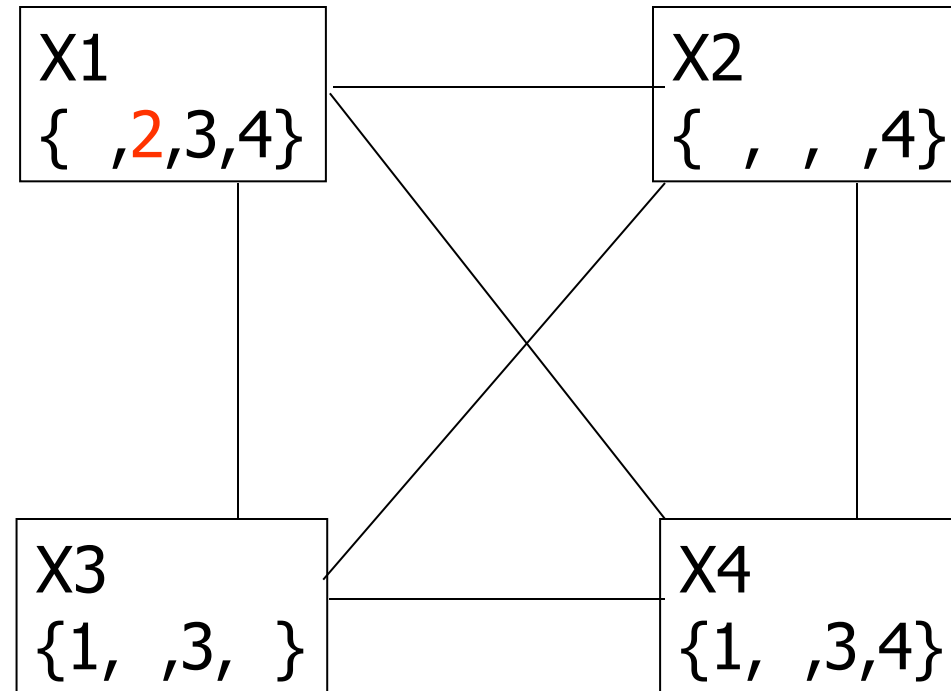


# Example: 4-Queens Problem











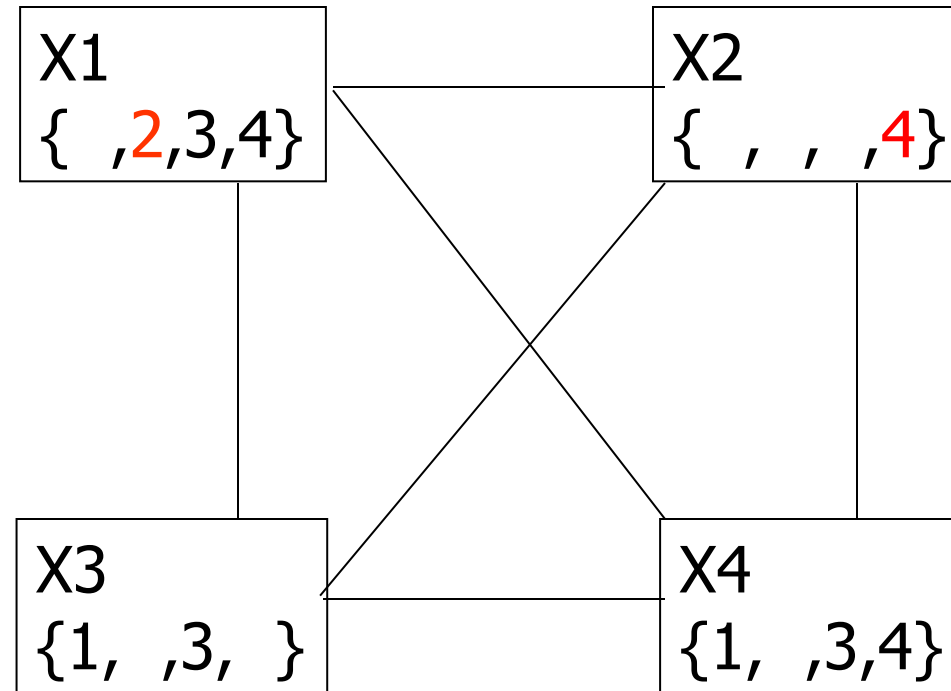
# Example: 4-Queens Problem

	1	2	3	4
1				
2				
3				
4				





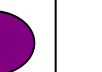

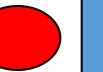


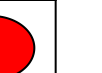


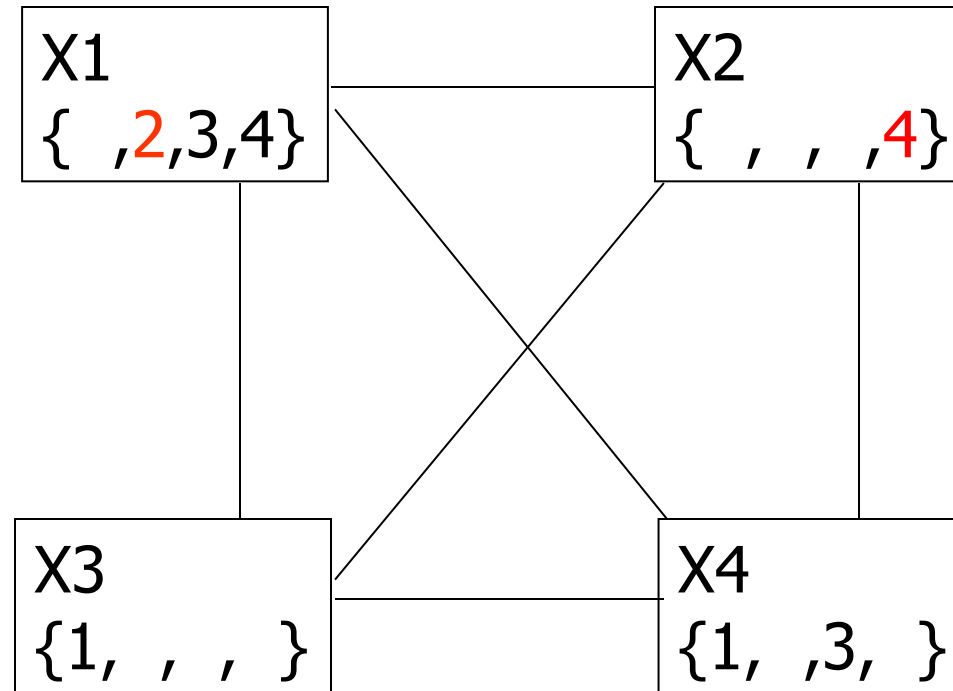
# Example: 4-Queens Problem

	1	2	3	4
1				
2				
3				
4				

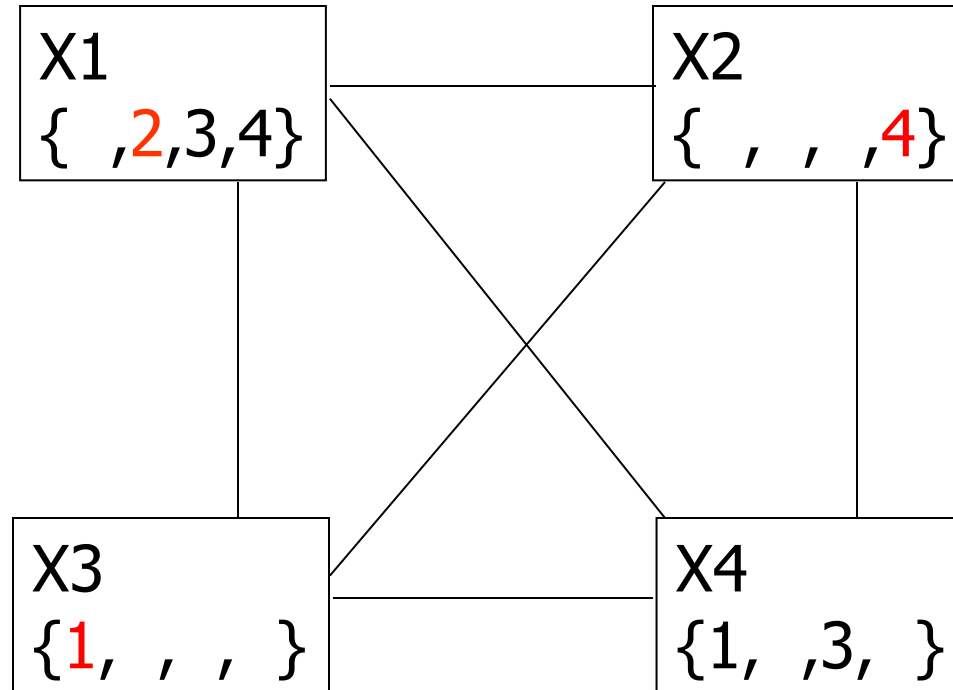
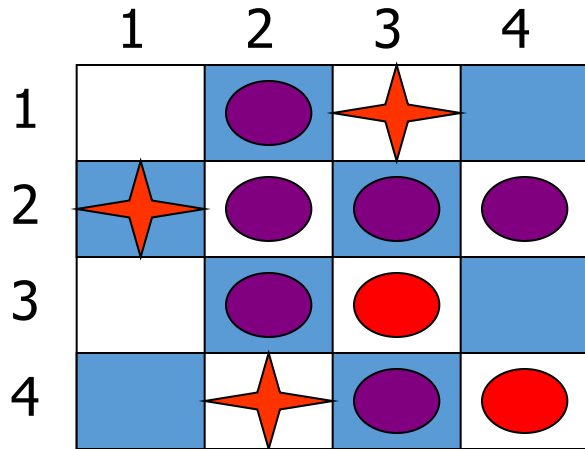


# Example: 4-Queens Problem










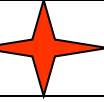

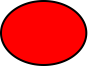
	1	2	3	4
1				
2				
3				
4				

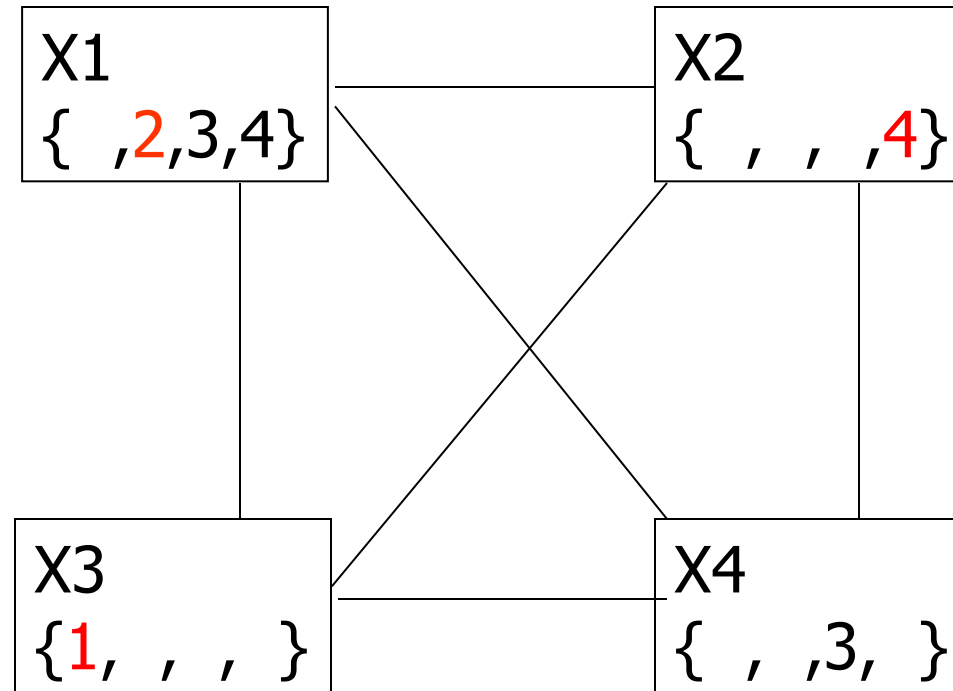


# Example: 4-Queens Problem



# Example: 4-Queens Problem

	1	2	3	4
1				
2				
3				
4				



# Example: 4-Queens Problem

