目录

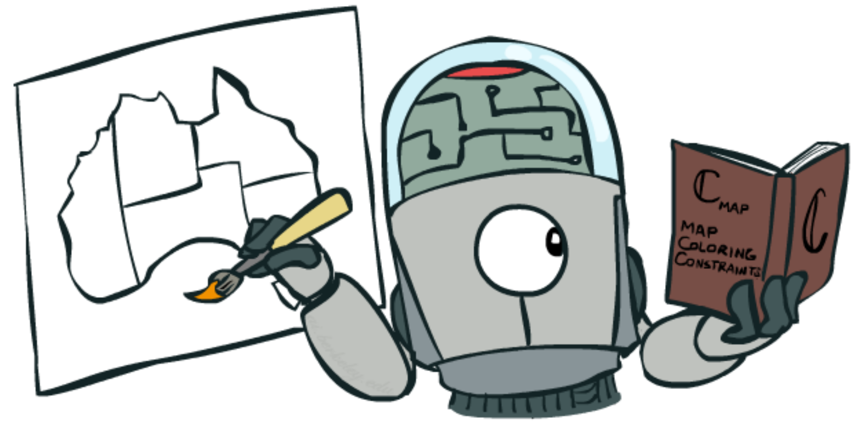
1 约束满足问题*

2 回溯搜索求解**

3 搜索优化***

4 局部搜索*

5 习题及实验



1. 约束满足问题

标准搜索问题：

状态表示：数据结构

后继函数：操作或动作

初始及目标状态：测试

问题的解：将开始状态转换为目标状态的一系列动作

约束满足问题：

搜索问题的特殊子集

约束满足问题包含：

变量的集合$X = \{X_1, X_2, \ldots, X_n\}$

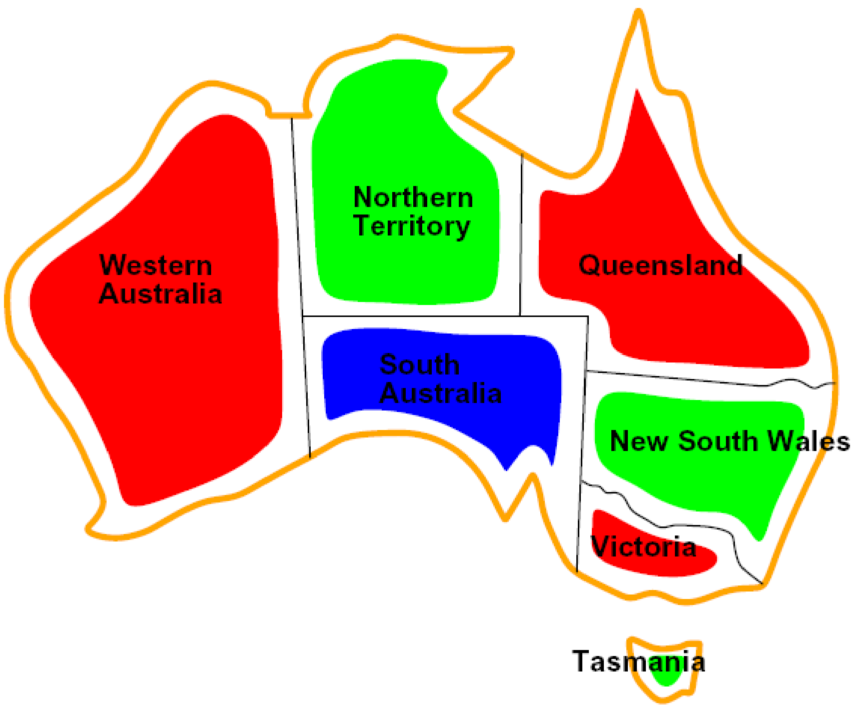
值域的集合$D = \{D_1, D_2, \ldots, D_n\}$，每个变量有自己的值域

约束的集合：描述变量取值的约束

问题的解：满足约束的所有变量分配

1. 约束满足问题

问题实例



地图着色

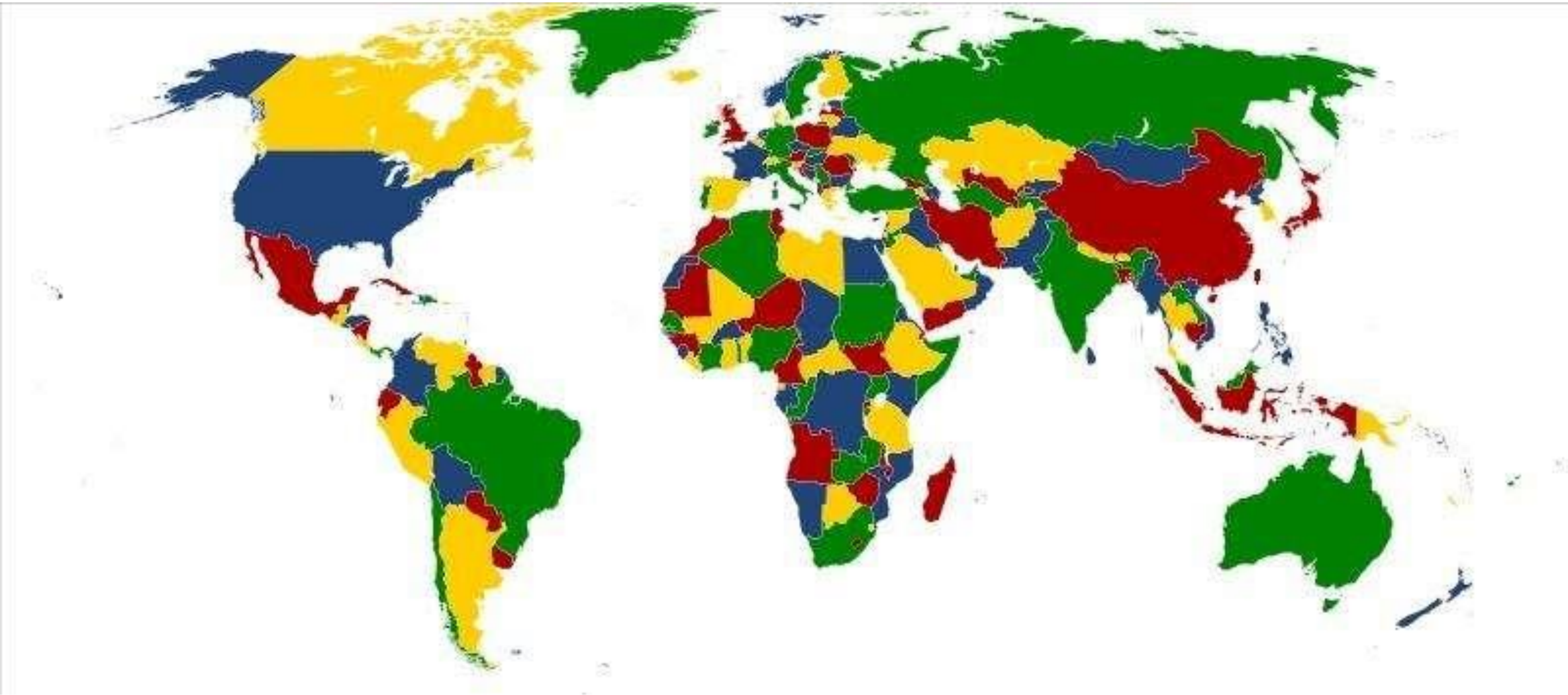
形式化描述

- 变量: WA，NT，Q，NSW，V，SA，T
- 值域: Color={red, green, blue}
- 约束条件: **相邻区域颜色不同**

  Color(WA)≠Color(NT)
- 满足条件的解：

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

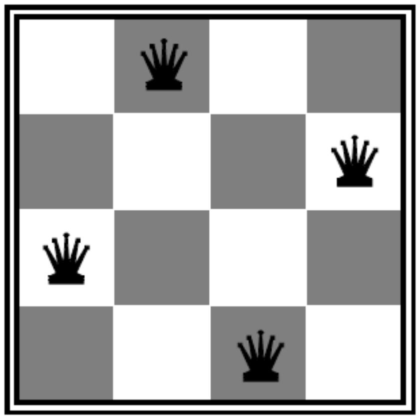
1. 约束满足问题



四色地图

1. 约束满足问题

问题实例



N皇后

形式化描述1

- 变量：X_{ij}
- 值域：$\{0,1\}$
- 约束条件

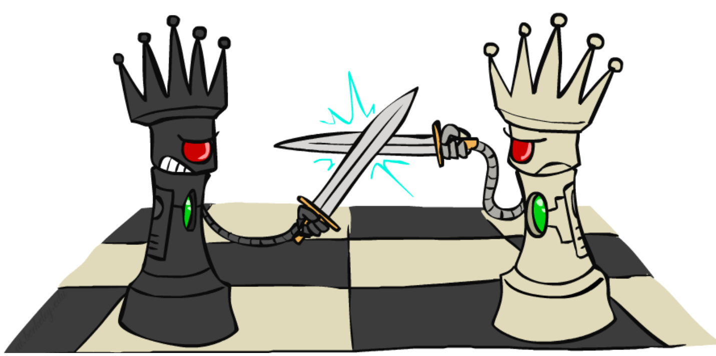
$$\sum_{i,j} X_{ij} = N$$

$$\forall i,j,k \quad (X_{ij}, X_{ik}) \in \{(0,0),(0,1),(1,0)\}$$

$$\forall i,j,k \quad (X_{ij}, X_{kj}) \in \{(0,0),(0,1),(1,0)\}$$

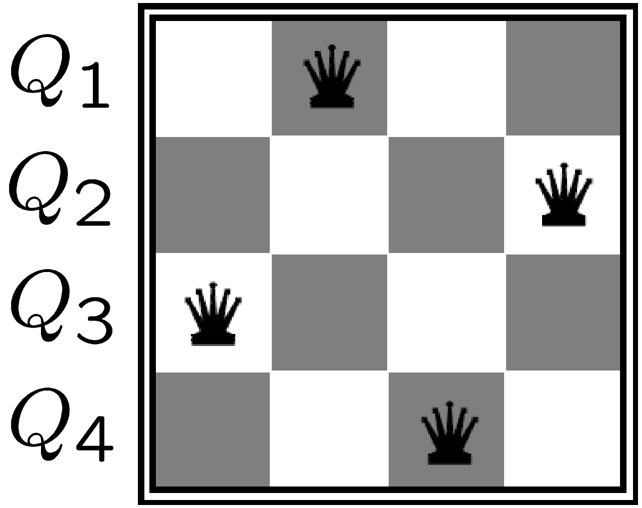
$$\forall i,j,k \quad (X_{ij}, X_{i+k,j+k}) \in \{(0,0),(0,1),(1,0)\}$$

$$\forall i,j,k \quad (X_{ij}, X_{i+k,j-k}) \in \{(0,0),(0,1),(1,0)\}$$



1. 约束满足问题

问题实例



形式化描述2

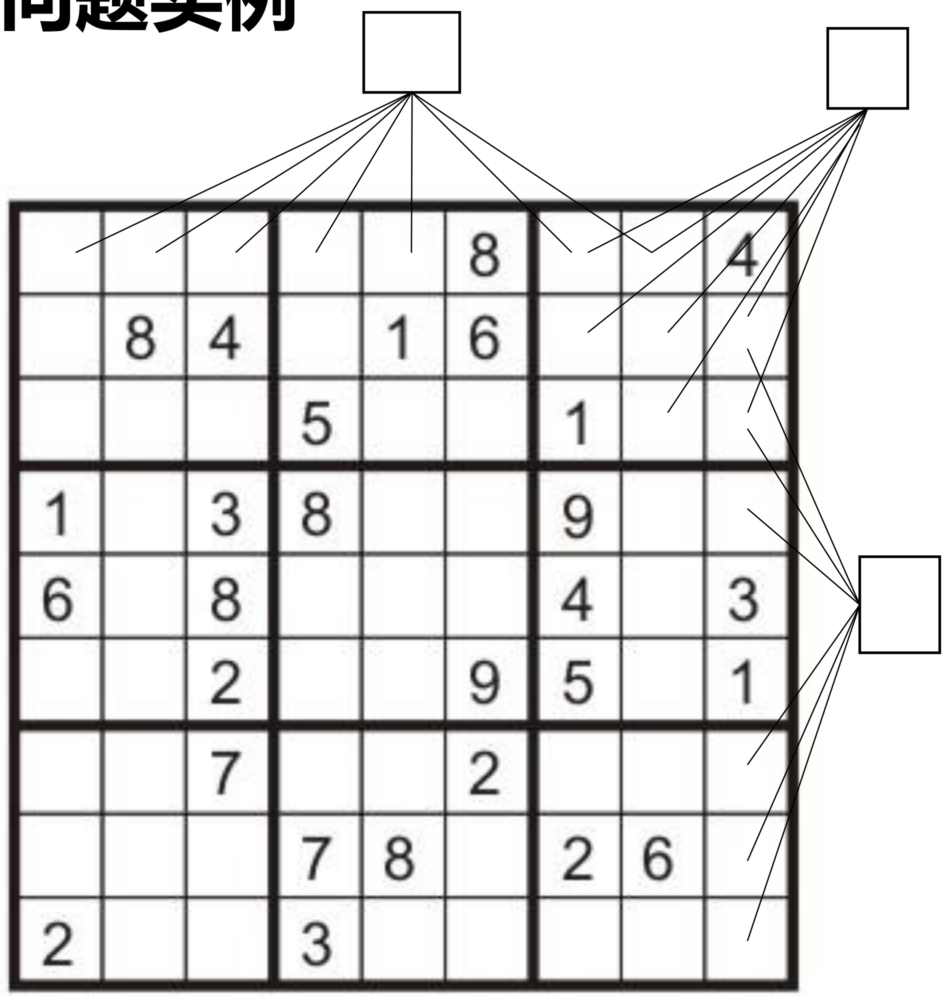
- 变量：Q_k
- 值域：$\{1, 2, 3, \dots, N\}$
- 约束条件

$$\forall i, j \;\; \text{non-threatening}(Q_i, Q_j)$$

$$(Q_1, Q_2) \in \{(1,3), (1,4), \dots\}$$

1. 约束满足问题

问题实例



形式化描述

- 变量：空白方格
- 值域：$\{1,2,3,\ldots,9\}$
- 约束条件

  每行9个不同数字

  每列9个不同数字

  每块9个不同数字

1. 约束满足问题

变化形式

变量变化

有限值域：大小为d意味着$O(d^n)$种可能

连续值域：如任务规划，变量为每个任务的起始/终止时间

约束变化

单变量的一元约束(值域缩小)，如SA ≠ green

成对变量的二元约束，如SA ≠ WA

涉及3个或更多变量的高阶约束

全局约束（所有变量约束Alldiff），如数独

约束偏好

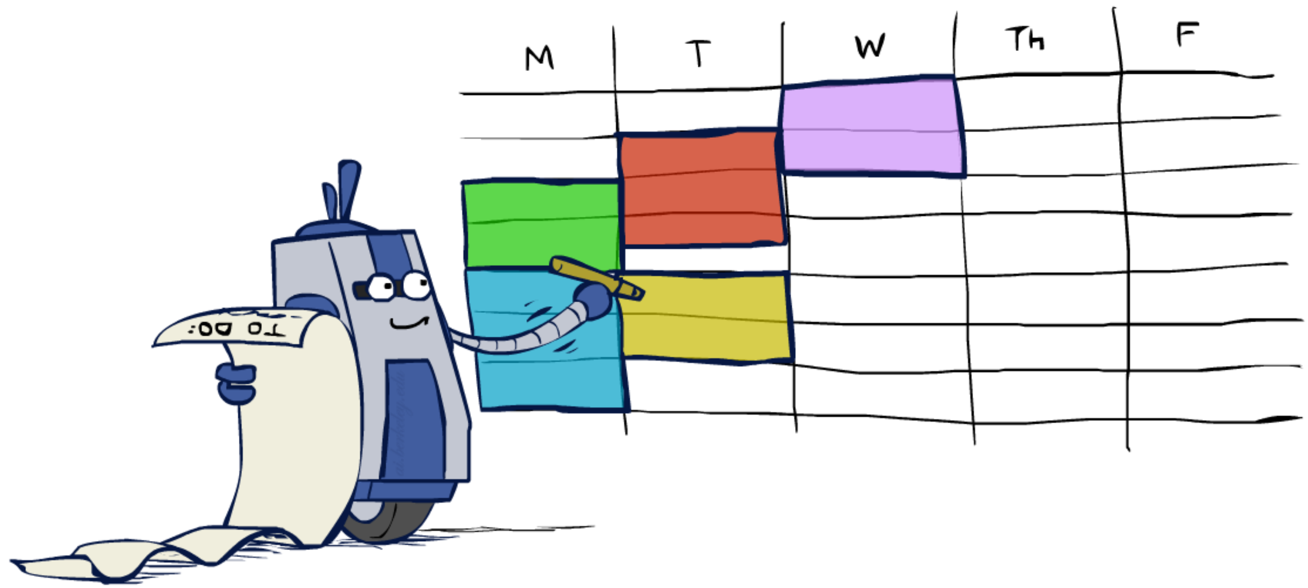
地图着色：红色比绿色更好

每个变量分配涉及不同代价

1. 约束满足问题

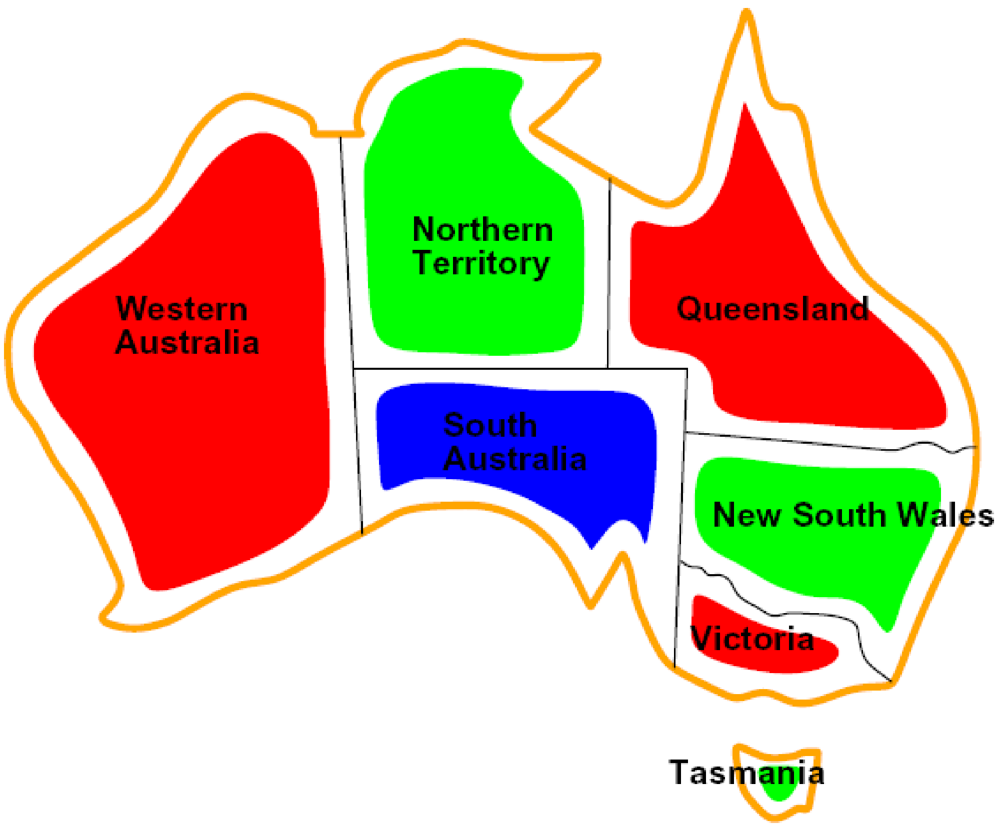
实际问题

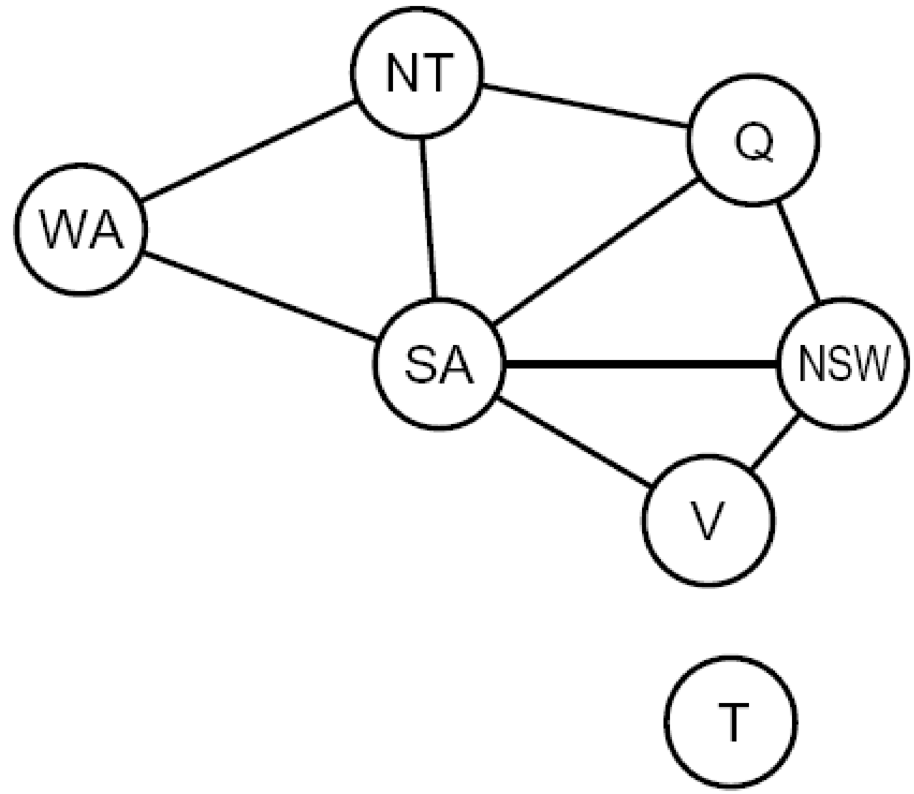
课程分配问题
排课问题
资源分配问题
产品装配
列车时刻表
工厂排班
等…



2. 回溯搜索求解

图着色-约束图





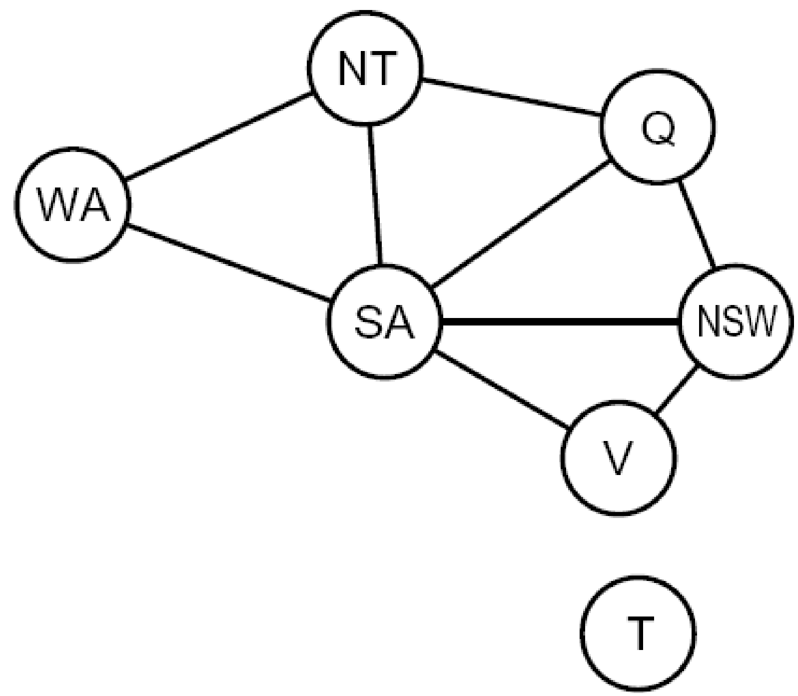
2. 回溯搜索求解

图着色-约束图

二元约束满足问题(CSP):每个约束最多与两个变量相关

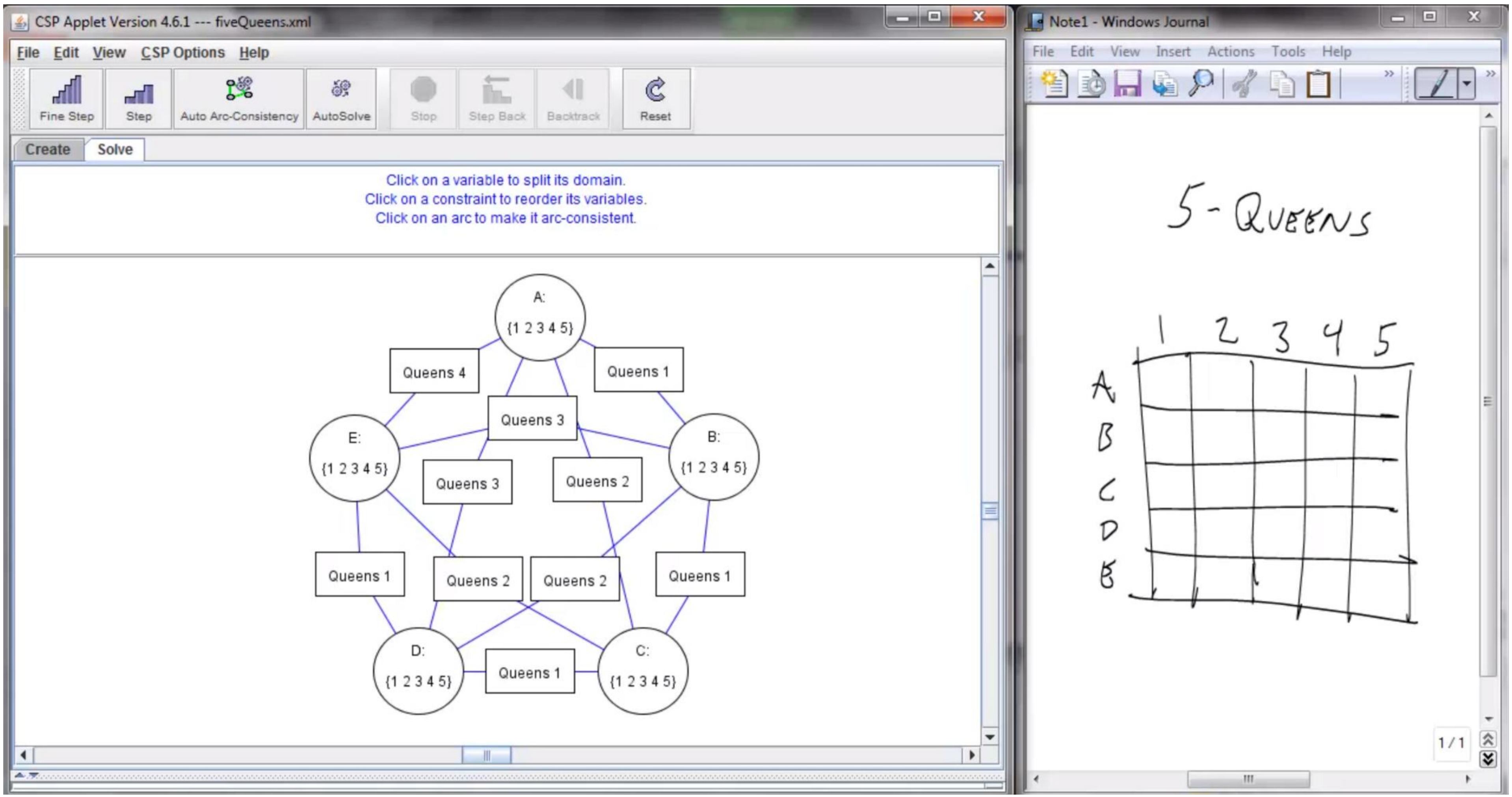
二元约束图：结点为变量，边表示约束

一般CSP算法利用图结构加速搜索



2. 回溯搜索求解

N皇后-约束图



2. 回溯搜索求解

算式谜-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：F, T, U, W, R, O值不同

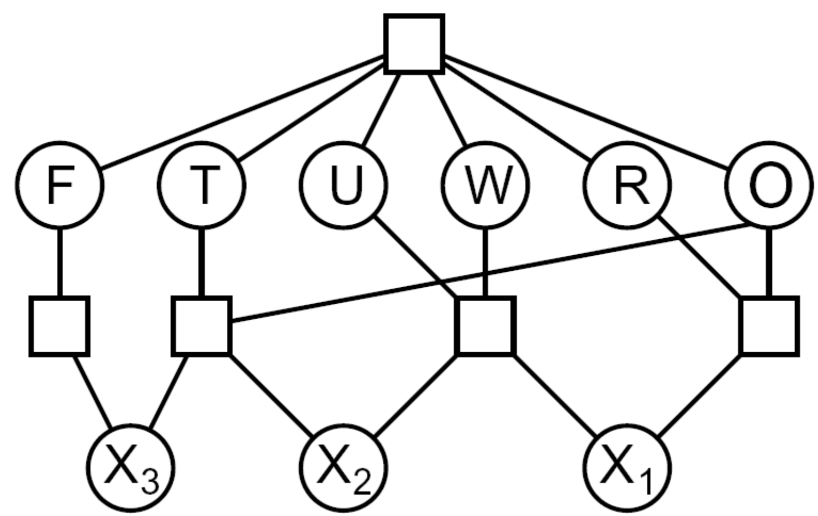
$$O + O = R + 10 \cdot X_1$$

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

$$T + T + X_2 = O + 10 \cdot X_3$$

$$F = X_3$$


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



2. 回溯搜索求解

状态：

初始状态：所有变量未分配{}

后继函数：对未分配的变量分配一个值

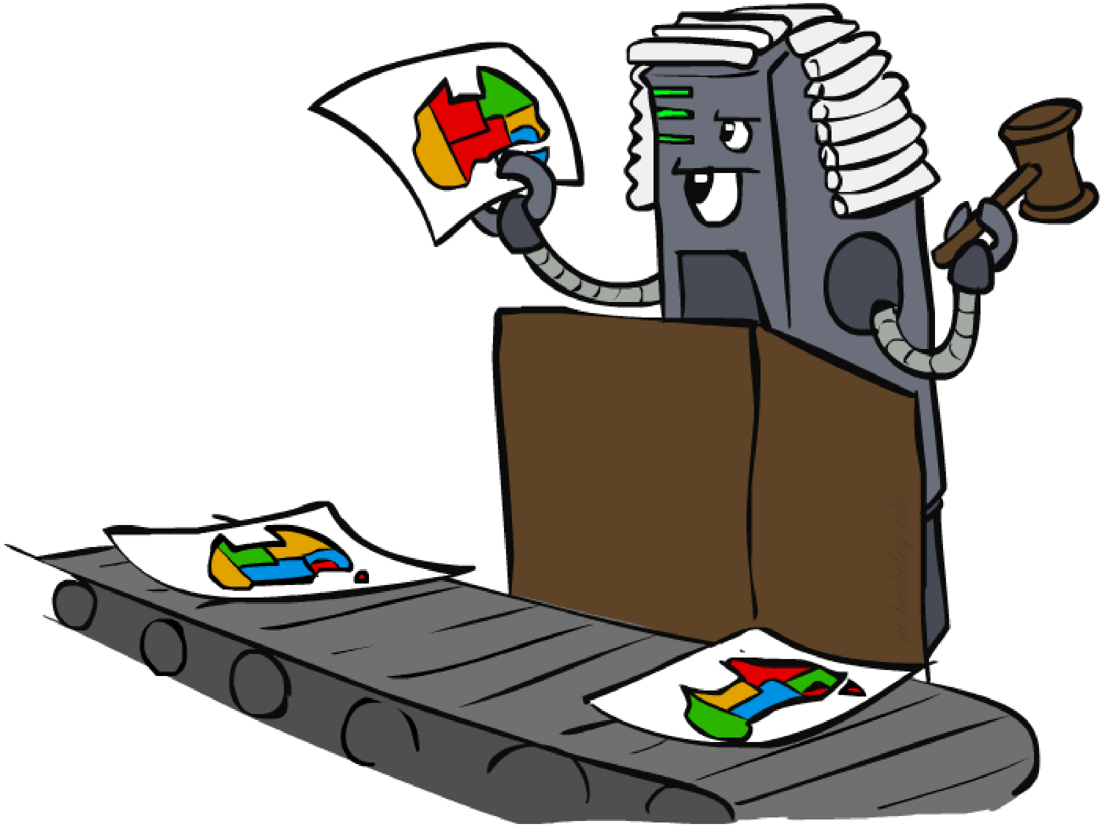
目标测试：变量都已分配值而且满足约束条件

搜索：

宽度优先?

深度优先?

简单搜索?



2. 回溯搜索求解

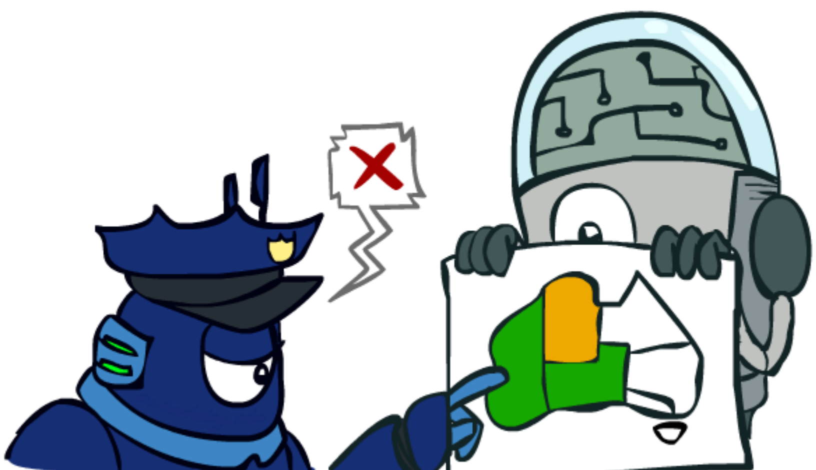
无信息搜索-回溯法

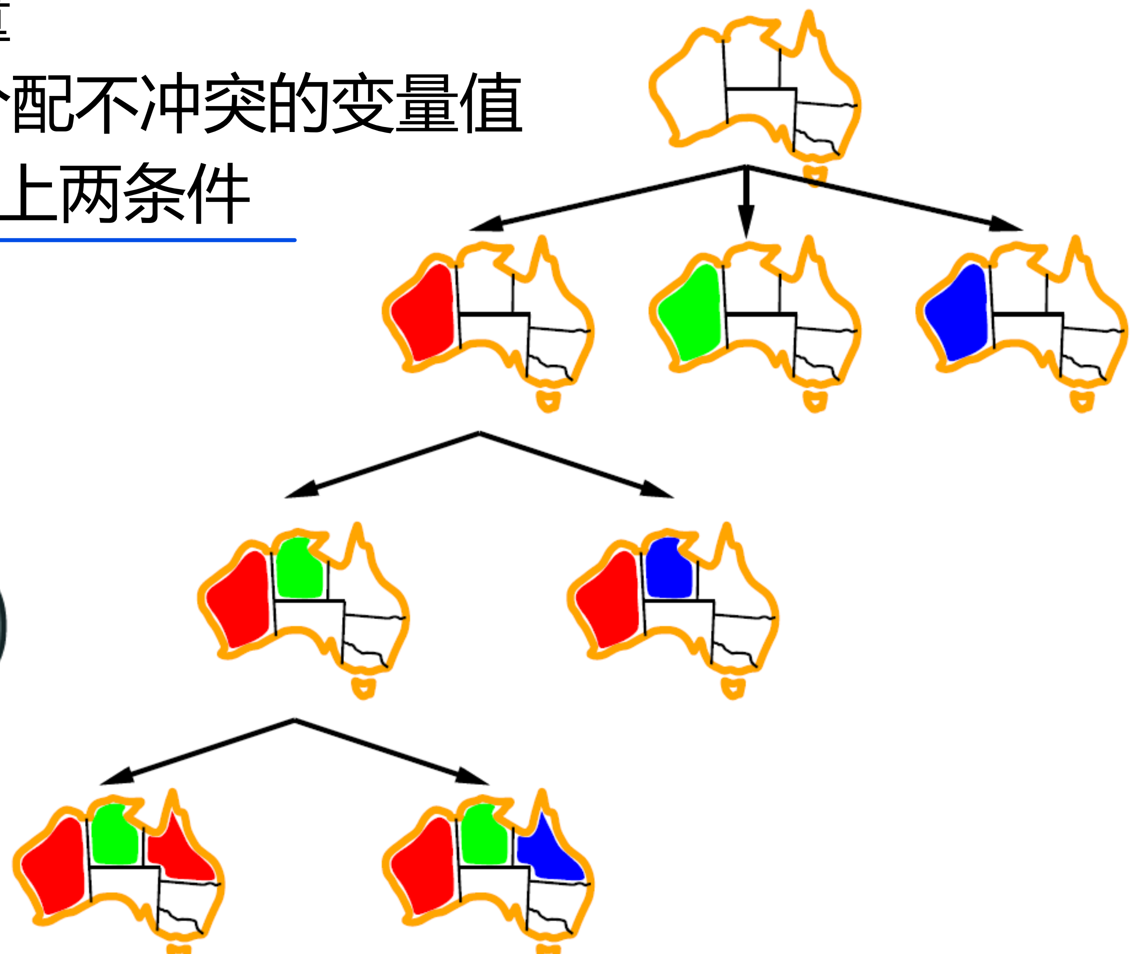
回溯搜索是求解约束满足问题最基本的无信息搜索算法

变量分配：每次一个变量

约束检查：考虑与前面分配不冲突的变量值

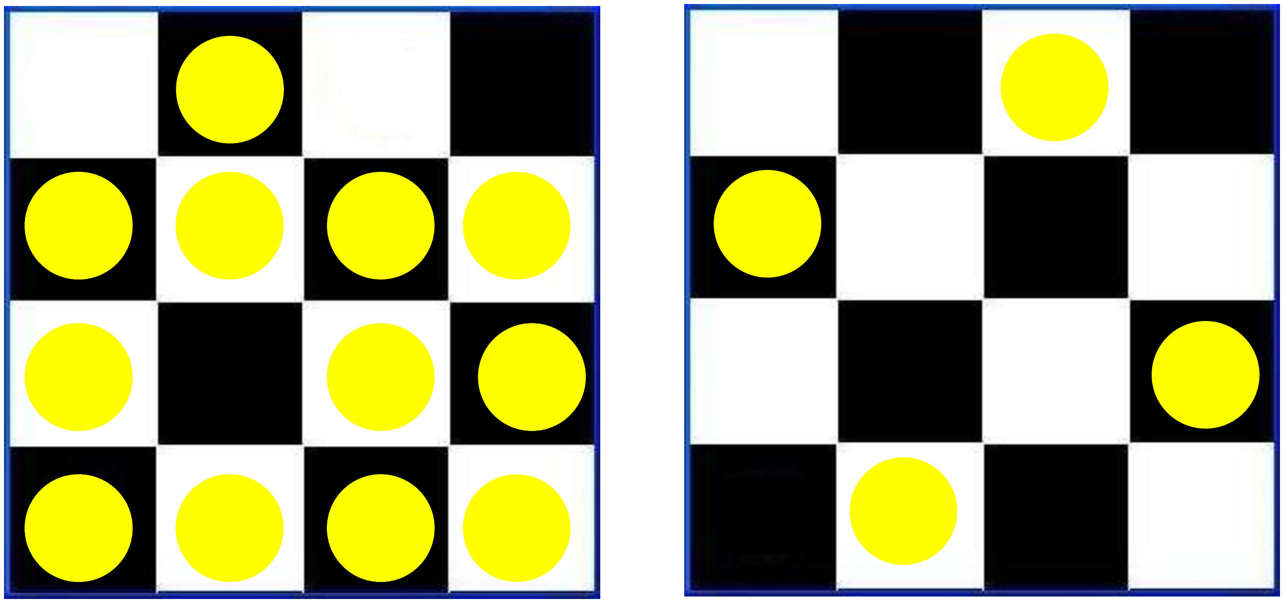
回溯搜索：深度优先+以上两条件





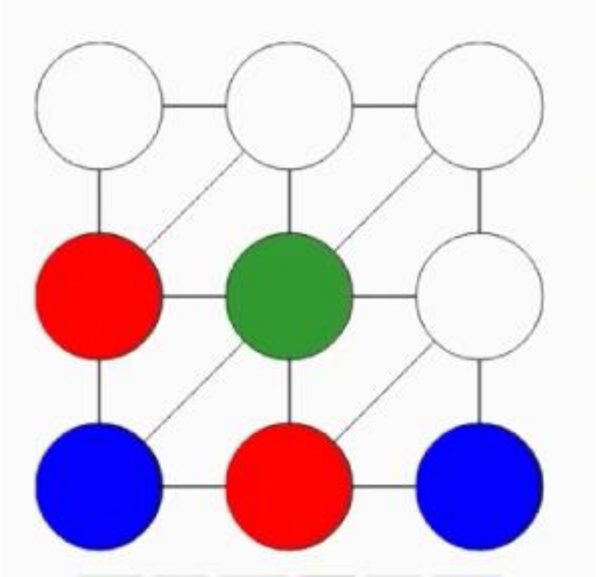
2. 回溯搜索求解

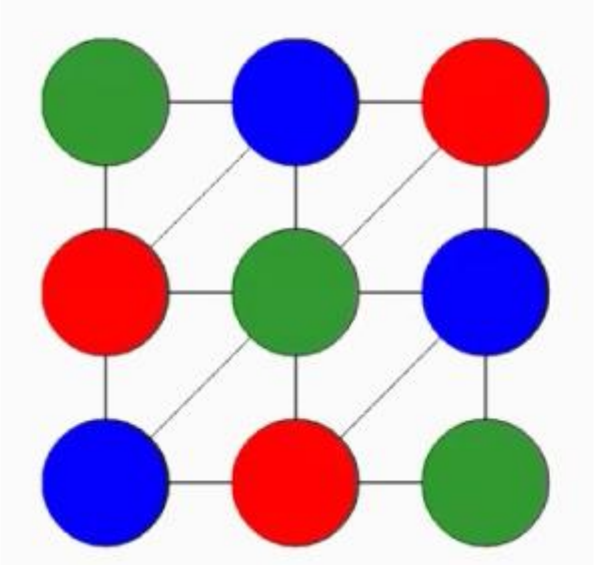
无信息搜索-回溯法



2. 回溯搜索求解

2. 回溯搜索求解





2. 回溯搜索求解

```
function BACKTRACK(csp, assignment ) returns a solution or failure
  if assignment is complete then return assignment
  var ←SELECT-UNASSIGNED-VARIABLE(csp, assignment )
  for each value in ORDER-DOMAIN-VALUES(csp, var, assignment ) do
     if value is consistent with assignment then
        add {var = value} to assignment
        inferences ←INFERENCE(csp, var , assignment )
        if inferences ≠ failure then
          add inferences to csp
          result ←BACKTRACK(csp, assignment )   递归
          if result ≠ failure then return result
          remove inferences from csp
        else remove {var = value} from assignment
return failure
```

3. 搜索优化

简单的启发式策略可大大提高速度

排序策略

如何决定下一个需分配的变量　SELECT-UNASSIGNED-VARIABLE

如何选择尝试分配的值　ORDER-DOMAIN-VALUES

推理策略

能否尽早的发现可避免的分配尝试　INFERENCE

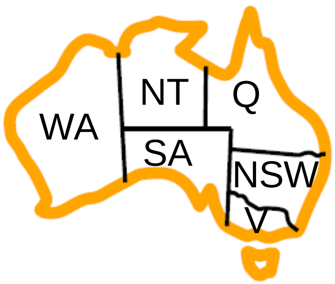
结构特性

能否利用问题结构特性　CSP

3. 搜索优化

变量排序-最少可取值（MRV）

变量排序：选择值域最少可取值的变量



该如何选择
下一个变量?

为什么不是最多值的变量?

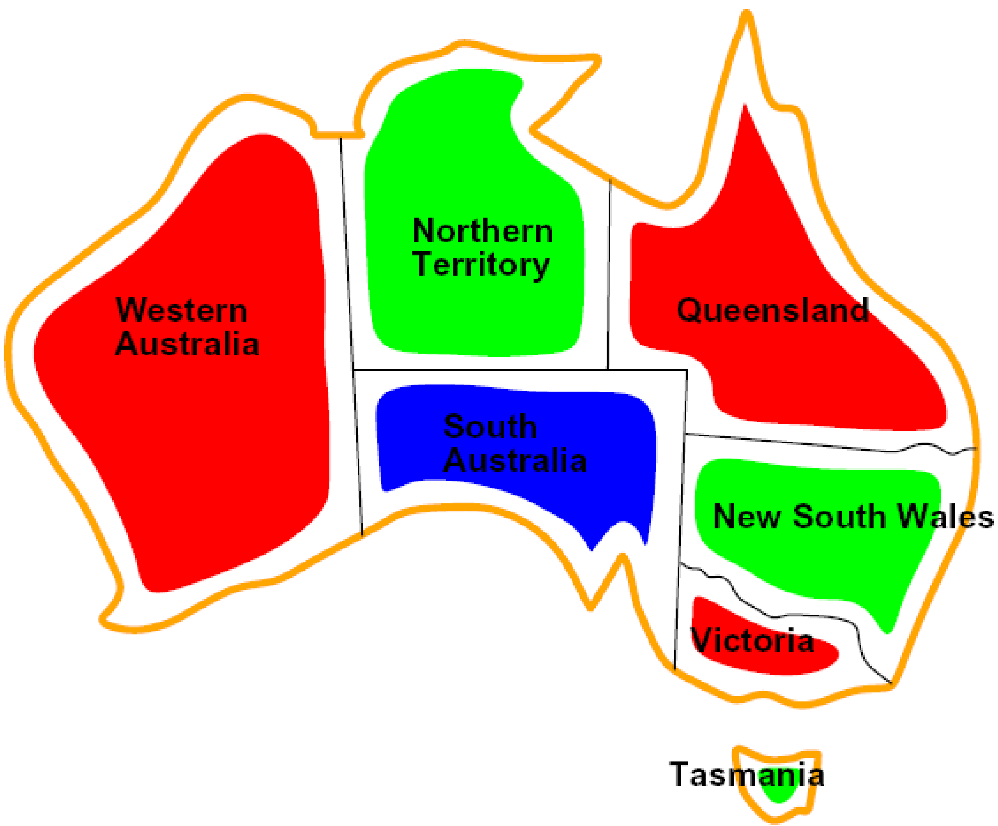
最少值变量也称为最多限制变量（变量空值，直接退出）

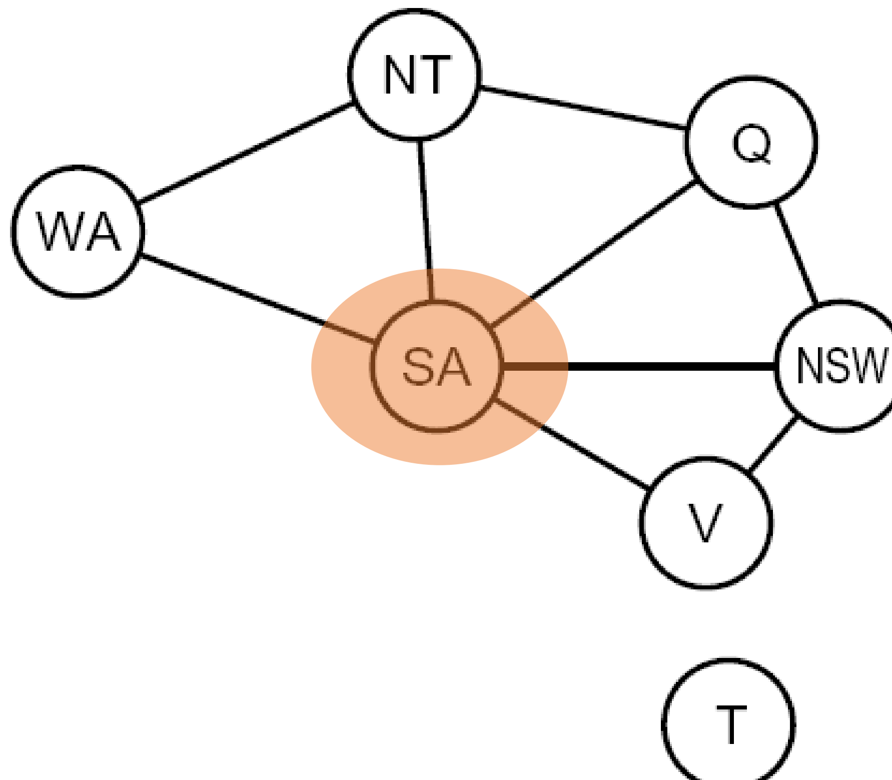
“失败-最快”排序-预剪枝，性能提升有时可达1000倍

3. 搜索优化

变量排序-度启发式

变量可取值数目相同时，选择**度较大**的变量

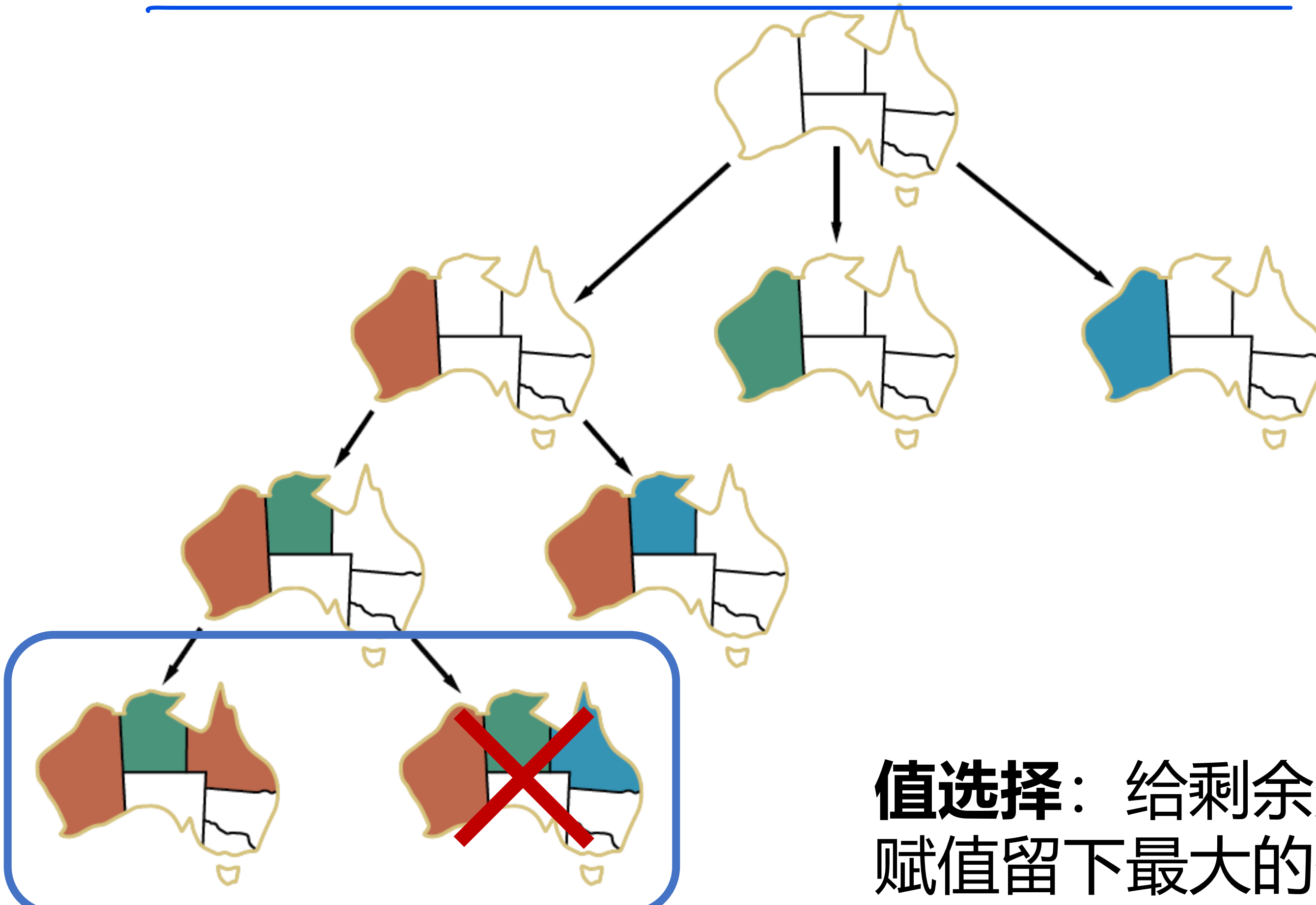




3. 搜索优化

值排序-最少约束值（MCV）

优先选择的值：给近邻变量留下更多选择的值



值选择：给剩余变量赋值留下最大的空间

3. 搜索优化

推理策略-约束传播

搜索过程进行**约束传播**的特殊**推理**-**局部相容性**。

使用约束来减小一个变量的合法取值范围，从而影响与此变量有约束关系的其它变量的取值。

约束传播与搜索可交替进行，亦可作为搜索前的预处理步骤。

结点相容

单个变量值域中的所有取值满足它的一元约束，则称此变量是**结点相容**的。

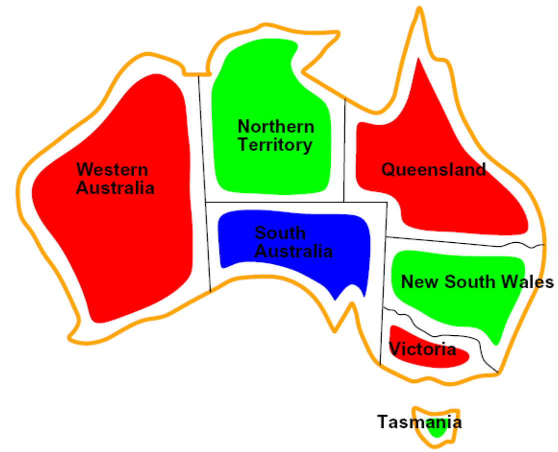
结点约束(偏好选择)：SA不喜欢绿色，则SA值域为{红色、蓝色}

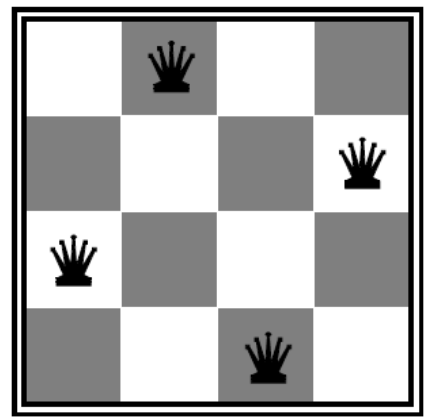
内容回顾

1 约束满足问题

2 回溯搜索求解

3 搜索优化





					8			4
	8	4		1	6			
			5			1		
1		3	8			9		
6		8				4		3
		2			9	5		1
		7			2			
			7	8		2	6	
2			3					

约束满足问题：

变量的集合$X = \{X_1, X_2, \dots, X_n\}$

值域的集合$D = \{D_1, D_2, \dots, D_n\}$，每个变量有自己的值域

约束的集合：描述变量取值的约束

问题的解：满足约束的所有变量分配

约束图

内容回顾

```
function BACKTRACK(csp, assignment ) returns a solution or failure
  if assignment is complete then return assignment
  var ←SELECT-UNASSIGNED-VARIABLE(csp, assignment )
  for each value in ORDER-DOMAIN-VALUES(csp, var, assignment ) do
     if value is consistent with assignment then
        add {var = value} to assignment
        inferences ←INFERENCE(csp, var , assignment )
        if inferences ≠ failure then
           add inferences to csp
           result ←BACKTRACK(csp, assignment )
           if result ≠ failure then return result
           remove inferences from csp
        else remove {var = value} from assignment
return failure
```

MRV

MCV

CP

3. 搜索优化

推理策略-约束传播

边相容

某变量值域中的所有取值满足它的所有二元约束，则称此变量是**边相容**的。

对于变量X和Y，若对变量X的每个取值X_i在变量Y中都存在Y_j满足边(X_i, Y_j)的二元约束，则称X相对Y是边相容的。

如果每个变量相对其他变量都是边相容的，则称该网络是边相容的。

路径相容

边相容通过边（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。

两变量X和Y对于第三个变量Z是相容的，指对每一个相容赋值(X_i, Y_j)，Z都有合适的取值使得(X_i, Z_k)和(Y_j, Z_k)是相容的。

3. 搜索优化

推理策略-约束传播

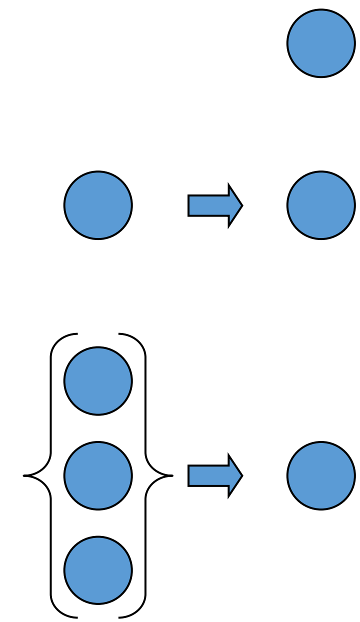
1阶相容(结点相容):每个单变量值域的值都满足该变量的一元约束

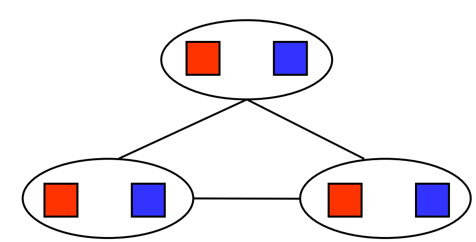
2阶相容(边相容):两变量取值满足约束

3阶相容(路径相容):三变量取值满足约束

K阶相容:对于每k个变量，可以将对k-1的任何相容分配扩展到第k个变量

k越大计算成本越高





3. 搜索优化

推理策略-约束传播

强K阶相容：

强k阶相容：k-1，k-2，…1也一致

注意：强k阶相容意味着无需回溯就可求解问题

为什么?

- 选择任何变量的任何赋值
- 选择一个新变量
- 通过2阶相容，可以选择与第一个相容
- 选择一个新变量
- 通过3阶相容，可以选择与前2个相容
- …

3. 搜索优化

推理策略-约束传播

```
function AC-3( csp) returns the CSP, possibly with reduced domains
   inputs: csp, a binary CSP with variables {X1, X2, ..., Xn}
   local variables: queue, a queue of arcs, initially all the arcs in csp

   while queue is not empty do
      (Xi, Xj) ← REMOVE-FIRST(queue)
      if REMOVE-INCONSISTENT-VALUES(Xi, Xj) then
         for each Xk in NEIGHBORS[Xi] do
            add (Xk, Xi) to queue
```

```
function REMOVE-INCONSISTENT-VALUES( Xi, Xj) returns true iff succeeds
   removed ← false
   for each x in DOMAIN[Xi] do
      if no value y in DOMAIN[Xj] allows (x,y) to satisfy the constraint Xi ↔ Xj
         then delete x from DOMAIN[Xi];  removed ← true
   return removed
```

边

边相容算法

时间复杂度从$O(n^2d^3)$降至$O(n^2d^2)$

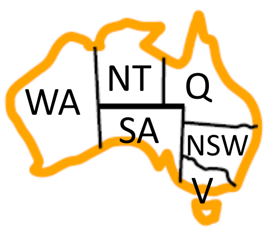
寻找出所有可能解是NP问题

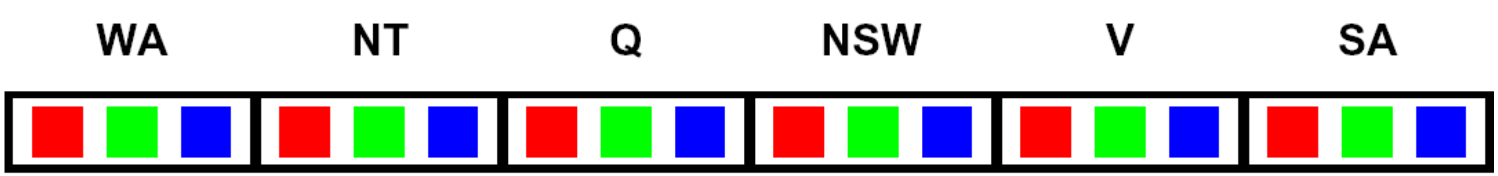
3. 搜索优化

推理策略-前向检查

排除：跟踪未分配变量的值域并去除错误的选项

前向检查：将违反约束的现有已分配变量值去掉

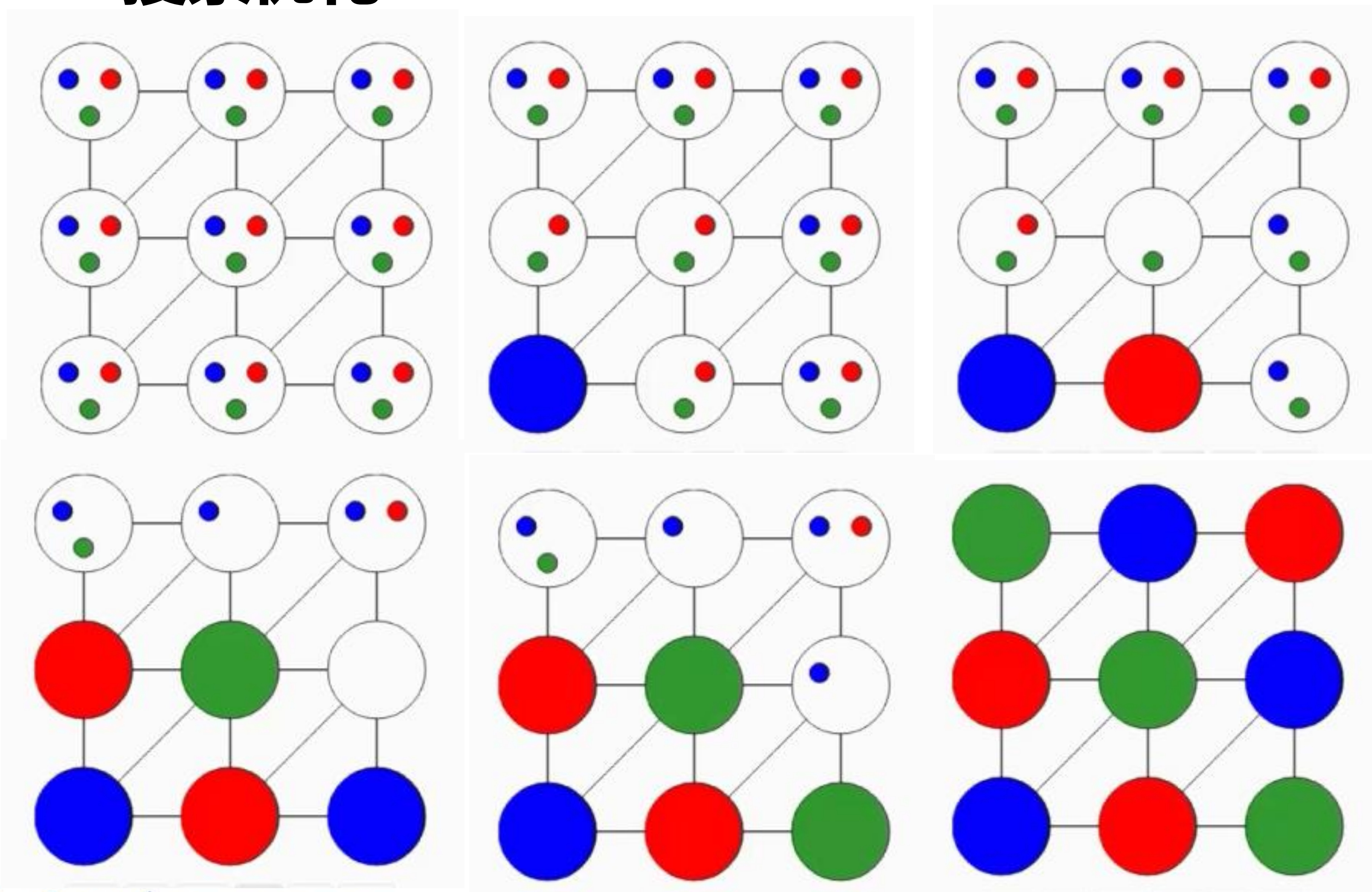




3. 搜索优化

排除策略-前向检查

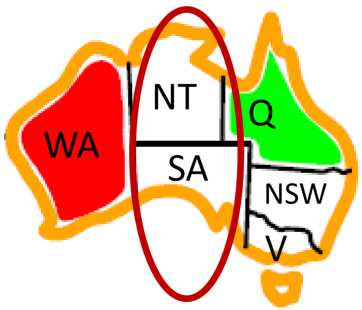
3. 搜索优化

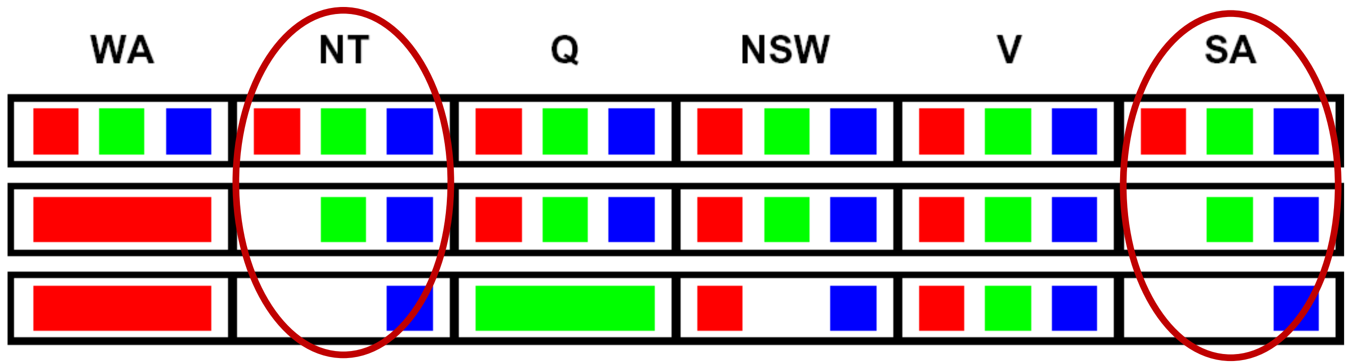


3. 搜索优化

推理策略-前向检查

前向检查传播分配变量的信息到未分配的变量，但不能提前检测错误选项。





NT和SA不可能同时为蓝色

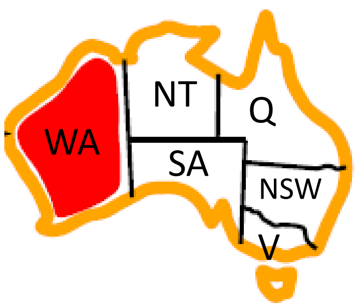
为什么检测不出来?

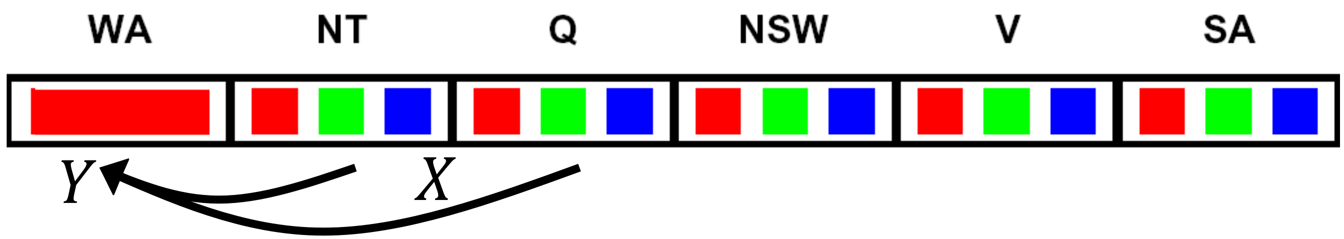
约束传播：约束到约束的推理

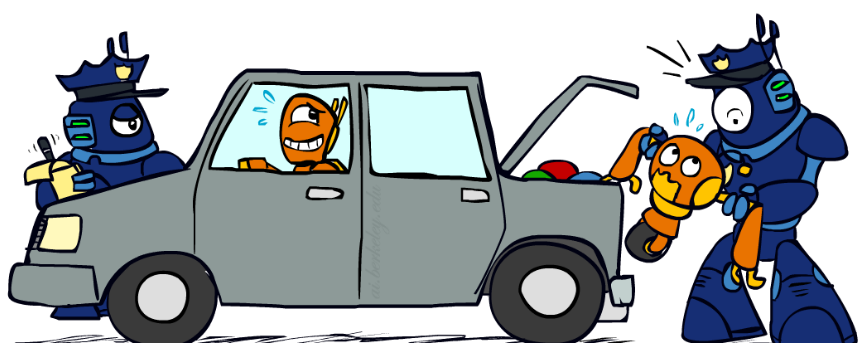
3. 搜索优化

推理策略-约束传播

边$X \rightarrow Y$是相容的当且仅当在尾部的每个x都有一个y，可以在不违反约束的情况下对其进行分配







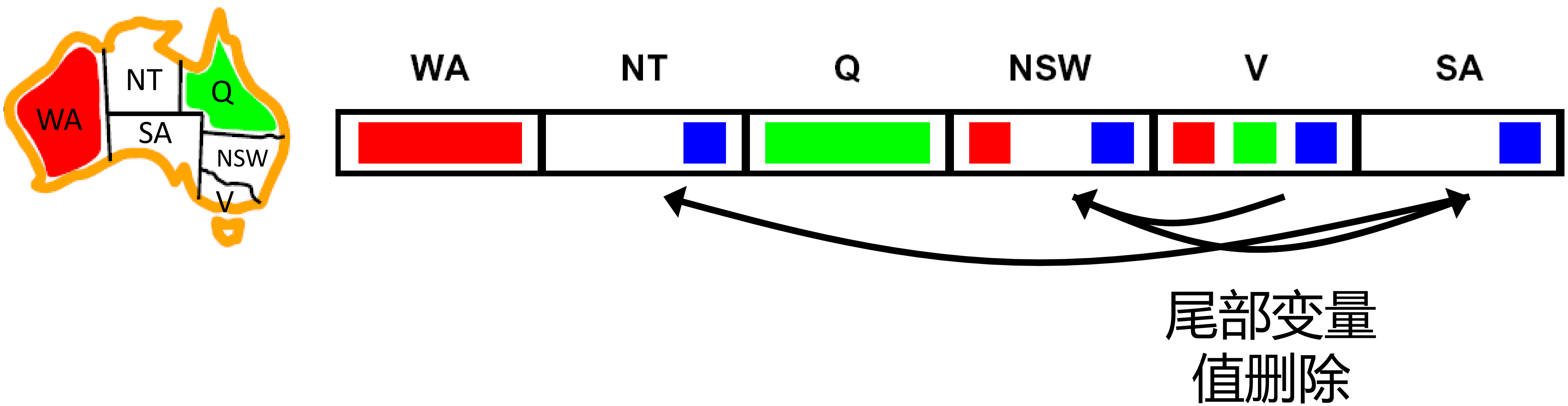
尾部变量值删除

前向检查：强化指向每个新分配的边的相容性

3. 搜索优化

推理策略-约束传播-边相容

简单形式的传播需保证所有边相容



重点：如果X去除某个值，X的近邻需再次检查

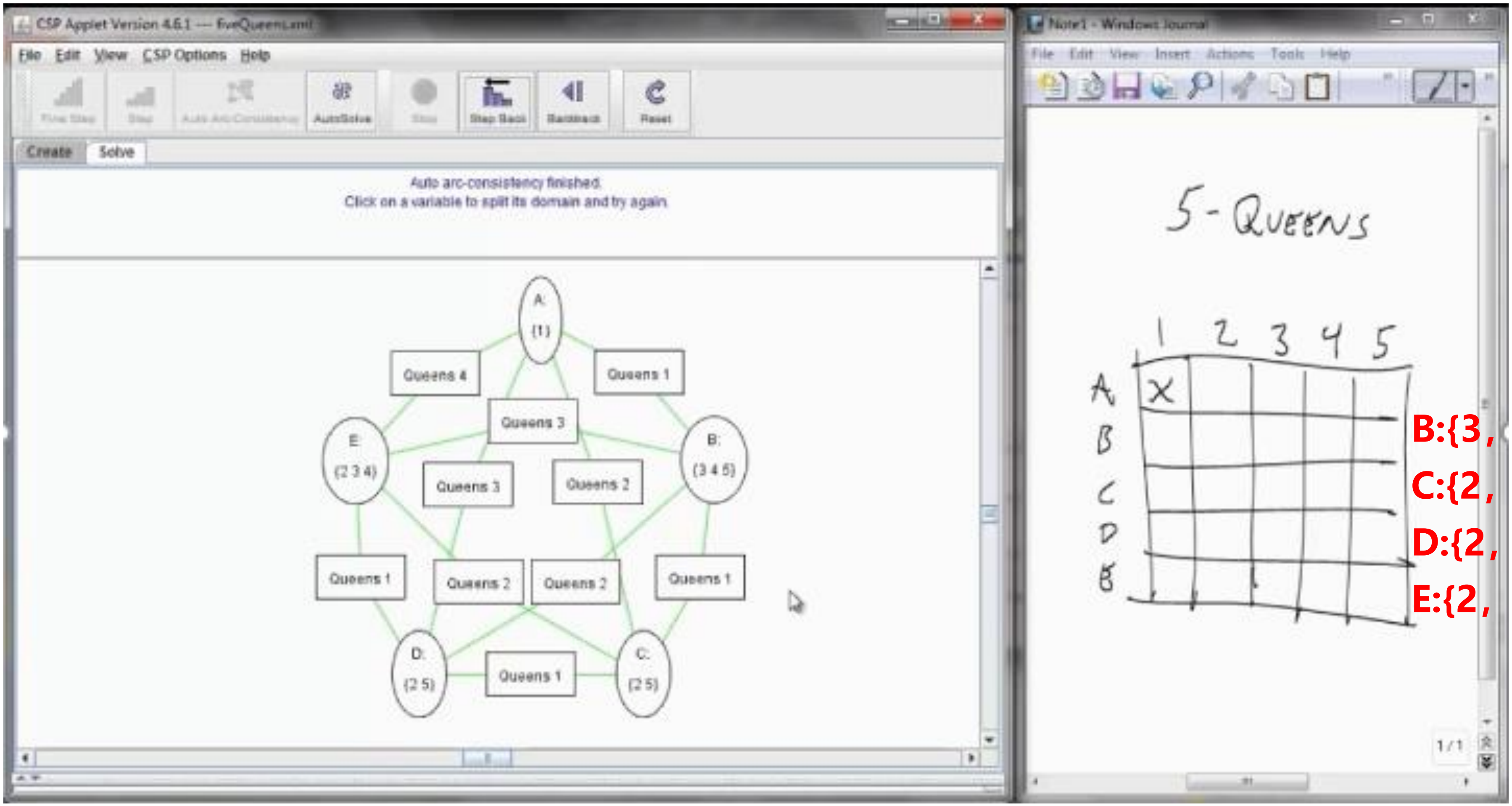
边相容比前向检查能**更早**检测出错误

3. 搜索优化

推理策略-约束传播-边相容

3. 搜索优化

推理策略-约束传播-边相容



B:{3，4，5}

C:{2，4，5}

D:{2，3，5}

E:{2，3，4}

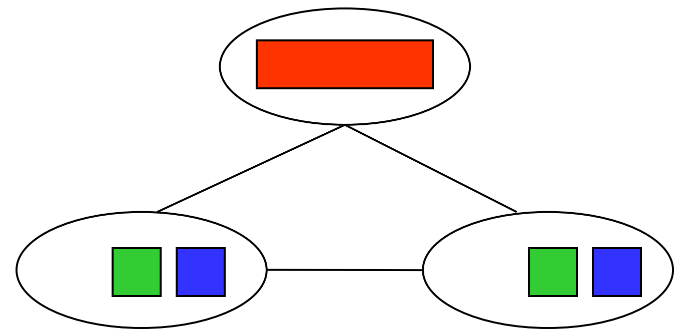
3. 搜索优化

推理策略-约束传播-边相容

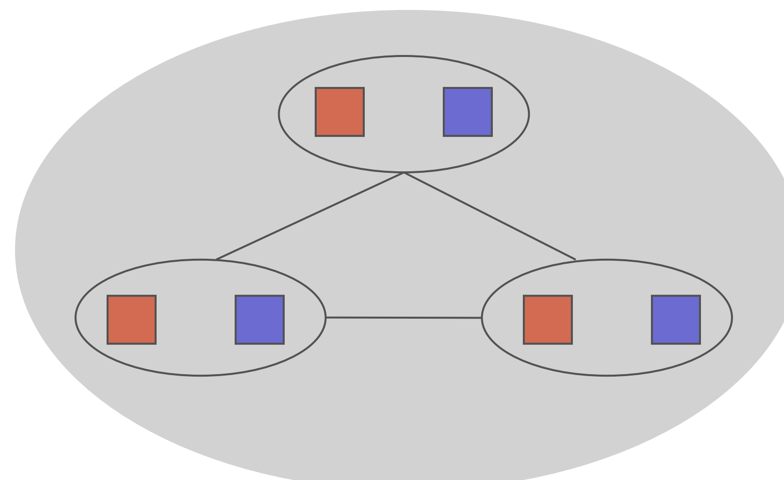
局限性

强化边相容后情况:

- 有唯一的解
- 有多个解
- 也有可能无解



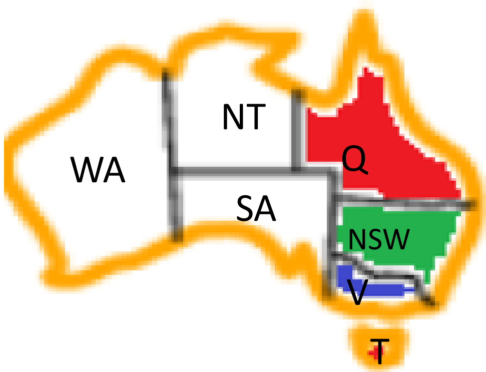
边相容性嵌入在回溯搜索



3. 搜索优化

推理策略-智能回溯

搜索失败处理原则：返回前一个变量，尝试另一个值（时序回溯）。



当前状态

SA着色 → 无解回溯

回溯到哪个变量？

按照顺序回溯到T，但回溯到T不能解决冲突

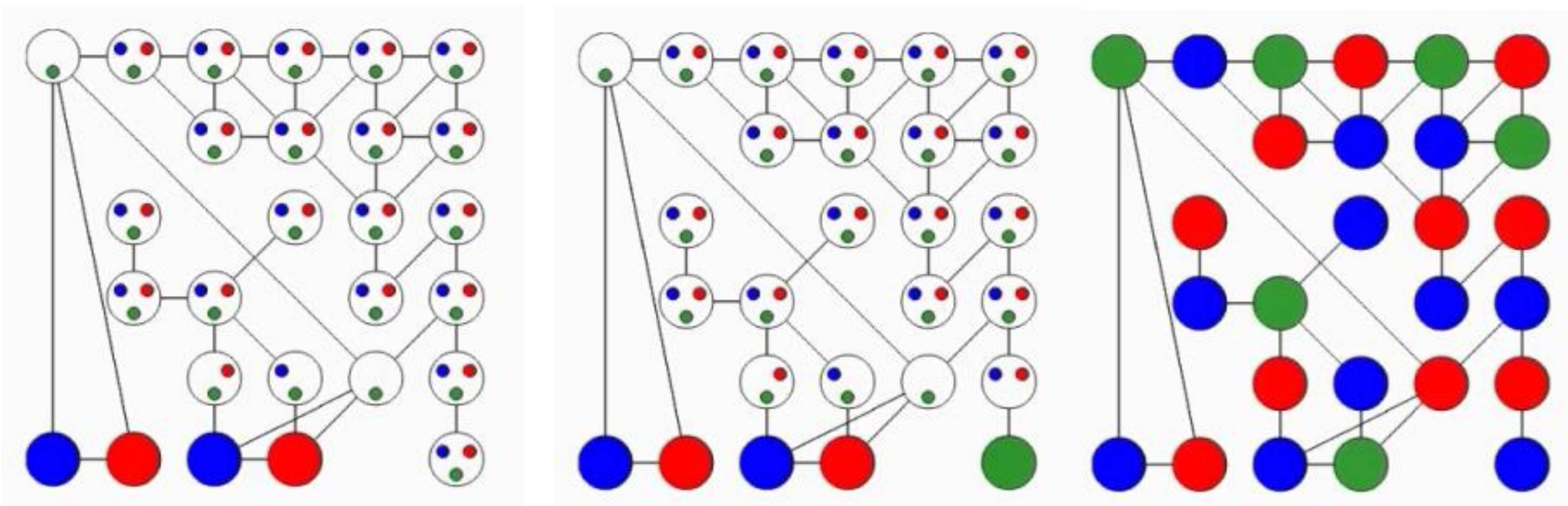
方案：建立冲突集，回溯到冲突集中时间最近的赋值，SA的冲突集为{Q,NSW,V}，所以回溯到V

3. 搜索优化

回溯法-前向检查

3. 搜索优化

回溯法-前向检查



3. 搜索优化

回溯法-边相容

3. 搜索优化

问题结构特性

极端情况：大问题由独立子问题构成

例子：Tasmania和大陆相互独立

独立的子问题可认为是约束图的连接组件（连通子图）

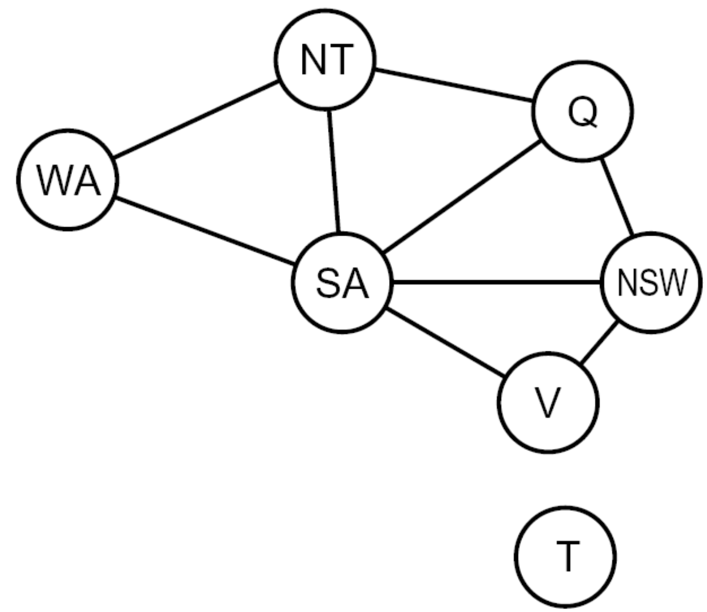
假设：n个变量的图分解为仅c个变量的子问题

最坏情况下的解决方案成本为$O((n/c)d^c)$，关于n线性复杂度

例：$n=80, d=2, c=20$

2^{80} = 40亿年，一千万个节点/秒

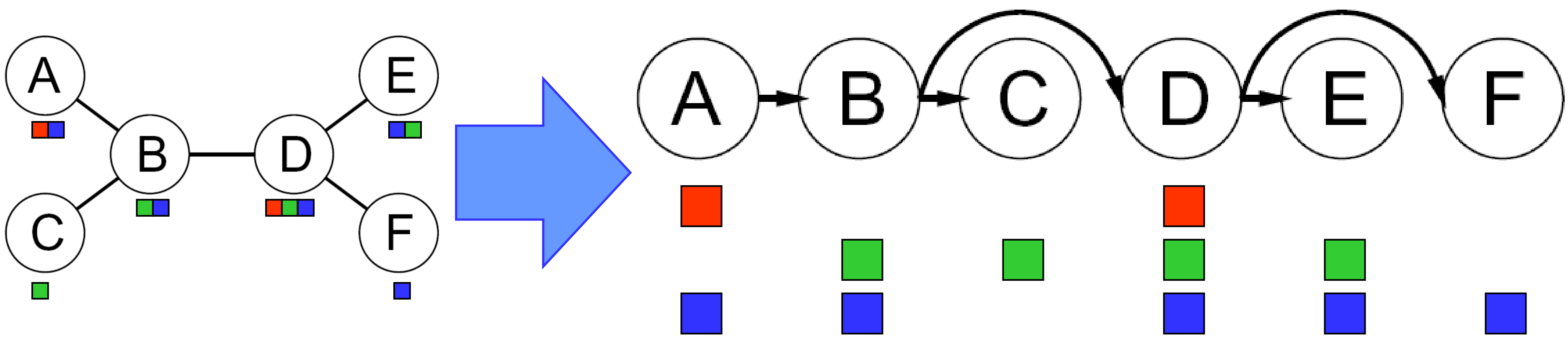
$(4)(2^{20})$ = 0.4秒，一千万个节点/秒



3. 搜索优化

树状约束满足问题-图转树

变量（拓扑排序）：选择一个根变量，对变量进行排序，先父母后孩子。



两变量最多有一条路径，所以n个结点的树有$n-1$条边。

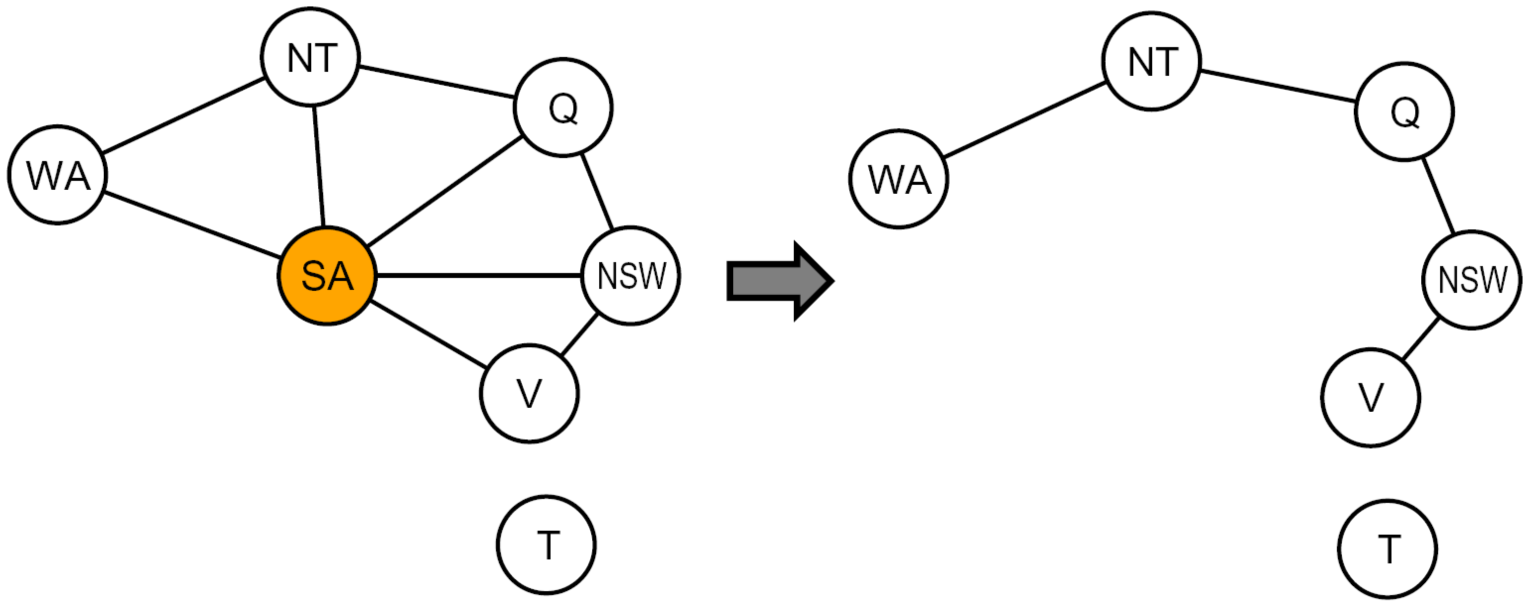
边相容：应用RemoveInconsistent$(Parent(X_i), X_i)$

变量分配：对于$i = 1: n$，相容地分配X_i与$Parent(X_i)$

父结点无论取什么值，子结点都有值可选，无需回溯，线性时间复杂度

3. 搜索优化

图转树-近似树状的CSP



基本思想：实例化一变量，修剪其近邻（边相容）

环割集条件：实例化一组变量，以使其余约束图成为一棵树

大小为c的割集时间复杂度$O((dc)(n-c)d^2)$，c较小非常快

3. 搜索优化

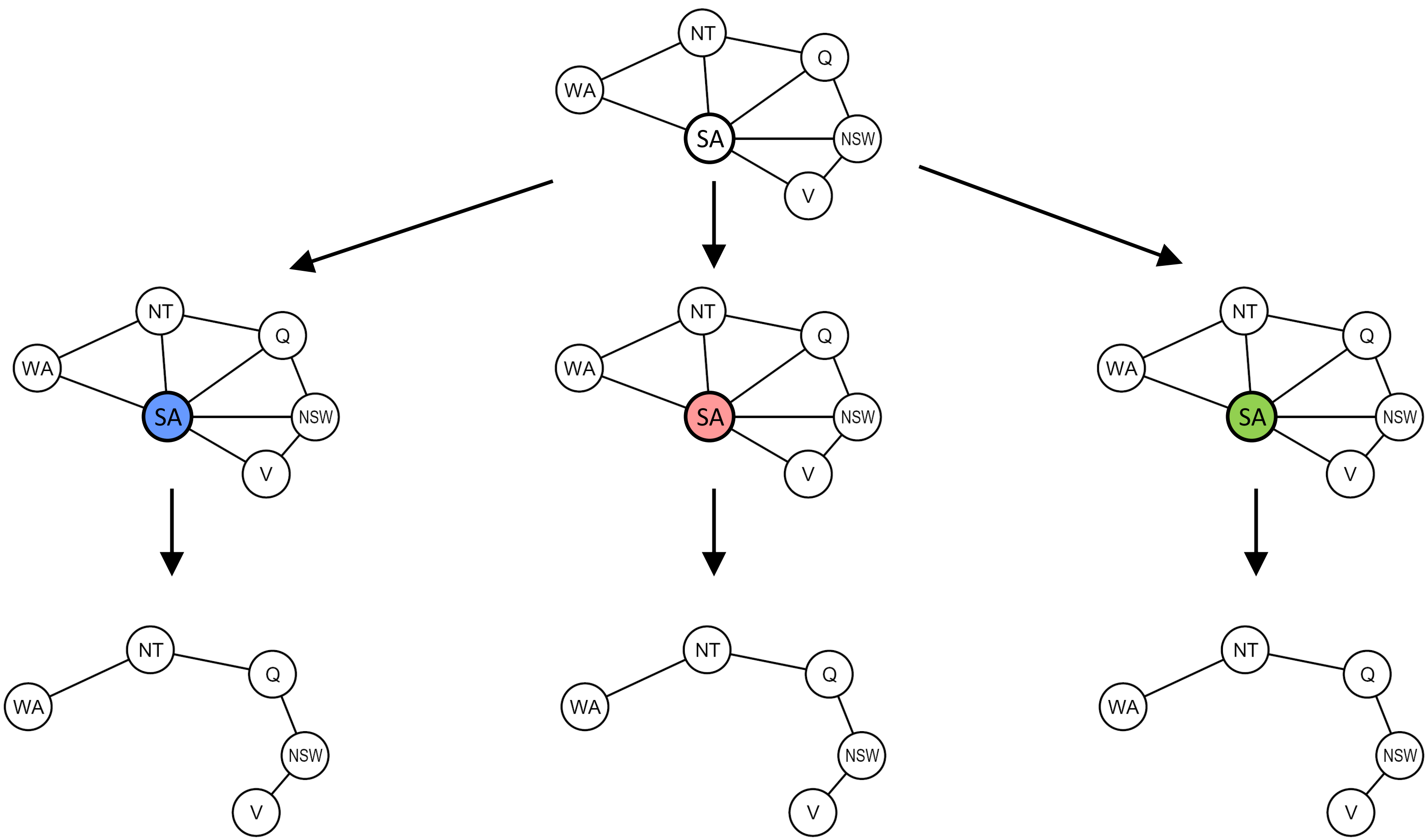
割集条件

选择一割集

实例化环割集
(所有可能值)

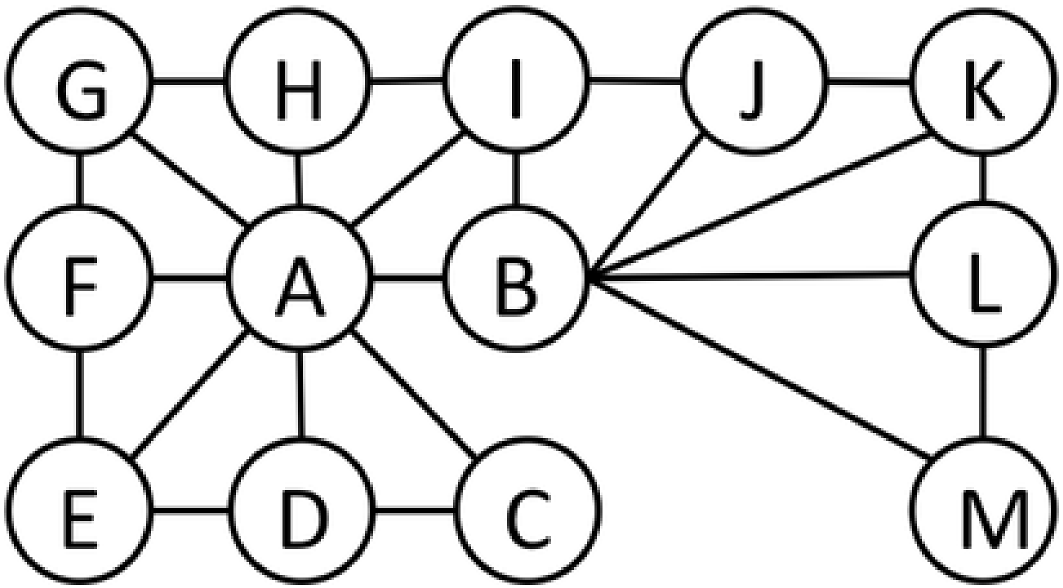
对剩余的CSP
进行分配

解决剩余的CSP
(树状结构)



3. 搜索优化

试一试：寻找最小环割集



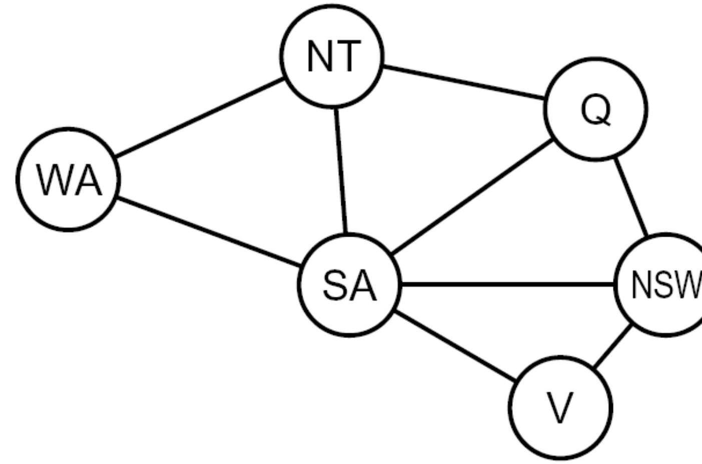
3. 搜索优化

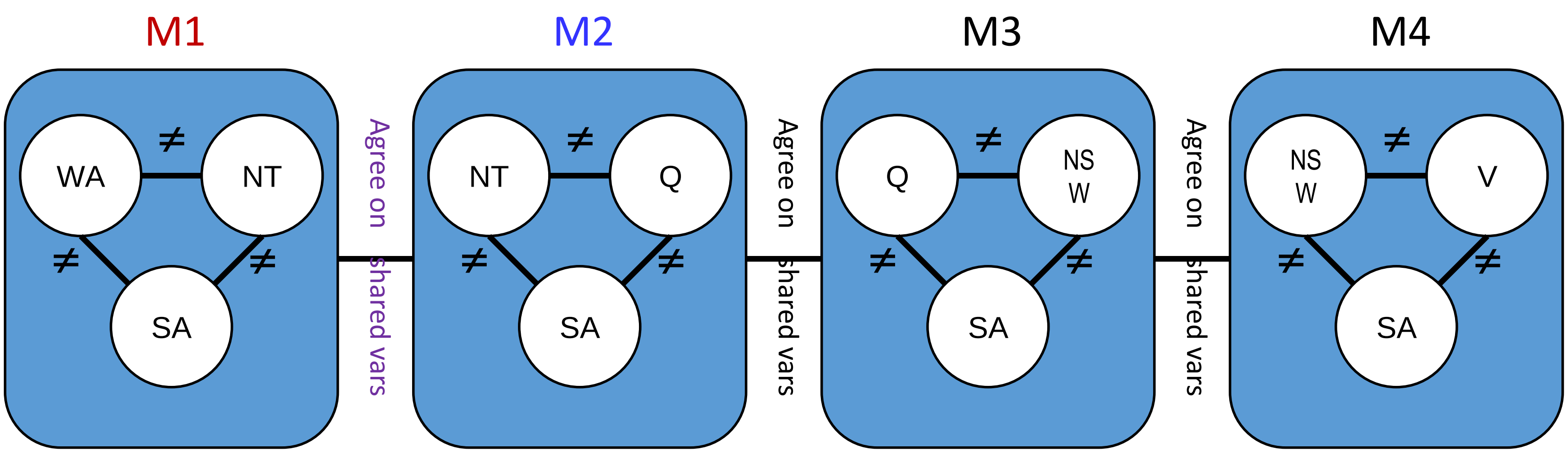
树分解

思想：创建变量的树形图

每个变量编码原始CSP的一部分

子问题重叠以确保一致的解决方案





{(WA=r,SA=g,NT=b),
(WA=b,SA=r,NT=g),
…}

{(NT=r,SA=g,Q=b),
(NT=b,SA=g,Q=r),
…}

Agree: (M1,M2) ∈
{((WA=g,SA=g,NT=g), (NT=g,SA=g,Q=g)), …}

4. 局部搜索

问题背景：某些问题只关注**解状态**，而不是系统地探索从初始状态开始的路径（不关心怎么到达目的状态）。

局部搜索：不关心路径，从单个当前状态节点（而不是多条路径）出发，通常只移动到它的邻近状态。一般情况下不保留搜索路径。

局部搜索优点：

1）通常只用常数级的内存；

2）通常能在标准约束满足算法不适用的很大或无限的（连续的）状态空间中找到合理的解。

此外，局部搜索算法对于解决纯粹的最优化问题十分有用，其目标是根据目标函数找到最佳状态。

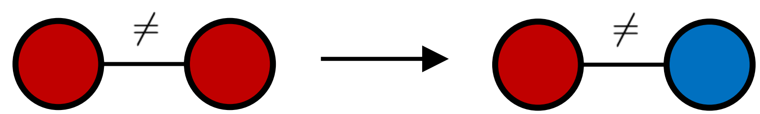
如果存在解，最优的局部搜索算法可找到全局最大/最小值

4. 局部搜索

局部搜索方法通常基于“完整”状态，即每个变量都赋一个值，搜索过程一次改变一个变量的取值。

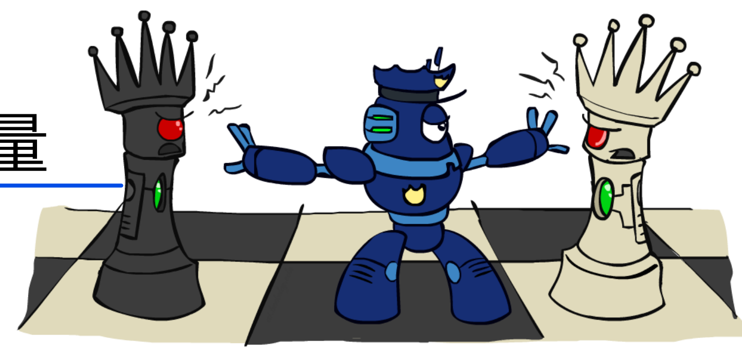
CSP局部搜索：

- 初始变量分配违反一些约束
- 重新分配变量值



基本思想：

变量选择：随机选择任何有冲突的变量

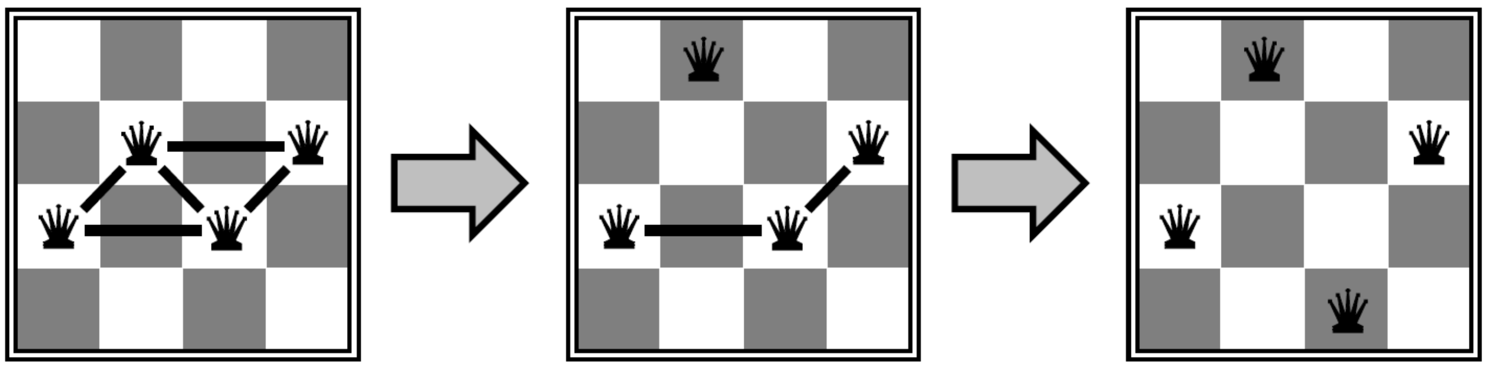


值选择：最小冲突启发式

- 选择一个违反最少约束的值
- 爬山法—启发式信息$h(n)$ = 违反约束的总数
- 其它局部搜索方法：模拟退火、局部束搜索、遗传算法

4. 局部搜索

实例：4皇后



h = 5　　h = 2　　h = 0

状态：4列4个皇后($4^4 = 256$状态)

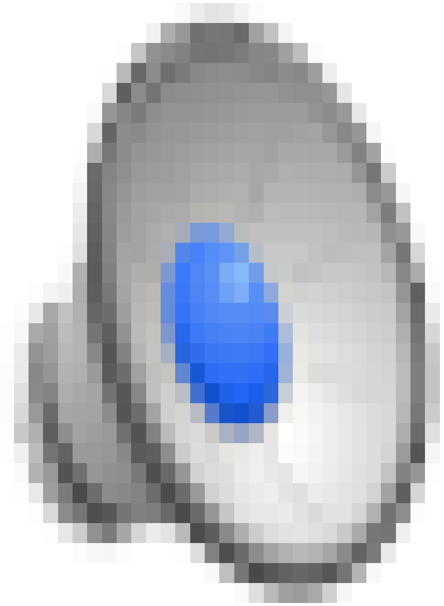
操作：在每列移动皇后

目标测试：无攻击

评价函数：$c(n)$ =攻击的数目

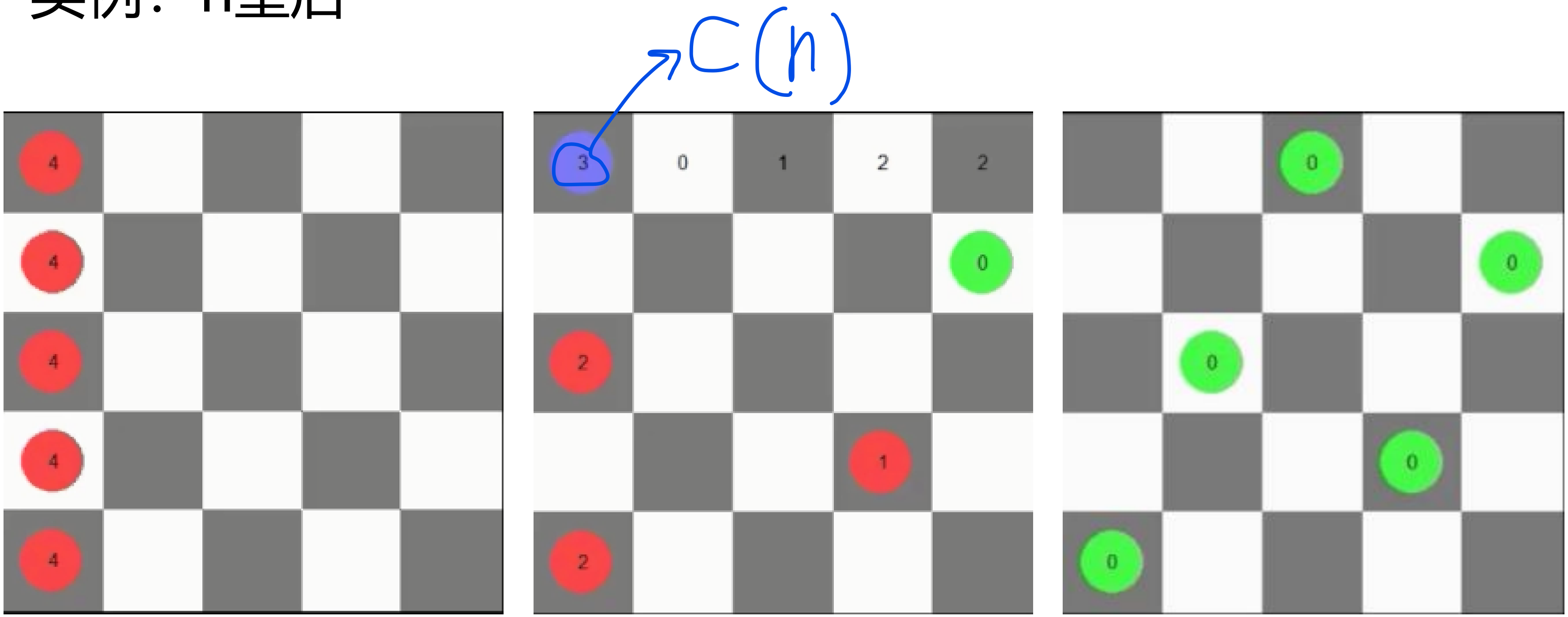
4. 局部搜索

实例：n皇后



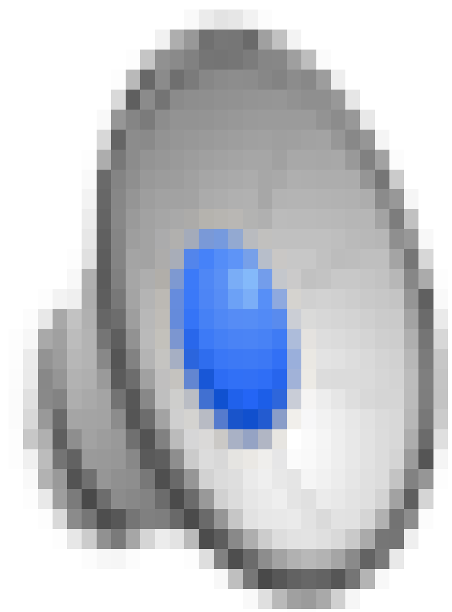
4. 局部搜索

实例：n皇后



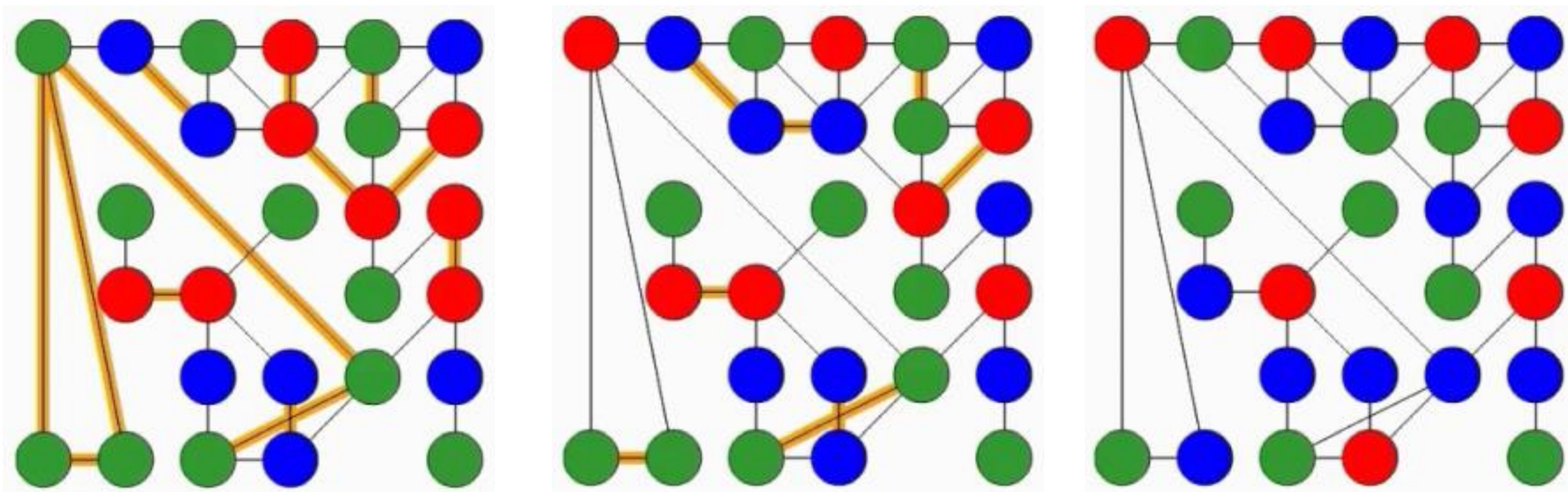
4. 局部搜索

实例：着色



4. 局部搜索

实例：着色



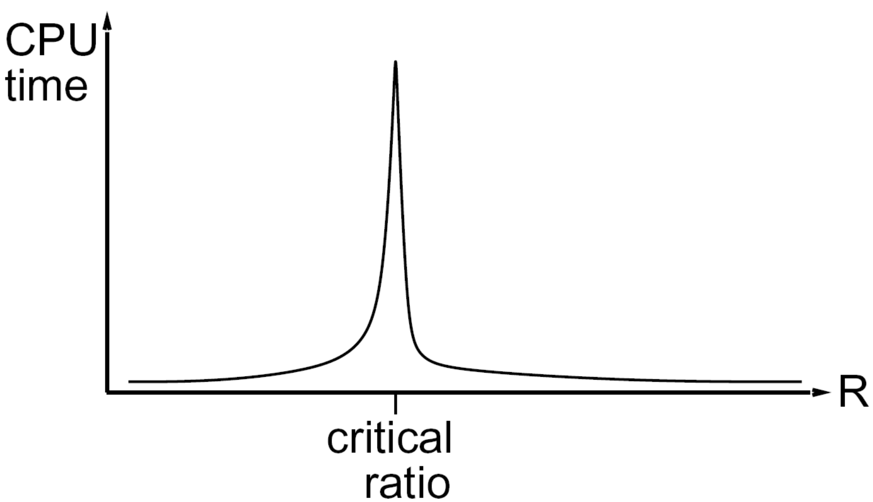
4. 局部搜索

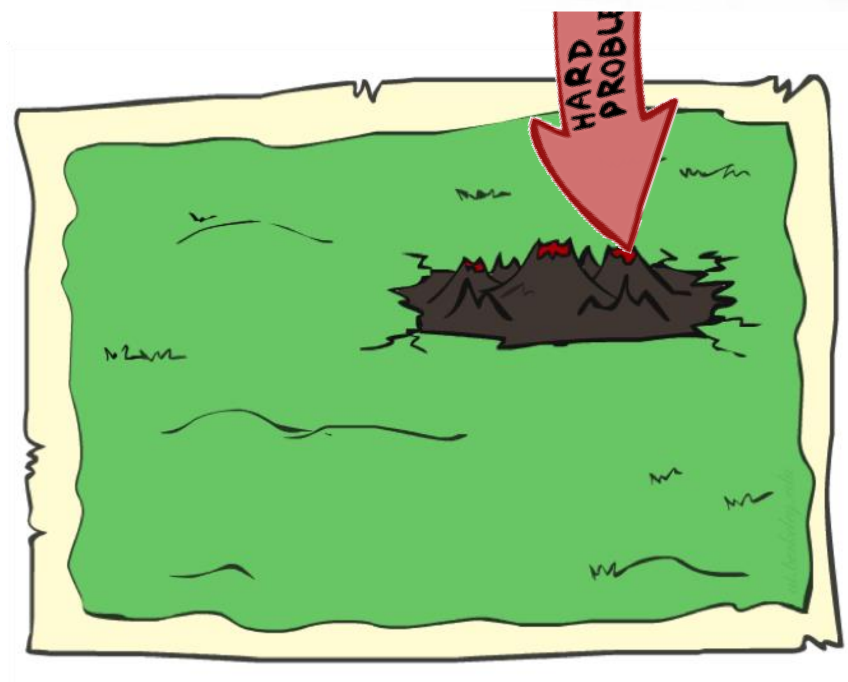
最小冲突性能

给定随机初始状态，可以在几乎恒定的时间内求解n个皇后（例如50步之内求解n = 1,000,000）！

对于任何随机生成的CSP而言，都可近似达到性能

$$R = \frac{\text{number of constraints}}{\text{number of variables}}$$





4. 局部搜索

爬山法(Hill Climbing)-思想简单且通用

- 可从任意地方开始
- 重复：移动到最好的近邻状态
- 如果无近邻状态比当前更好，则退出

简单的循环过程，不断向值增加的方向移动，即“登高”，在到达一个“峰顶”（邻接状态中没有比它更高的）的时候终止。

```
function Hill-Climbing( problem) returns a state that is a local maximum
  current ← Make-Node(problem. Initial-State)
  loop do
      neighbor ← a highest-valued successor of current
      if neighbor. Value ≤ current. Value then return current. State
      current ← neighbor
```

4. 局部搜索

爬山法(Hill Climbing)-思想简单且通用

- 可从任意地方开始
- 重复：移动到最好的近邻状态
- 如果无近邻状态比当前更好，则退出

简单的循环过程，不断向值增加的方向移动，即“登高”，在到达一个“峰顶”（邻接状态中没有比它更高的）的时候终止。

算法不会考虑与当前状态不相邻的状态，算法不维护搜索树，当前节点的数据结构只记录当前状态和目标函数值。

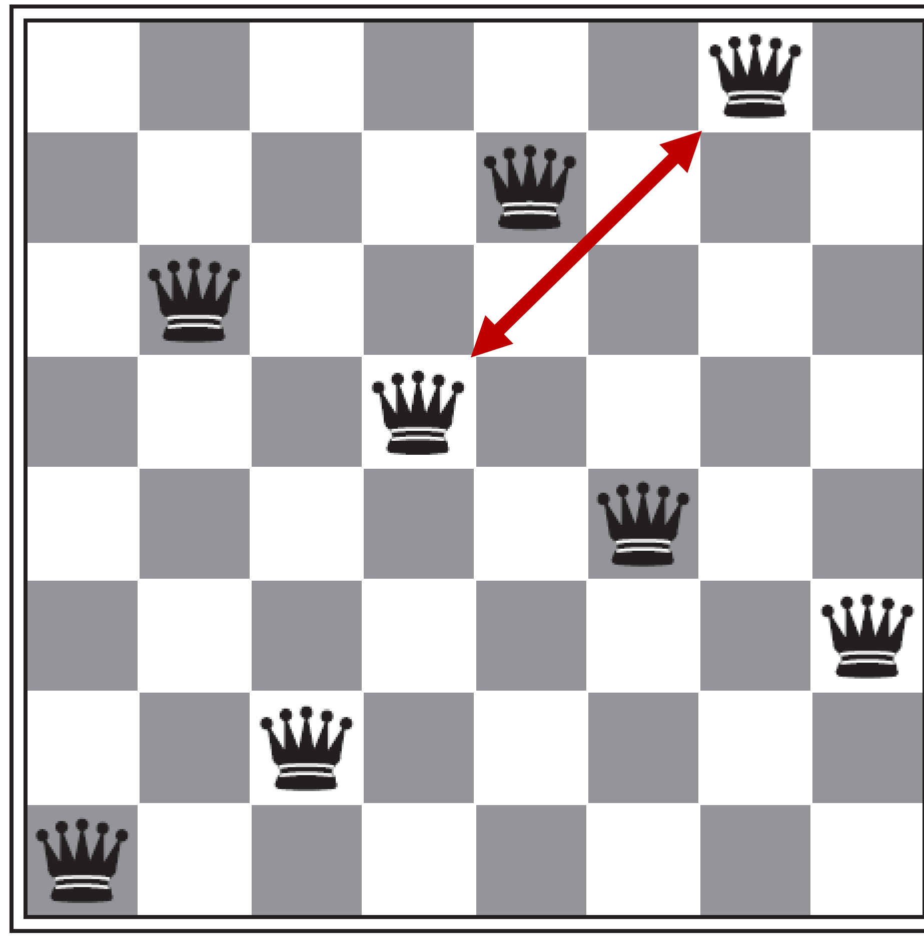
完备性?

最优性?

4. 局部搜索

3	4	2	3	2	2	1	0
18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	♛	13	16	13	16
♛	14	17	15	♛	14	16	16
17	♛	16	18	15	♛	15	♛
18	14	♛	15	15	14	♛	16
14	14	13	17	12	14	12	18

状态h=17及后继的h值（列位置）



局部最优h=1(后继大于1)

4. 局部搜索

爬山法(Hill Climbing)

爬山法有时被称为**贪婪局部搜索**，因为它只是选择邻居中状态最好的一个，而不考虑下一步怎么走。

在遇到以下情况时，爬山法都会到达无法再取得进展的地点

1）局部极大值；

2）山脊；

3）高原，

设置允许侧向移动的次数，可以提高爬山法的成功率，但是相应的平均步数会增加。

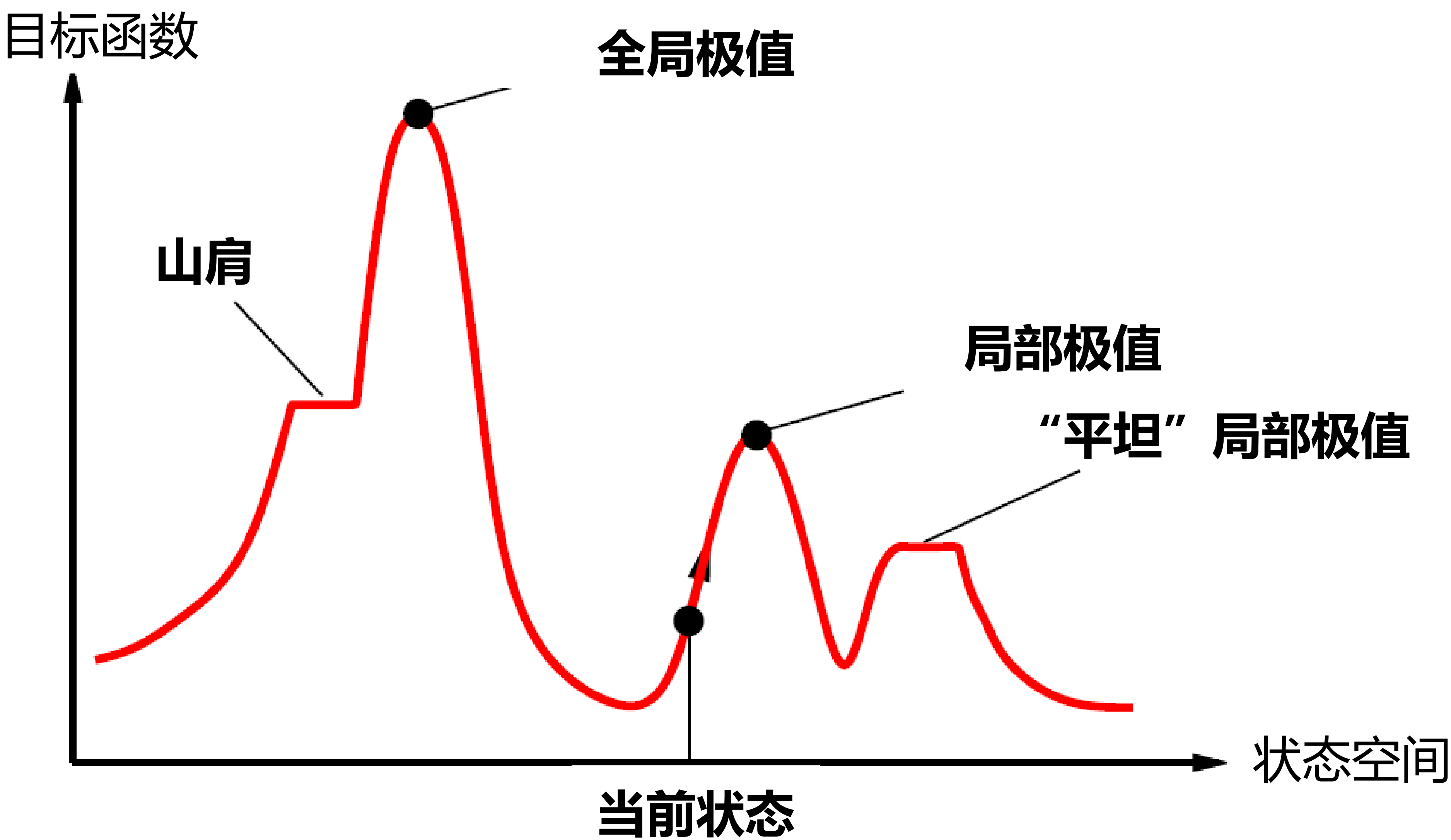
随机爬山法：概率随机选择下一步，可能找到更好的解。

首选爬山法：随机直至选到优于当前的结点，后继多时较好。

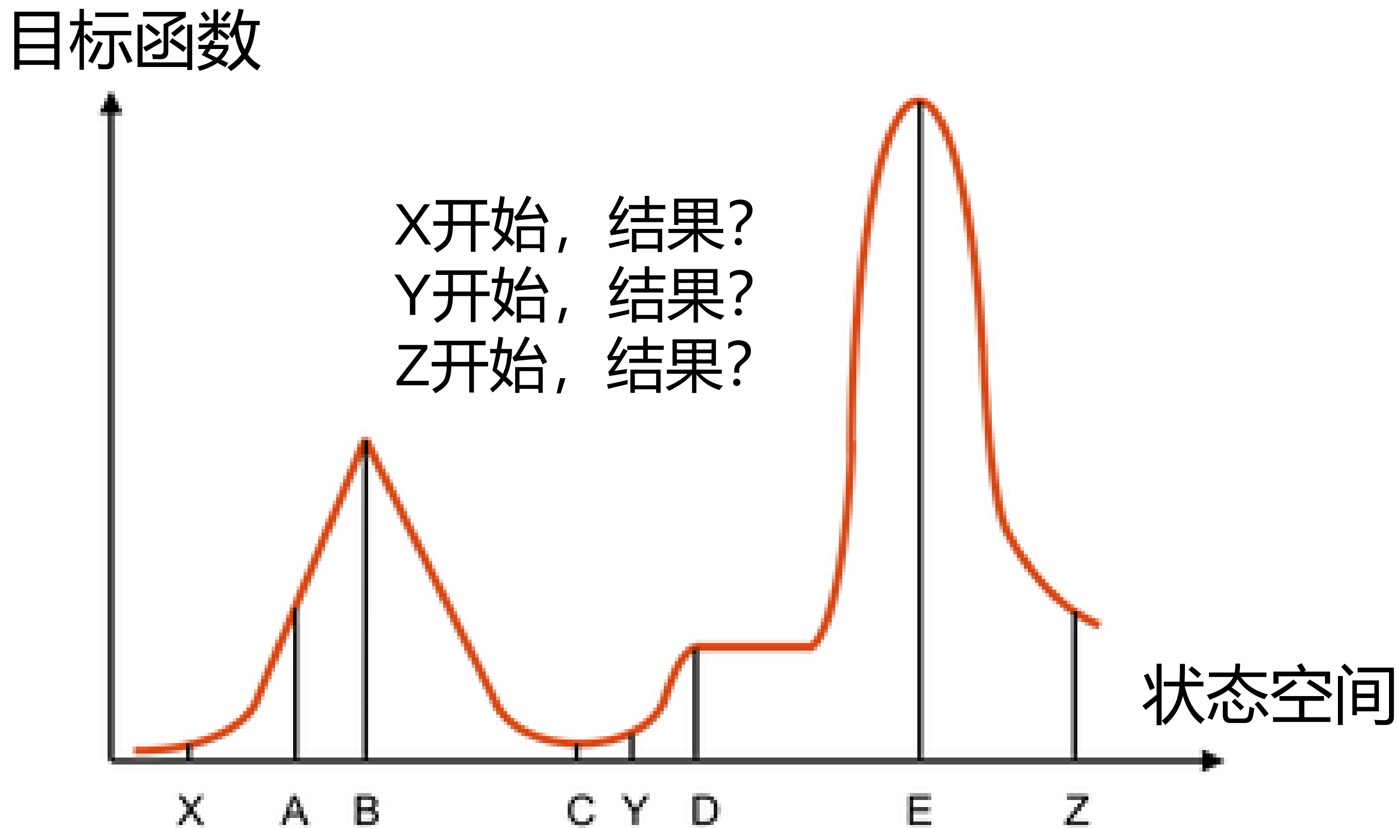
随机重启爬山法：多次尝试，算法完备性的概率接近1。

300万皇后，不超过1分钟

4. 局部搜索



4. 局部搜索



4. 局部搜索

模拟退火(Simulated Annealing)

基本思想：允许“下山”动作跳出局部极小值

- 爬山法不完备，因为会碰到局部极大值
- 随机走是完备的（因为完全等概率选择后续节点，不受局部极大值影响），但是效率极低。

模拟退火算法：把爬山法和随机行走以某种方式结合，同时得到效率和完备性。

退火：将固体加温至充分高，再让其徐徐冷却，加温时，固体内部粒子随温升变为无序状，内能增大，而徐徐冷却时粒子渐趋有序，在每个温度都达到平衡态，最后在常温时达到基态，内能减为最小。

https://baike.baidu.com/item/%E6%A8%A1%E6%8B%9F%E9%80%80%E7%81%AB%E7%AE%97%E6%B3%95/355508?fromtitle=%E6%A8%A1%E6%8B%9F%E9%80%80%E7%81%AB&fromid=8664695&fr=aladdin

4. 局部搜索

模拟退火(Simulated Annealing)

function SIMULATED-ANNEALING(*problem, schedule*) **returns** a solution state
  inputs: *problem*, a problem
    schedule, a mapping from time to "temperature"
  local variables: *current*, a node
    next, a node
    T, a "temperature" controlling prob. of downward steps

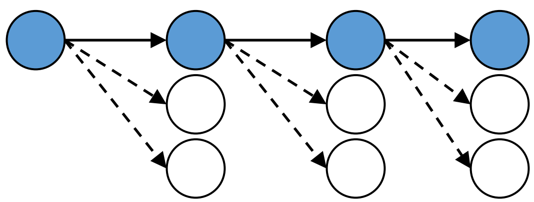
  current ← MAKE-NODE(INITIAL-STATE[*problem*])
  for $t \leftarrow 1$ **to** ∞ **do**
    $T \leftarrow schedule[t]$
    if $T = 0$ **then return** *current*
    next ← a randomly selected successor of *current*
    $\Delta E \leftarrow$ VALUE[*next*] – VALUE[*current*]
    if $\Delta E > 0$ **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{\Delta E/T}$

时间越长，概率越小

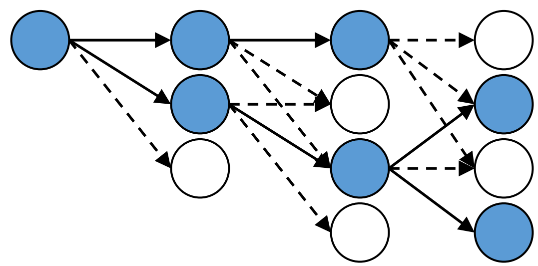
4. 局部搜索

局部束搜索(Local Beam Search)

从k个随机生成的状态开始，每一步全部k个状态的所有后继全被生成。如果其中有一个是目标状态，则算法停，否则从整个所有后继列表中选择k个最佳后继，重复直至达到目标状态。



贪婪搜索



束搜索

局部束搜索缺乏多样性，很快就会聚集到状态空间中的一小块区域内，使得搜索代价比爬山法版本还多。

随机束搜索（stochastic beam search）与随机爬山法类似，并不是选最好的k个，而是随机选择k个，其中选中概率是状态值的递增函数（正比）。随机束搜索类似于自然选择。

4. 局部搜索

遗传算法(Genetic algorithm)

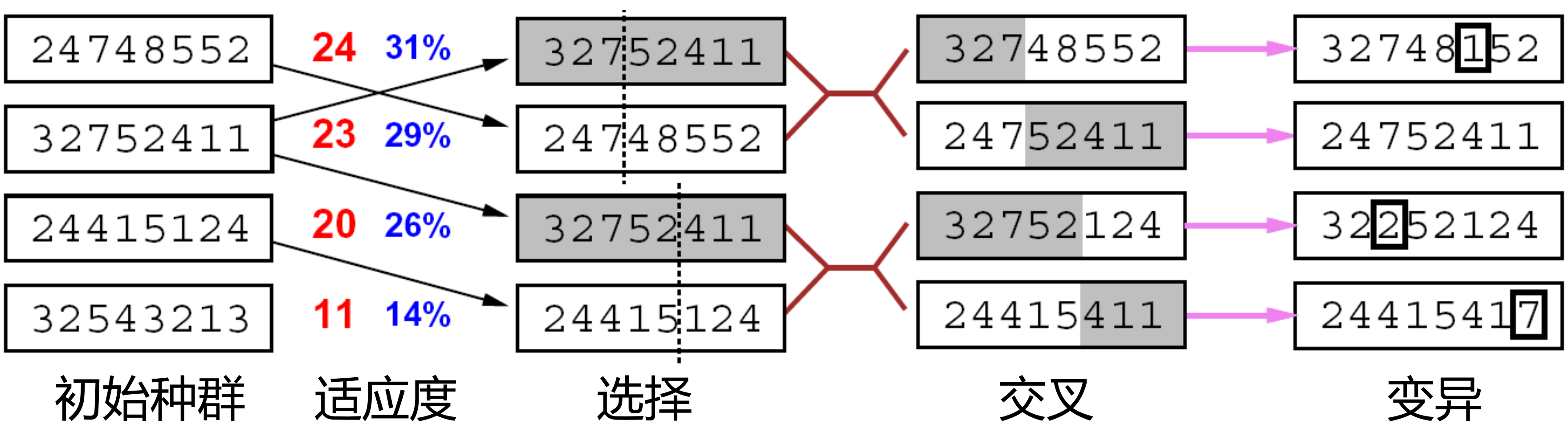
遗传是随机束搜索的一个变形。把两个父状态结合来生成后继，而不是通过修改单一状态进行

种群： k个随机初始状态

个体： 每个状态

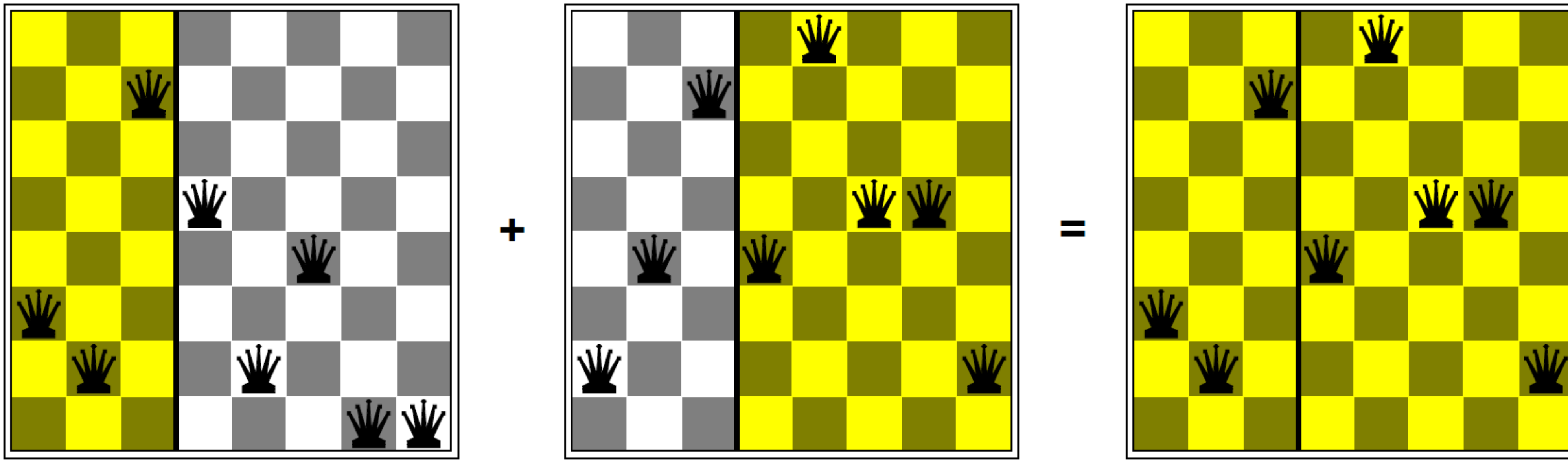
适应度函数： 评估每个状态的值（不相攻击的皇后对数）

操作： 选择、交叉、变异



4. 局部搜索

遗传算法(Genetic algorithm)



八皇后问题交叉操作

4. 练习题

回溯算法优化方法有哪些，其主要思想是什么，可以带来哪些好处?

4. 编程题

编程实现N皇后回溯搜索算法，至少采用一种优化策略加速搜索过程。

实现N皇后至少一种局部搜索方法，并分析其算法的性能（四个搜索算法评价指标）。

谢谢聆听 欢迎提问