通知

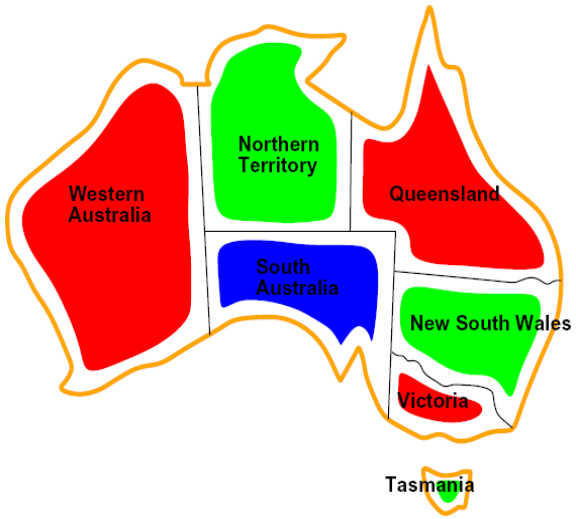
- 作业2 发布
  - 请于11月3日课堂上提交纸版。

今天的内容

- 约束满足问题及其求解

为什么研究约束满足问题（CSP）

- 约束满足问题
  - 变量（X），值域（D），约束（C）
  - 对所有变量的一组赋值，不违背任何约束条件
- 求解比状态空间搜索更快
  - 能够削减大量搜索空间
  - 例如，着色问题, if SA=蓝色,对于邻居的赋值，3^5=243 → 2^5=32, 减少87%
- 代表一系列的现实中问题
  - 配置问题；时间表计划问题；硬件配置；公交调度；工厂调度；电路规划；故障诊断



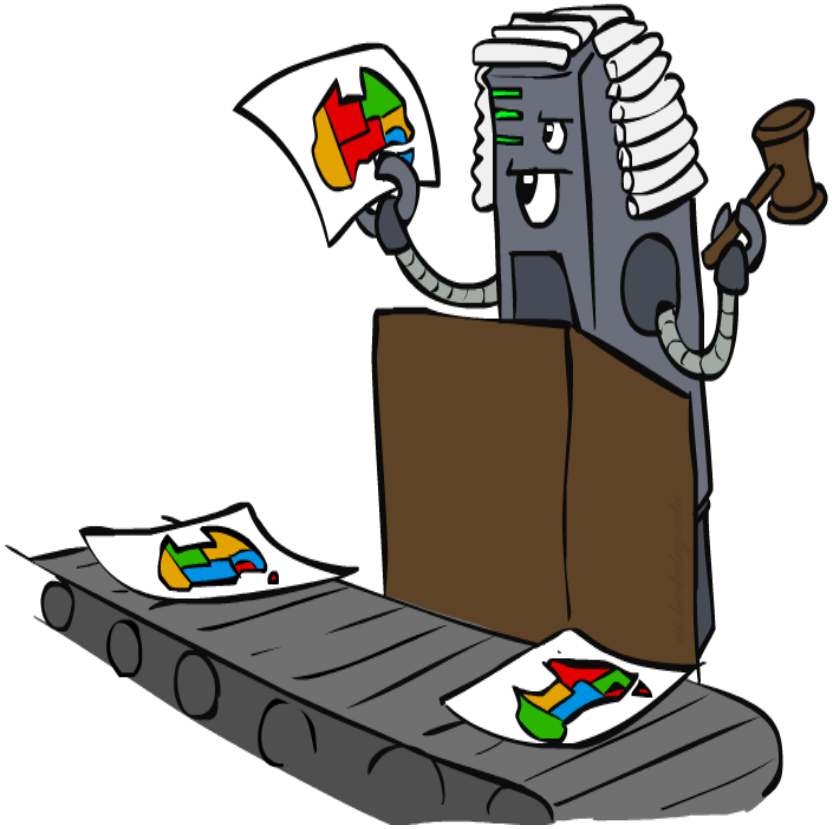
举例：课程安排问题

- 5门课程，3名教师，每名教师在一个时间只能教一门课
- 课程：
  - 课程1 – 计算机编程简介， 时间 8：00 – 9：00AM
  - 课程2 – 人工智能导论，时间 8：30 - 9：30AM
  - 课程3 – 软件工程， 时间 9：00 – 10：00 AM
  - 课程4 – 计算机视觉，时间 9：00 – 10：00 AM
  - 课程5 – 机器学习， 时间 10：30 – 11：30AM
- 教师：
  - 教师A，能够教课程1，2，5
  - 教师B，能够教课程3，4，5
  - 教师C，能够教课程1，3，4
- 描述为一个约束满足问题（一门课程是一个变量），指明变量的值域和约束；约束可以是隐式表达形式。

求解约束满足问题

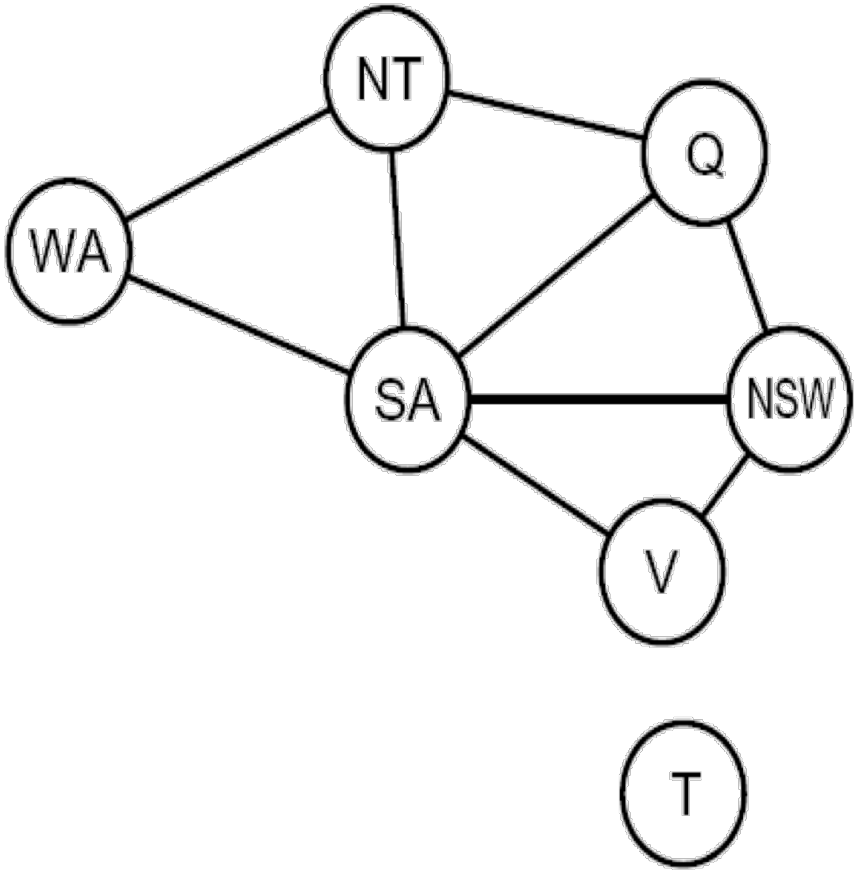
标准搜索的描述

- 状态反映了变量配值的当前情况（部分配值）
  - 初始状态: 没有配值, {}
  - 行动集合(s): 分配一个值给一个未赋值的变量
  - 结果状态(s,a): 该变量被赋了这个值
  - 目标-检测(s): 是否所有变量已被赋值 并且满足所有约束条件
- 我们开始将用最直接的方法，然后逐步改进



一般搜索方法

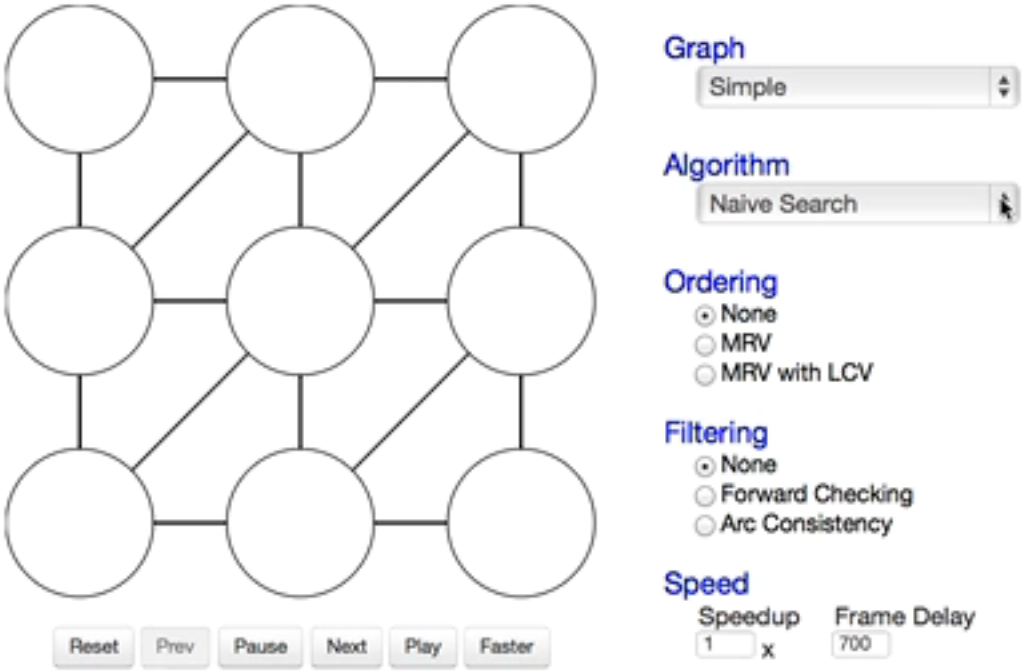
- 深度优先(DFS)会怎么样?



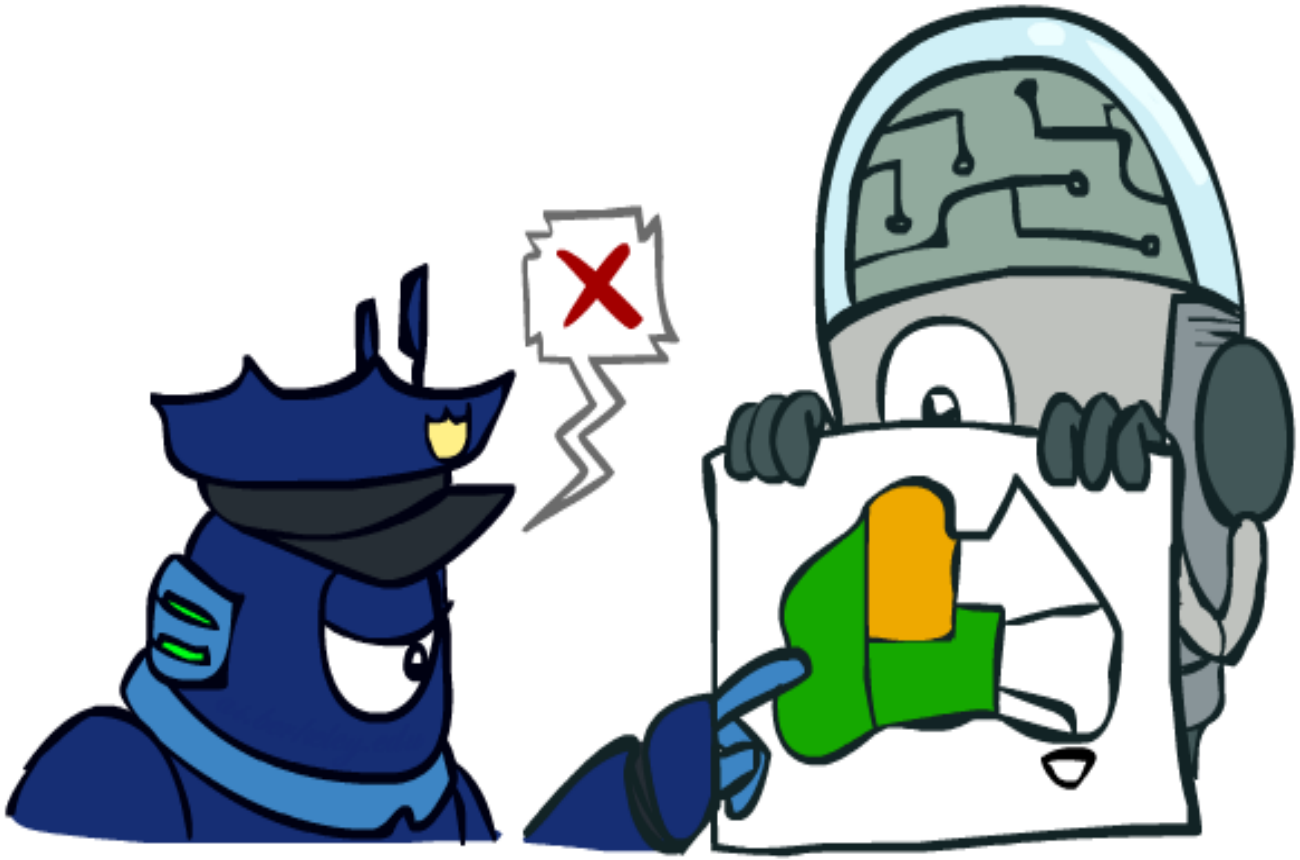
- 广度优先(BFS)会怎么样?

- 这些最直接的搜索方法在解这个问题时有什么问题?

Video of Demo Coloring -- DFS

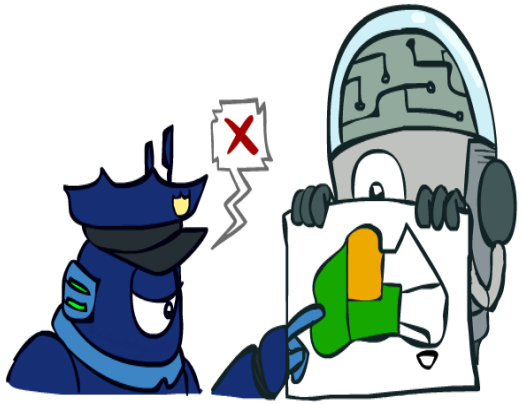


回溯搜索（Backtracking search）

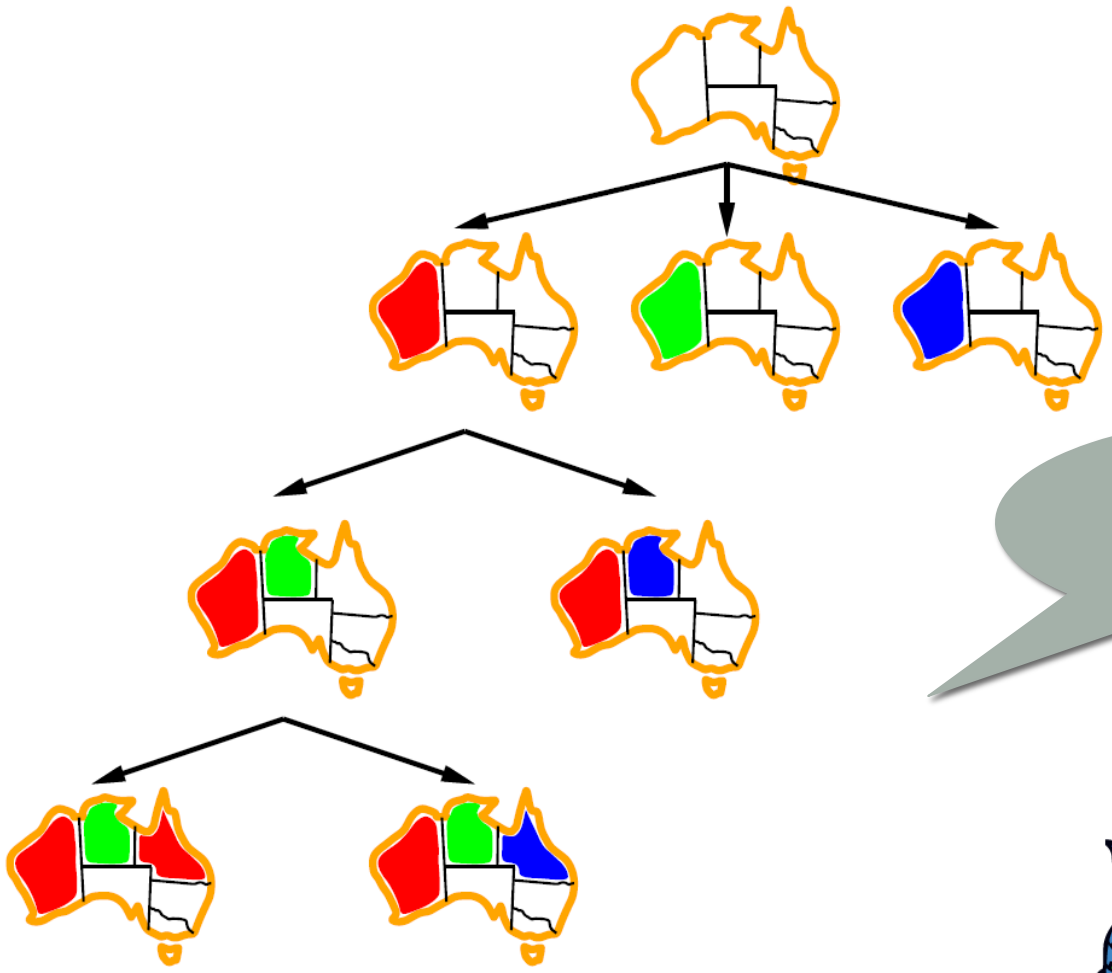


回溯搜索

- 回溯搜索是基本的无启发式信息的算法，用来求解CSP问题
- 想法 1: 一次探索一个变量
  - 在每一步只需考虑给一个变量配值
- 想法 2: 一边探索一边检查约束条件
  - 探索过程中检查当前的变量配值是否满足约束条件
  - 也许需要花费一些计算来检查约束条件是否满足
  - “逐步增加的目标测试”
- 深度优先搜索结合这两点改进，就叫作 **回溯搜索**
- 能够解决 n-皇后问题，直至 n = 25

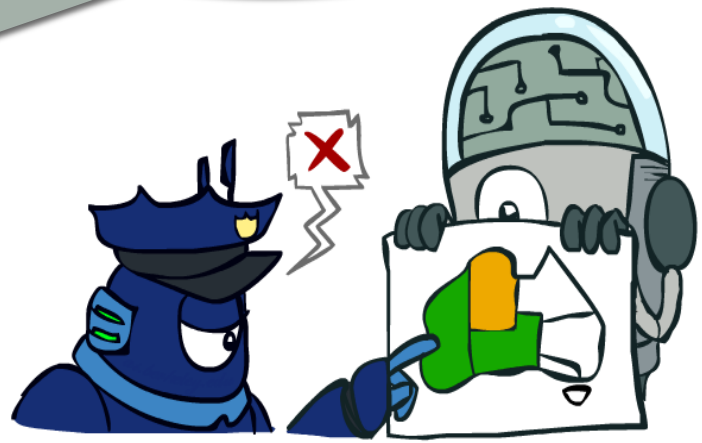


回溯搜索举例



一次一个变量

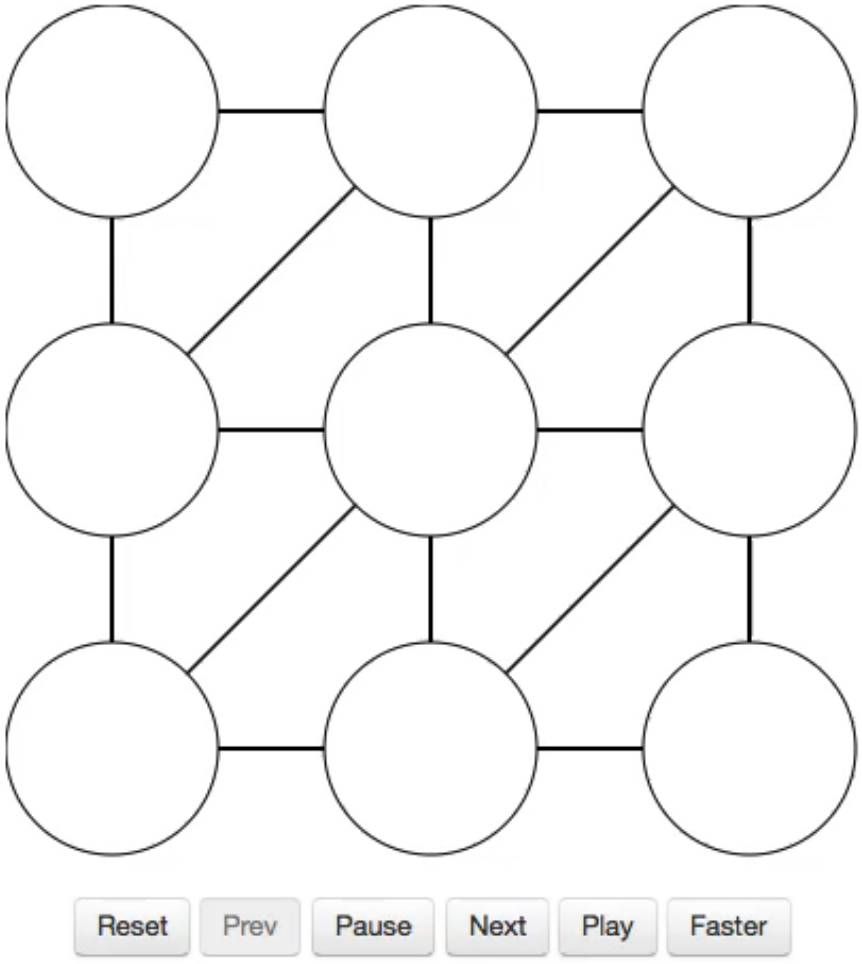
满足约束的值



回溯搜索算法

```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
    return BACKTRACK({ }, csp)
function BACKTRACK(赋值,csp) returns a solution, or failure
    if 赋值 是完全的 then return 赋值
    var ← 选择-未赋值的-变量(csp,赋值)
    for each value in 排序-值域中的值(var,赋值,csp) do
        if value 是一致的 with 赋值 then
            添加 {var = value} to 赋值
            推断结果 ←推断(csp,var,赋值)
            if 推断结果 ≠ failure then
                添加 推断结果 to 赋值
                result ← BACKTRACK(assignment, csp)
                if result ≠ failure then return result
        移除 {var = value} and 推断结果 从 赋值
    return failure
```

回溯搜索演示—着色问题


Reset
Prev
Pause
Next
Play
Faster
Graph
Simple
Algorithm
Naive Search
Ordering
None
MRV
MRV with LCV
Filtering
None
Forward Checking
Arc Consistency
Speed
Speedup
1
x
Frame Delay
700

CSP中的推断– 约束的传播

- 普通的状态空间搜索算法，只做一件事：搜索
- CSP有另外的选择
  - 搜索（选择一个新变量的配值）
  - 推断（也叫约束传播，为了变量值域保持一致性）
- 节点一致性
  - 一个变量，其值域里所有值满足该变量的一元约束。
- 弧一致性
  - X_i 对于X_j **的弧一致**的（X_i 与X_j 有二元约束关系）：如果D_i 中的每一个值在D_j 中能找到一个值使得满足它们间的二元约束。
  - X_i 是弧一致的，其所有相关的二元约束
  - 网络的弧一致性，每个变量都是弧一致的。

弧一致性算法

Function AC-3(csp) **return** false 如果不一致性存在，否则返回 true.

输入：csp,只包含二元约束关系

局部变量：弧队列，初始包含csp中所有的弧

while 弧队列 非空 **do**

(Xi, Xj) <- REMOVE-FIRST(弧队列)

if REVISE(csp, Xi, Xj) **then**

if 值域 Di = 0 **then return** false

for each Xk in Xi.邻居变量集 – {Xj} **do**

add (Xk, Xi) to 弧队列

return true

Function REVISE(csp, Xi, Xj) **return true** iff 当且仅当Xi的值域被改变

revised <- false

for each x in Di **do**

if Dj 中不存在值 y 使得（x, y）满足Xi 和 Xj 间的约束 **then**

从 Di 中 删掉 x 值

revised <- true

return revised

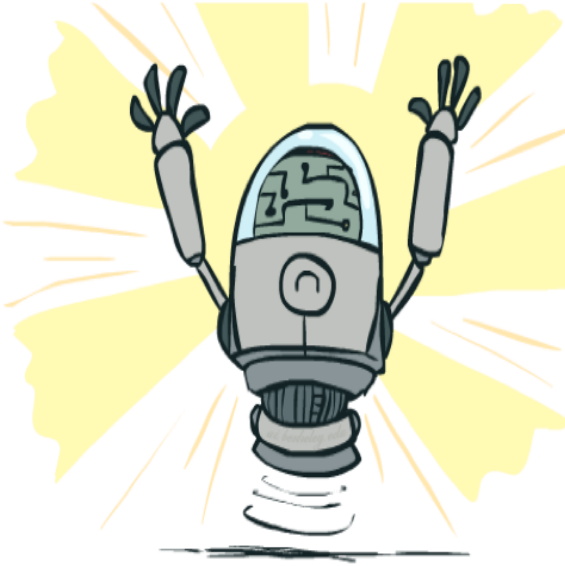
回溯搜索的改进

- 改进思想能极大提升搜索速度，并且适用于多方面的问题

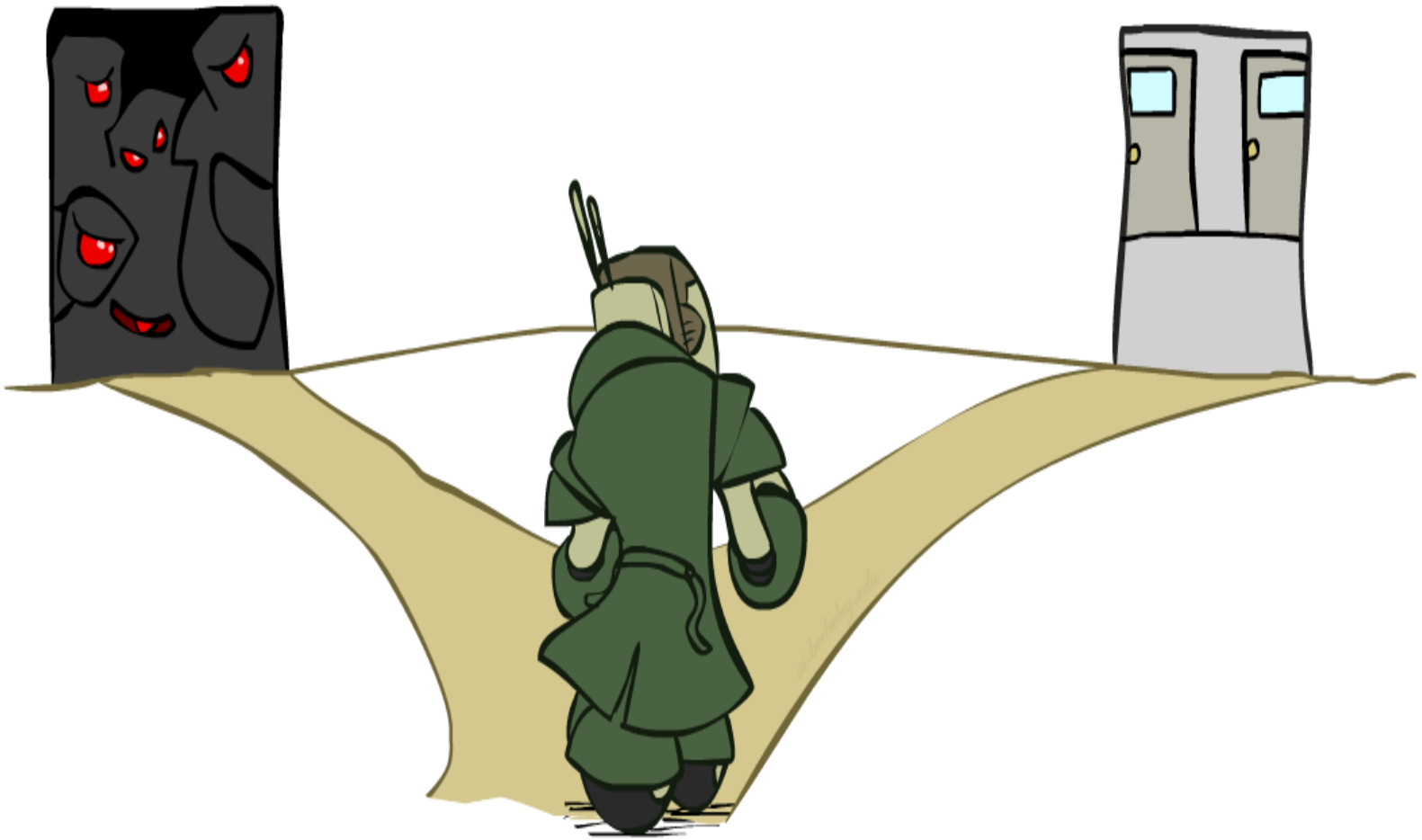
- 排序:
  - 下一轮挑选哪个变量进行配值?
  - 挑选值时，有什么顺序上的考虑?

- 过滤: 我们能否提前预测不可避免的失败?

- 结构: 我们能否利用问题的结构?

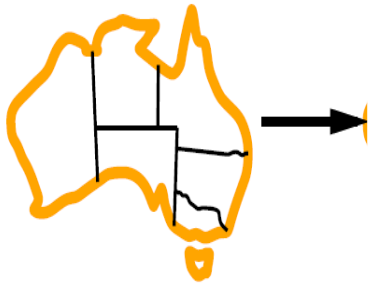


排序选择

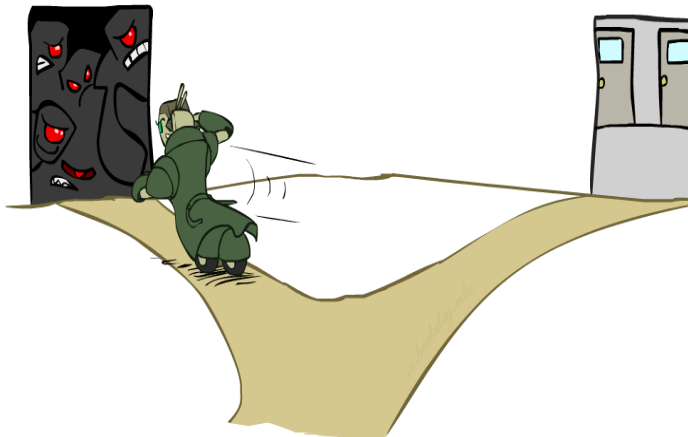


变量排序（选择）

- var ← 选择-未赋值的-变量(csp,assignment)
- 变量排序: 最小剩余值 (MRV):
  - 先选择其值域中所剩合理可选的值最少的变量

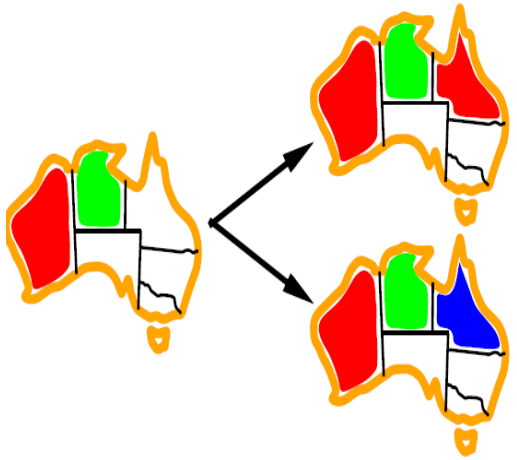


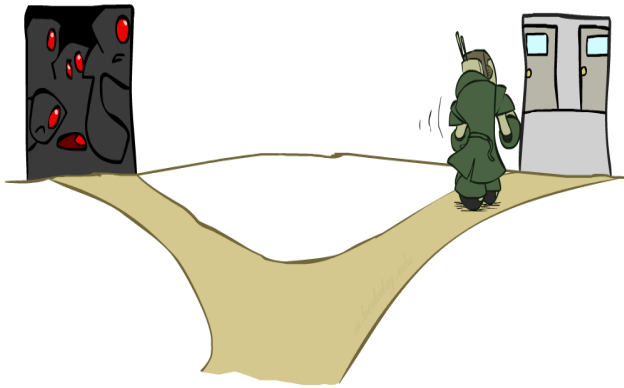
- 为什么是最少而不是最大?
  - 削减搜索状态空间
  - “快速失败” 排序
- 使用连接度数启发信息 打破一样的情况
  - 选择和其他变量连接数最多的变量



对值的排序选择

- **for each** value **in** 排序-值域里的值 (var,assignment,csp) **do**
- 选择最小制约的值 (LCV)
  - 选择对剩下的变量在选值的时候影响最小的值
  - 这可能需要花费些计算时间!
- 为什么最小而不是最大制约的?
  - 只需找到一个解，而不是所有的解
  - 最有可能形成一个解的
- 结合这些排序上的改进，能够解决1000-皇后问题

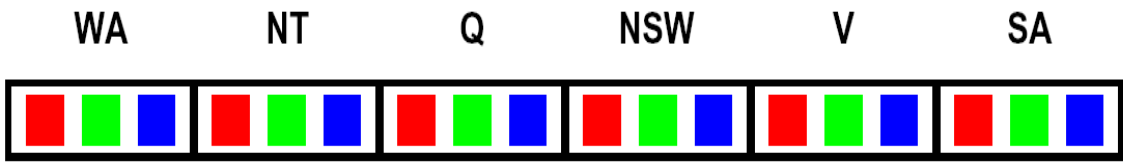




过滤

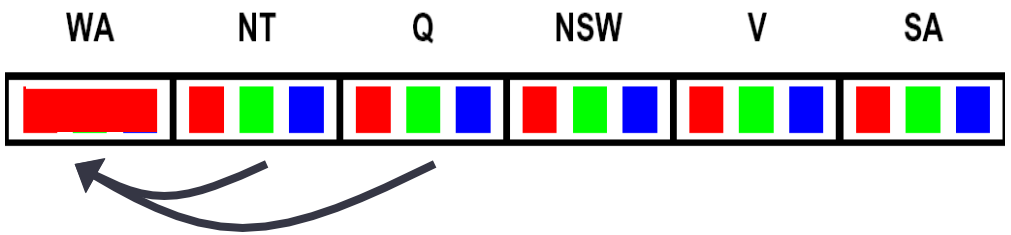
过滤: 前向检查法

- 过滤：搜索中持续观测未赋值的变量的值域，去掉违反约束条件的选项
- 简单的过滤: 向前检查法
  - 当添加对一个变量的赋值后，划掉剩下变量值域中违反约束条件的值



有用的概念: 弧的一致性（连贯性; Consistency）

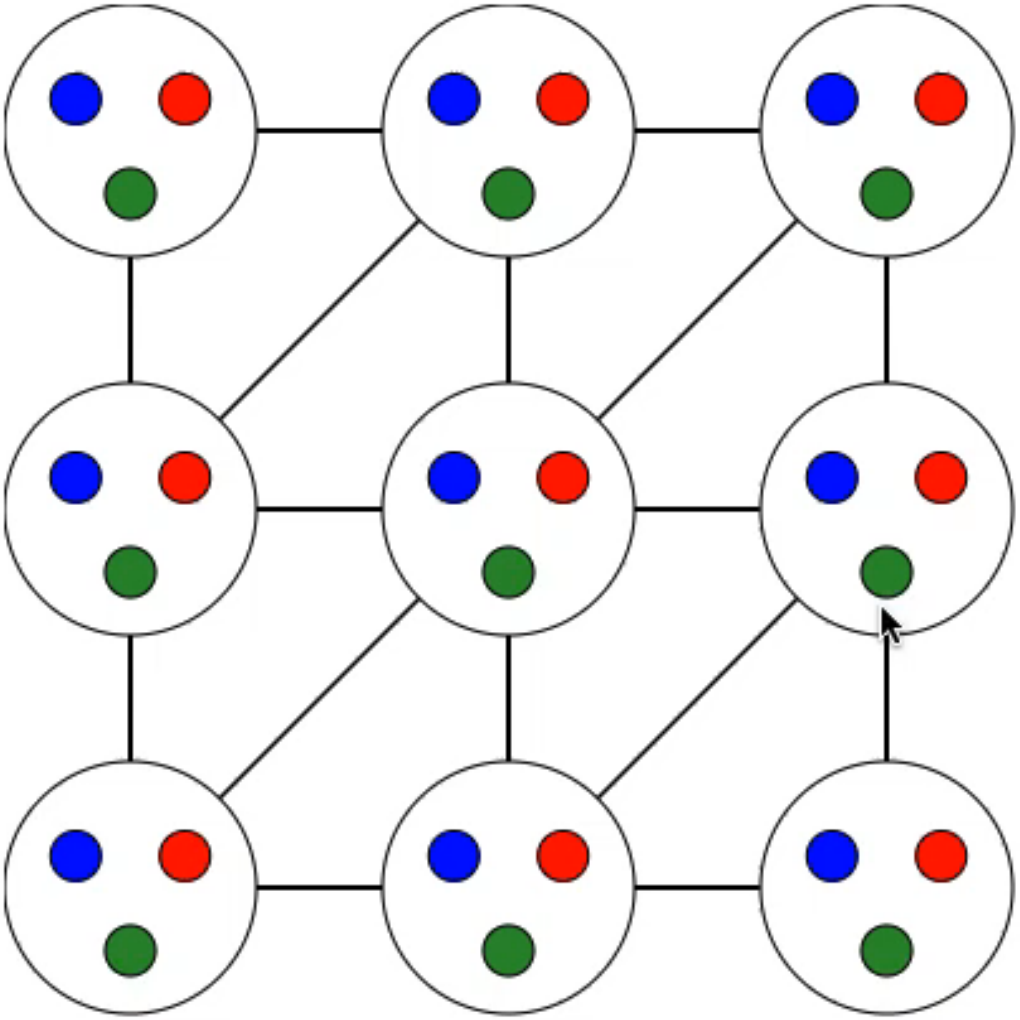
- 一个弧 X → Y 是 **一致的** 当且仅当 对于X中的 每一个 x ， Y中存在 某个 y 值 不违背任何一个约束条件



从弧的尾部删除

- 前向检查: 强制检查剩余未赋值（邻近的）变量指向新赋值变量的弧的一致性

演示– 回溯搜索结合前向检查法



Graph

Simple

Algorithm

Backtracking

Ordering

- None
- MRV
- MRV with LCV

Filtering

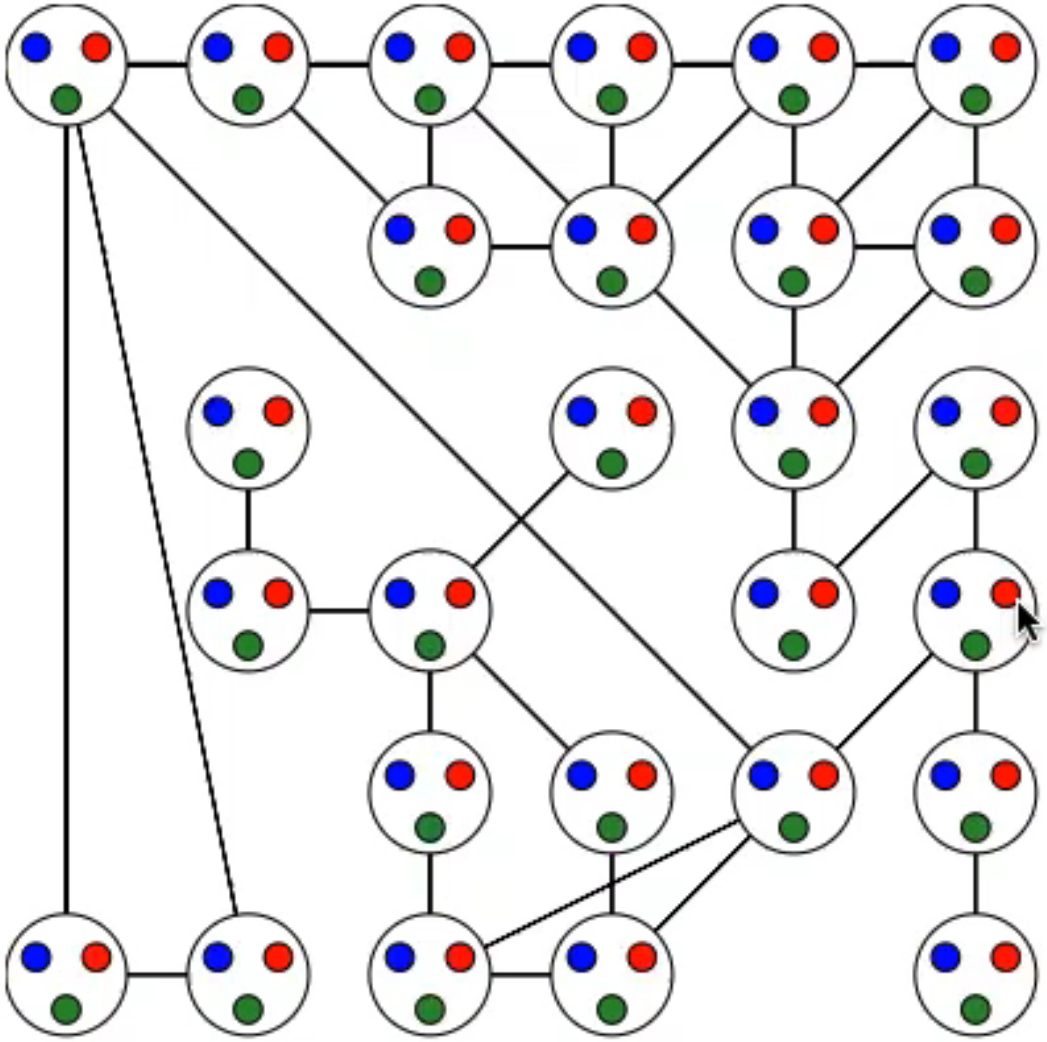
- None
- Forward Checking
- Arc Consistency

Speed

Speedup 1 x

Frame Delay 700

复杂的图



Reset | Prev | Pause | Next | Play | Faster

Graph

Complex

Algorithm

Backtracking

Ordering

- (•) None
- ( ) MRV
- ( ) MRV with LCV

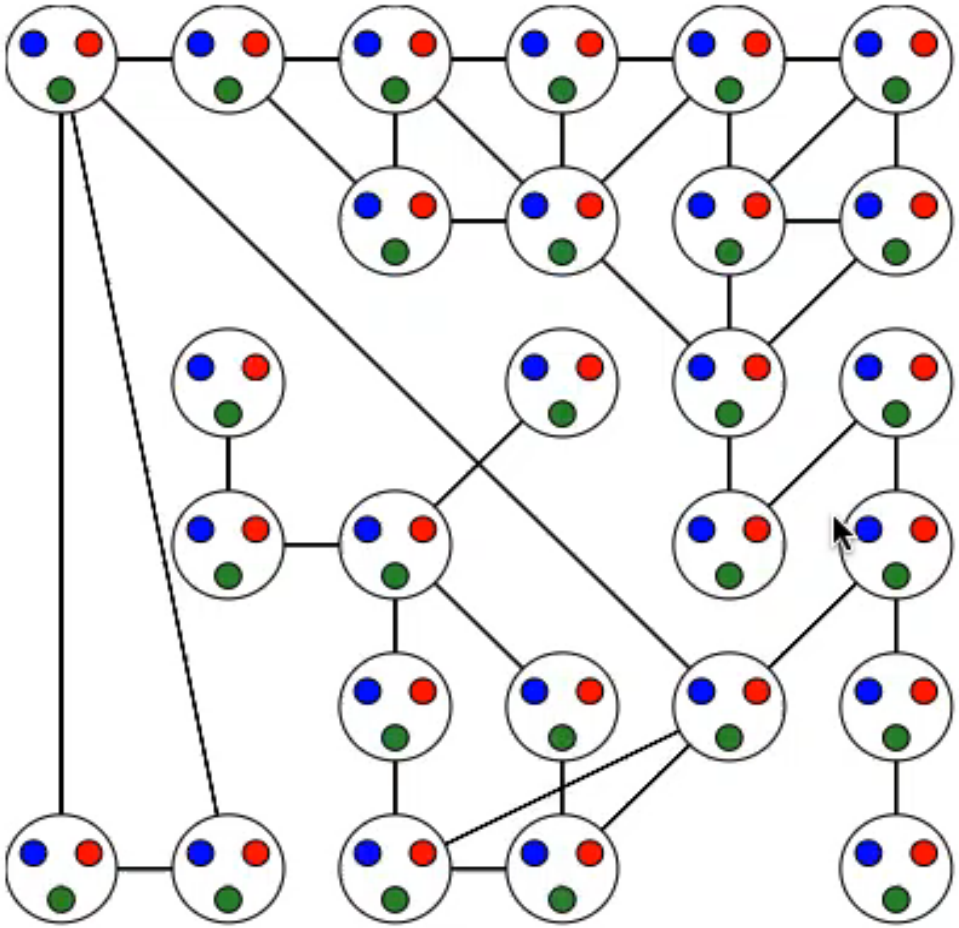
Filtering

- ( ) None
- (•) Forward Checking
- ( ) Arc Consistency

Speed

Speedup	Frame Delay
1 x	700

演示：着色问题 – 回溯 + 前向检查 + 排序



Reset | Prev | Pause | Next | Play | Faster

Graph

Complex

Algorithm

Backtracking

Ordering

- None
- MRV
- MRV with LCV

Filtering

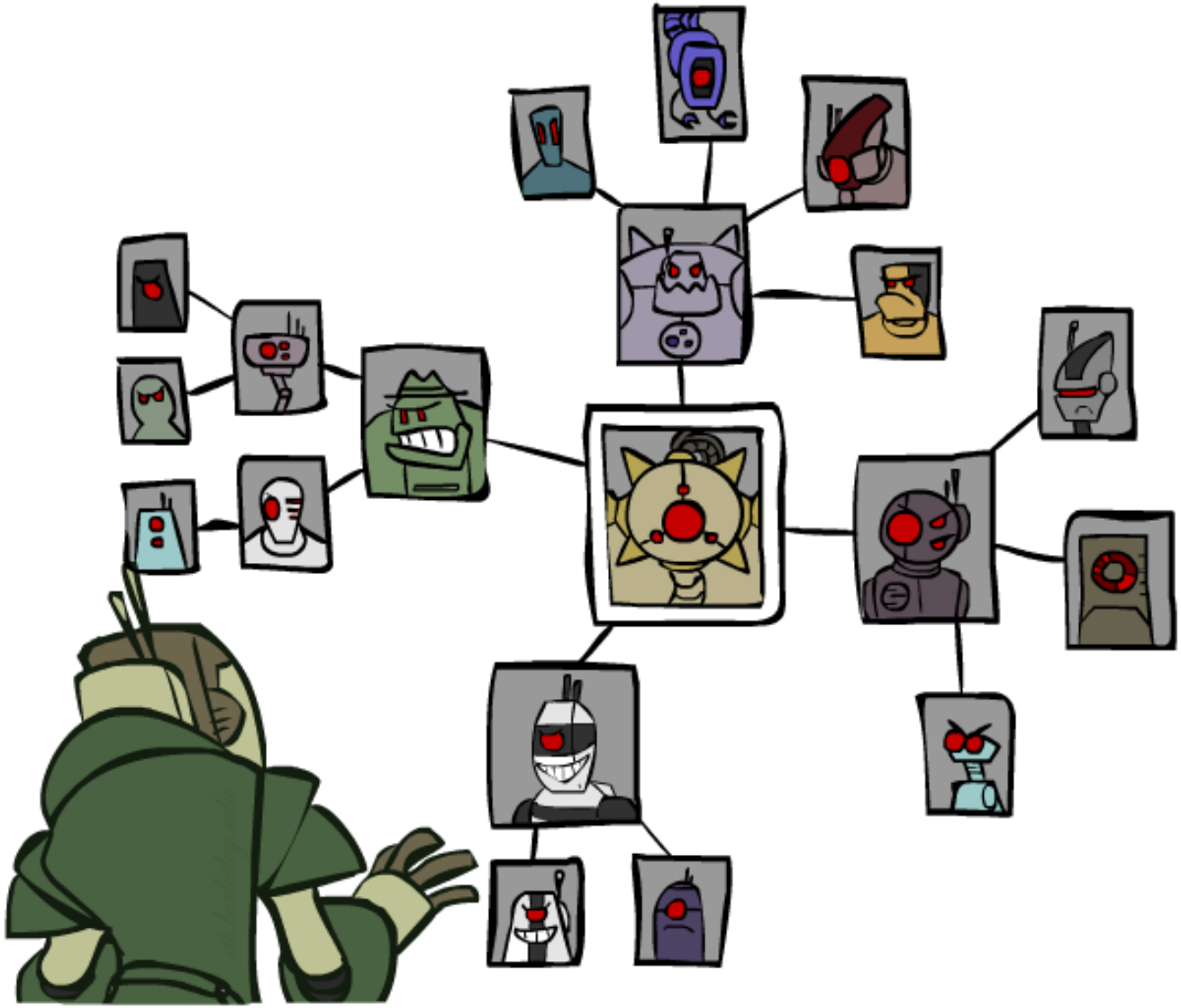
- None
- Forward Checking
- Arc Consistency

Speed

Speedup: 1 x

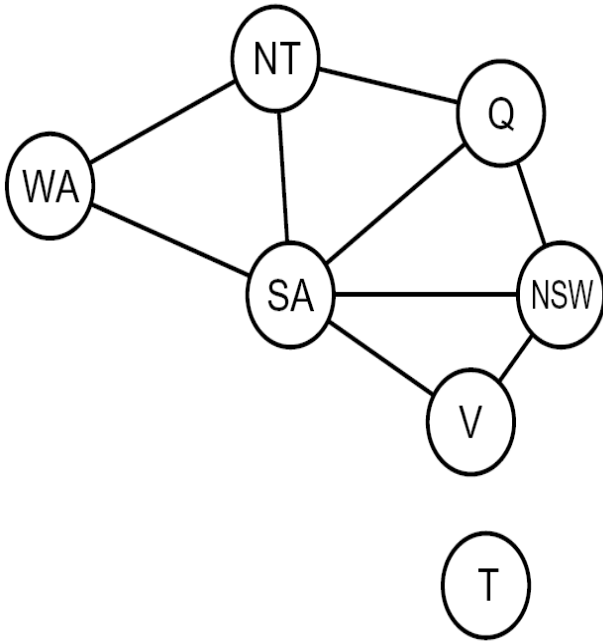
Frame Delay: 700

利用问题的结构



问题结构

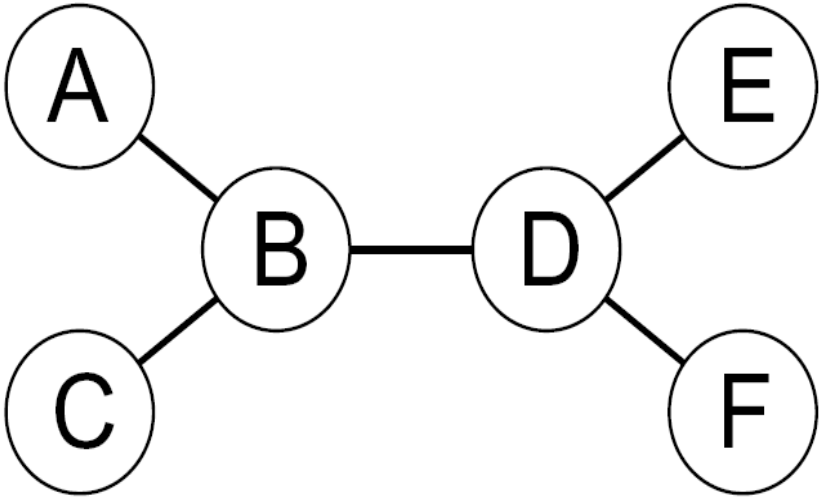
- 极端情况: 独立的子问题
  - 例如: Tasmania 和大陆不相连



- 独立的子问题可以被认为是约束图中连接的组件:
  - ***分治法!***

- 假设一个 n 变量的图可以被分成 几个子问题, 每个子问题有 c 个变量:
  - 最差情况下的求解成本是 $O((n/c)(d^c))$, ***n*的线性关系**
  - 比如, n = 80, d = 2, c =20, 搜索 1千万 节点/秒
  - 原始问题: 2^{80} = **40亿年（宇宙的年龄）**
  - 4个子问题: 4 x 2^{20} = **0.4 秒**

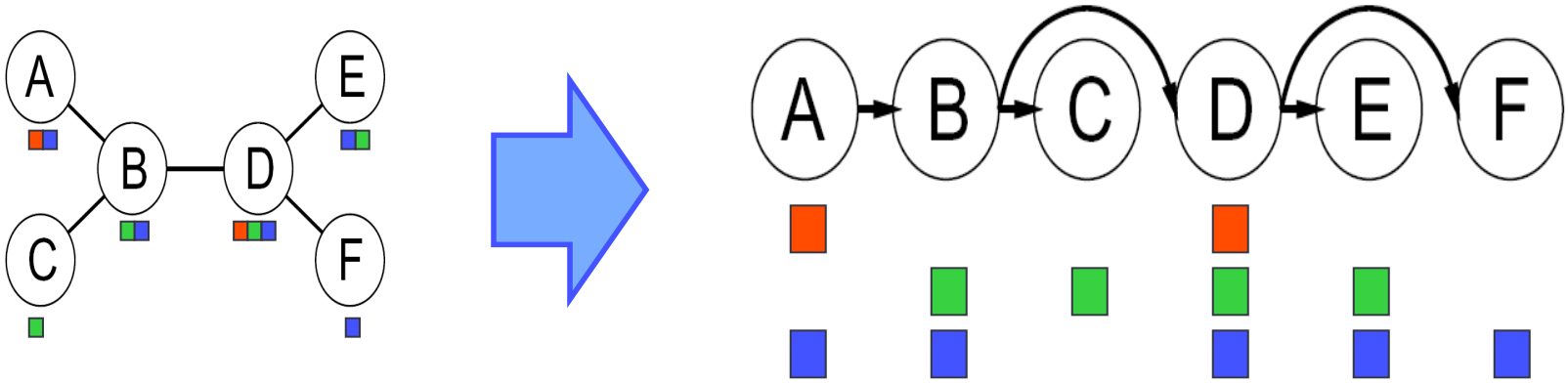
树结构的 CSPs



- 定理: 如果约束图无环，则其对应的约束满足问题的求解时间复杂度是 $O(n\ d^2)$
  - 比较一般的 CSPs, 最差时间复杂度是 $O(d^n)$
  - 不需要回溯

树结构的 CSPs的求解算法

- 树结构的CSPs的求解算法：
  - 排序: 随便选一个根节点（弧方向既定），把变量线性（拓扑）排序，使得父节点排在子节点之前

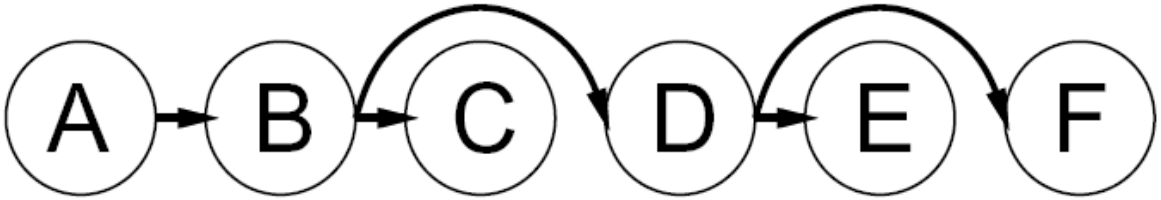


  - 从后向前删除（保持弧一致）：For i = n : 2, 应用弧一致性检查，删除不一致的值(父节点(X_i),X_i)
  - 从前向后赋值: For i = 1 : n, 赋值 X_i 和父节点 Parent(X_i) 相一致的值

- 运行时间: $O(n\ d^2)$

树结构的 CSPs

- 声明 1: **在从后向前**的一致性检查后, 所有根到叶的弧都是一致性的
- 证明: 每个 X->Y 如果是一致性的，那么 Y的值域在此后不会被减小 (因为 Y 的 子节点 在Y之前先被处理过)

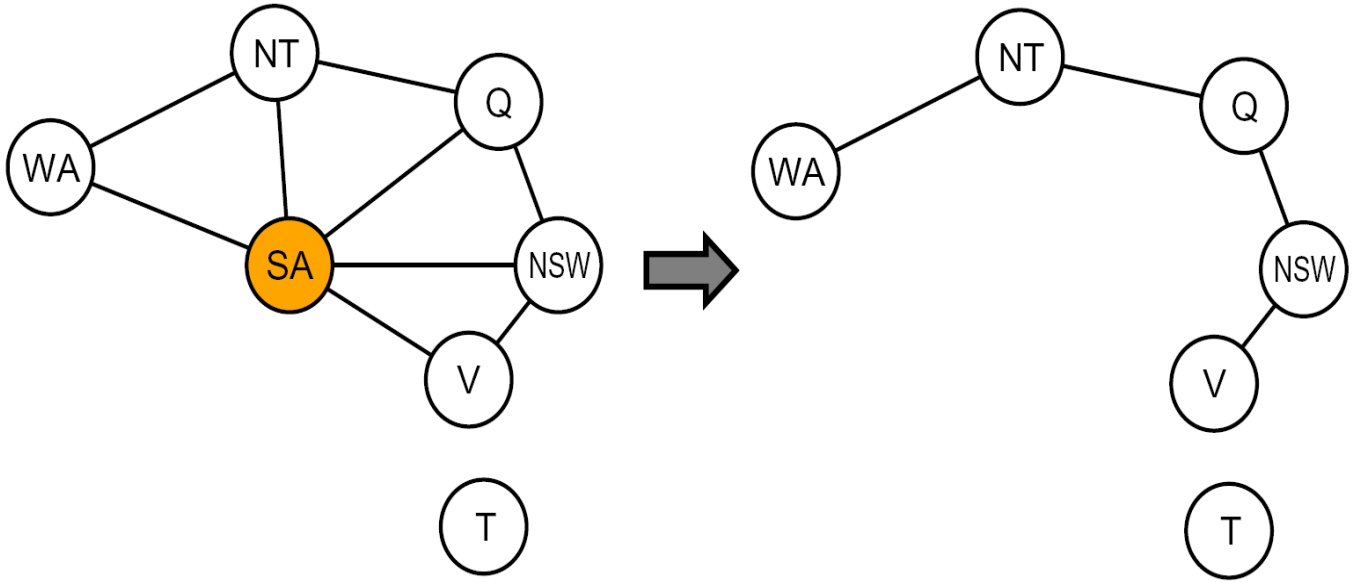


- 声明 2: **如果从根到叶的弧都是一致性的**, 那么从前向后的赋值过程将不会有回溯
  - 证明: 归纳法

- 这个算法不适用于约束图中有环的情况。

改进结构（利用树结构问题的求解优势）

几乎是树结构的 CSPs



- **条件制约**: 赋值一个变量, 剪裁这个变量的邻居变量的值域
- **切集条件制约**: 对切集变量赋值 (所有可能的组合)，使得 剩下的约束图变成一个树
- 切集大小是c，时间复杂度为 $O((d^c) (n-c) d^2)$，c 很小时算法很快
- 例如, 80 个变量, c=10, 40亿 years -> 0.029 秒

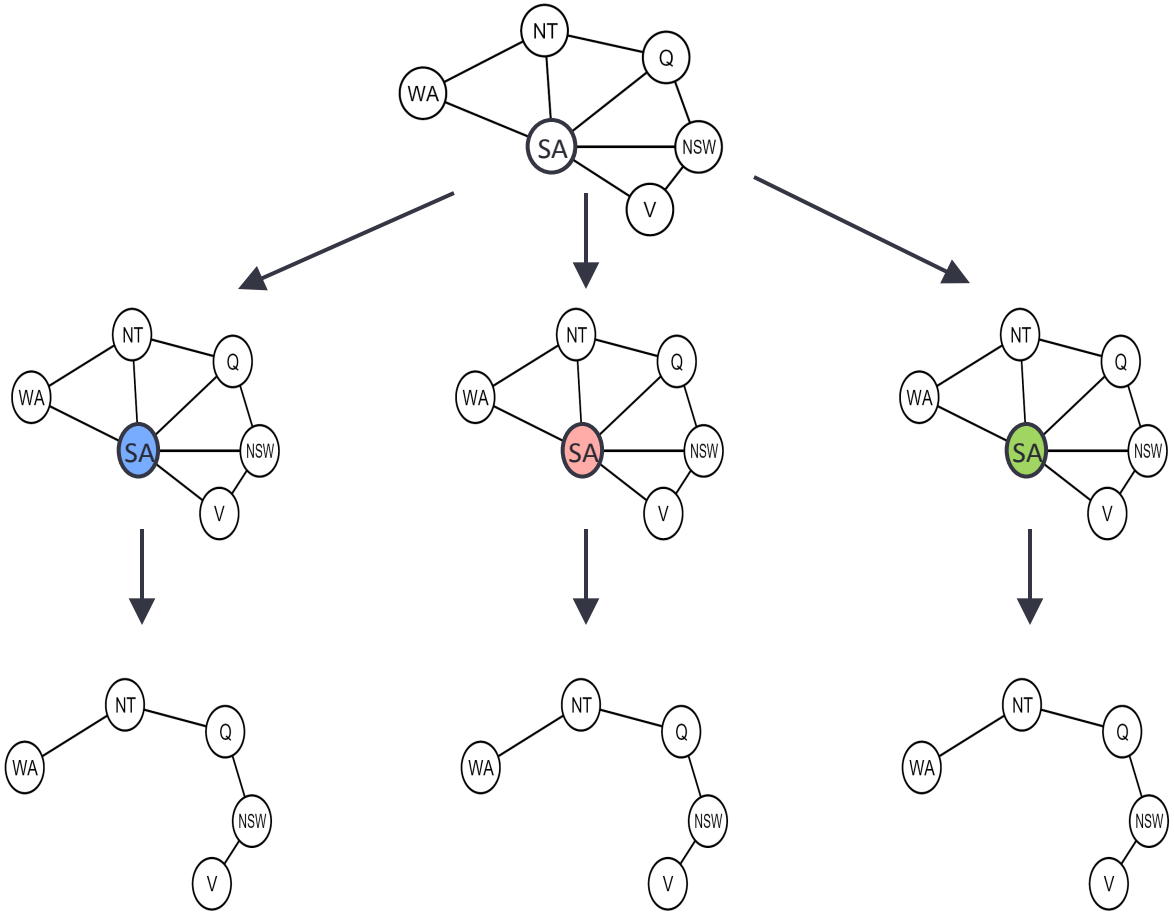
切集条件化算法

选择一个切集

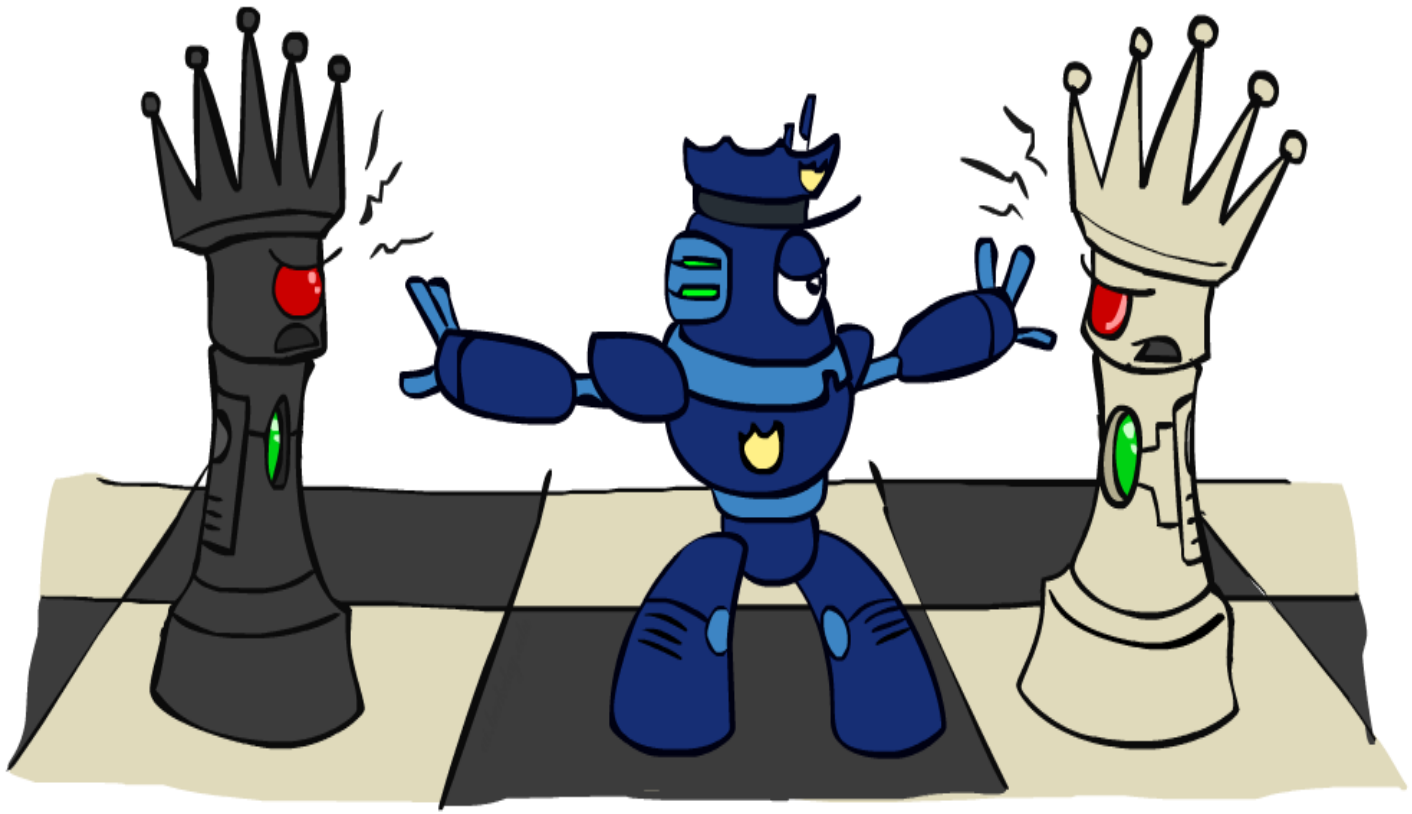
对切集变量赋值
(所有可能组合)

相对于每一组切集赋值，计算剩余变量相一致的值域

求解剩余的 CSPs
(树结构的)



局部方法求解 CSPs



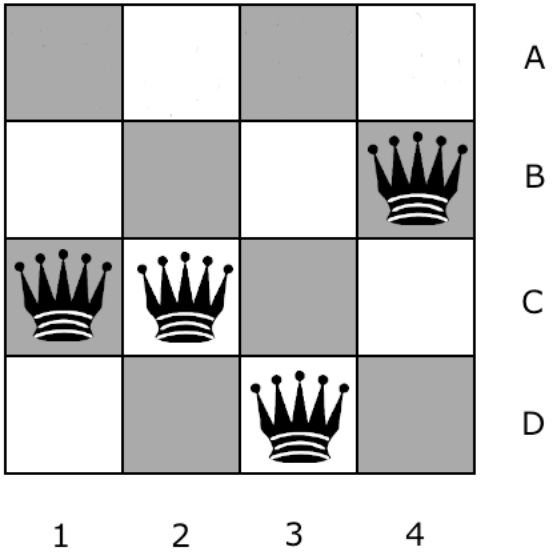
最小冲突算法

- 像爬山算法, 但不完全是!

- 算法: 开始时所有状态都已随机赋值，迭代改进，直至问题得到求解,
  - 变量选择: **随机选择** 任何冲突的变量
  - 值选择: 最小冲突启发信息:
    - 选择一个相对于当前的赋值情况，与约束条件相冲突最少的值
    - 相当于爬山算法中h(v)=冲突的约束条件总数
    - 随机打破平局情况

举例：4-皇后问题

- 假设
  - 随机总是选择最左边有冲突的皇后
  - 值平局时，选最上面的方格值
- 利用最小冲突局部算法
- 第一步
  - 变量？值？
  - 1 , A
- 第二步
  - 变量？值？
  - 2, A
- 第三步
  - 变量？值？
  - 1, C

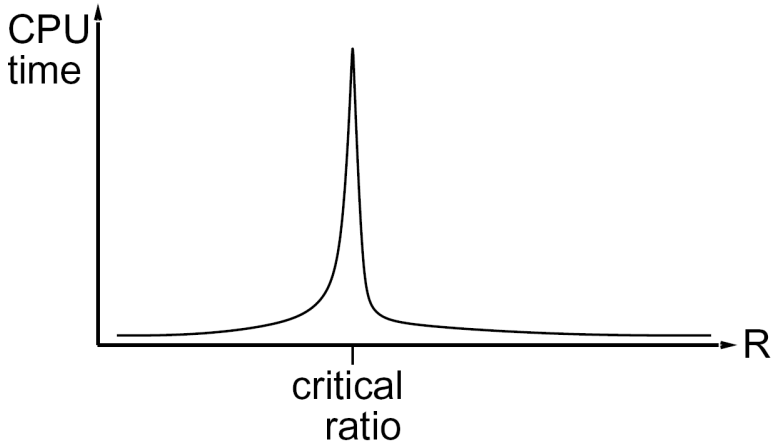


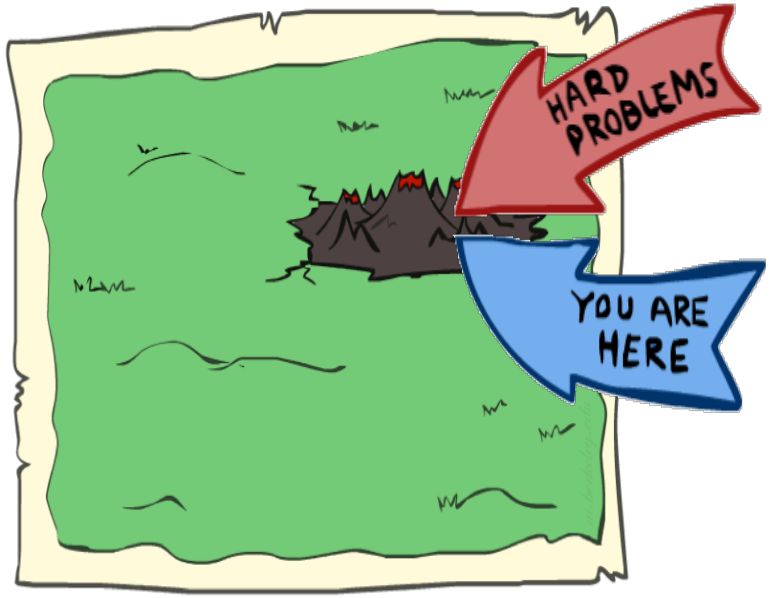
最小冲突算法的表现性能

- 给定随机初始化状态下，几乎可以在常量时间里以高成功概率求解 n 为任意数的 n-皇后问题， (例如, n = 10,000,000)!

- 同样的表现性能对任意随机产生的 CSP都适用，除了在一个很窄的比例范围内

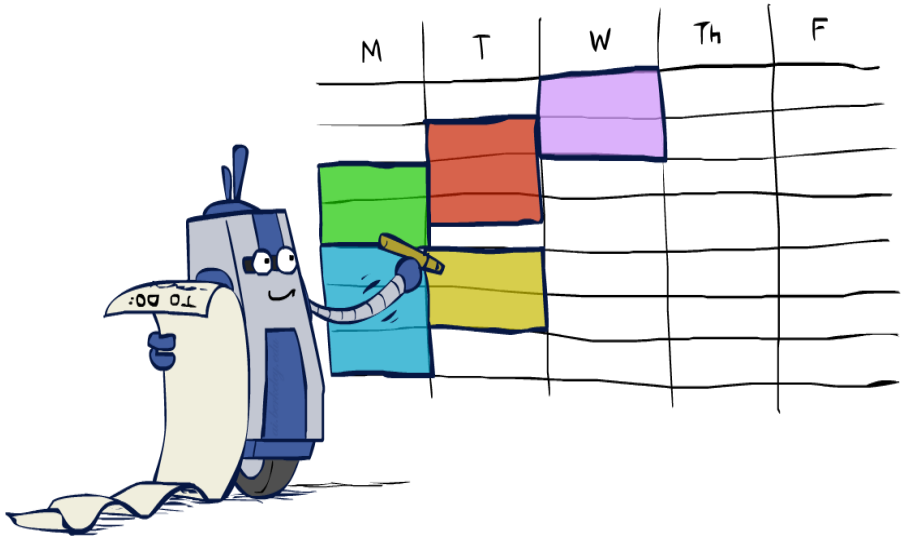
$$R = \frac{\text{约束条件数}}{\text{变量数}}$$





总结：约束满足问题

- 一类特殊的搜索问题:
  - 状态是变量的 (部分) 配值
  - 目标检测通过检查约束条件
- 基本算法: 回溯搜索
- 提速思路:
  - 排序选择
  - 过滤筛查
  - 结构利用



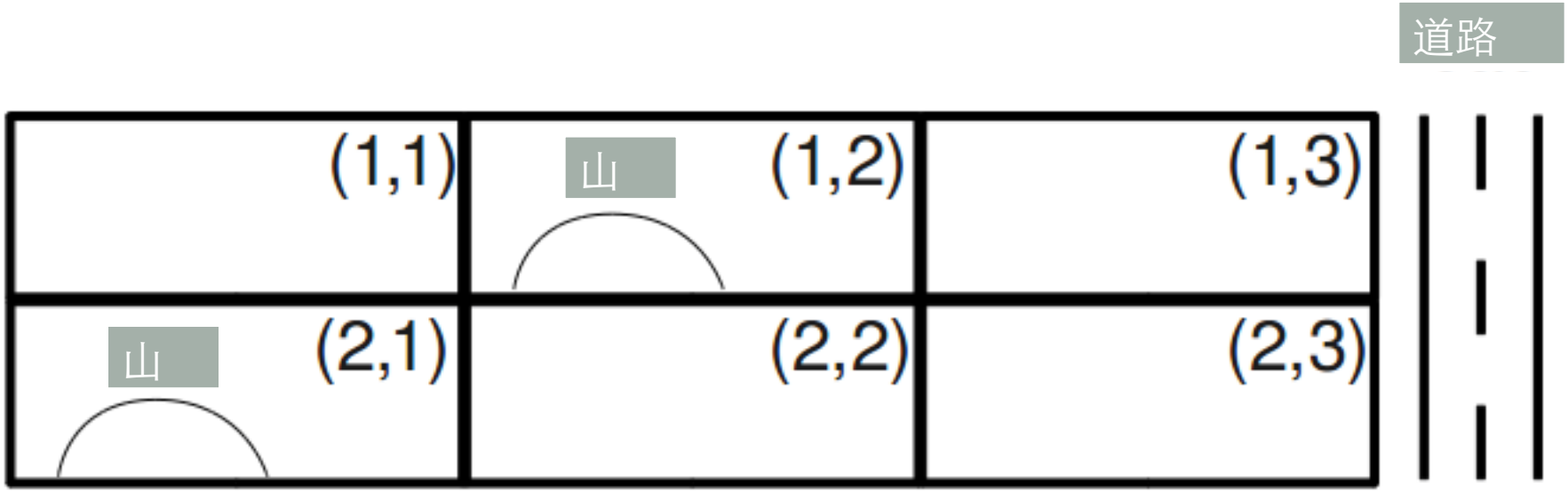
- 实践中，最小冲突法的局部算法经常很有效

举例：弧一致性

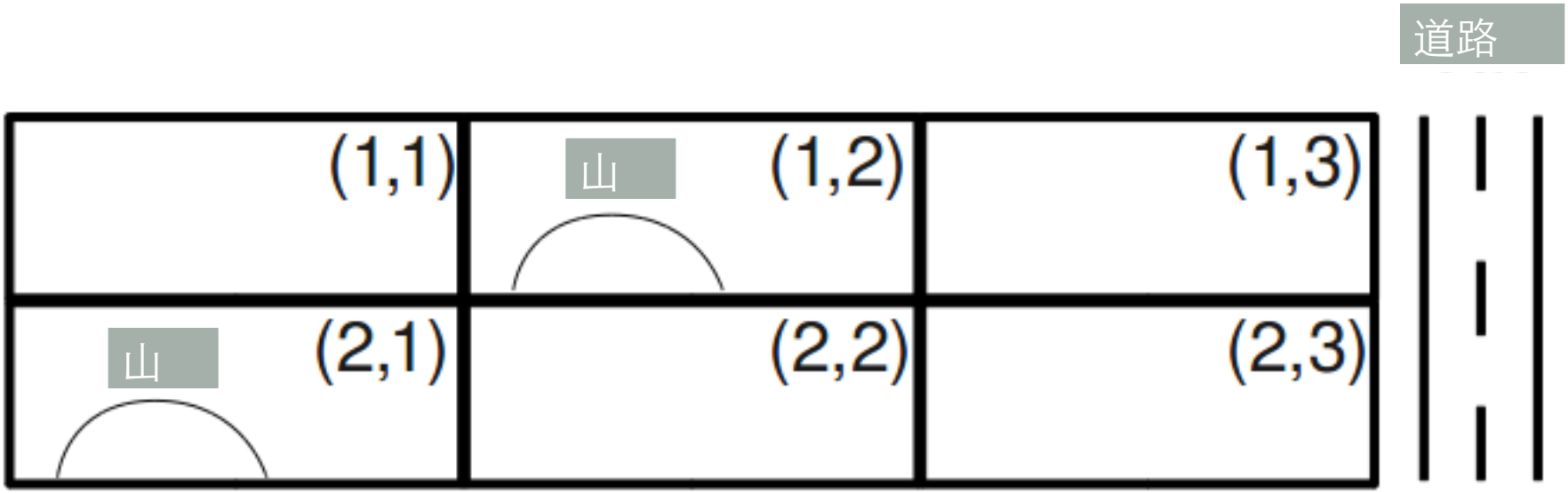
- 为会议安排演讲者的时间段。一共有3个演讲者（A，B, C），3个时间段(1, 2, 3)。约束是：
  - A，B，和C 需要在不同的时间段；
  - A < C
- 问题：
  - 在强化弧A→ C的一致性后，每个变量的值域是什么?
    - A(1,2); B(1,2,3); C(1,2,3)
  - 接着以上的结果，如果再强化弧 B→A 的一致性后，各个变量的值域是如何变化的?
    - A(1,2); B(1,2,3); C(1,2,3)
  - 继续从以上的结果出发，强化弧C→A 的一致性后，各变量的值域是什么?
    - A(1,2); B(1,2,3); C(2,3)

举例：校园规划

- 由你来决定一个校园中新区域的规划设计。这个新区域中有四个建筑：一个行政管理楼（A），一个汽车站（B），一个教室楼（C), 和一个学生宿舍楼(D)。这些建筑必须坐落于以下的网格里。



举例：校园规划（继续）



- 规划必须满足以下约束：
  1. 汽车站(B)必须邻近道路
  2. 行政管理楼(A)和教室楼(C)必须邻近汽车站(B)
  3. 教室楼(C)必须邻近宿舍楼(D)
  4. 行政管理楼(A)一定不能靠近宿舍楼(D)
  5. 行政管理楼(A)一定不能建在一个山上
  6. 宿舍楼(D)一定要么建在一个山上，或者靠近道路
  7. 所有建筑必须在不同的网格里。
- 邻近的意思是建筑所在网格必须共享一个边，而不是一个角。

举例：校园规划（继续）

- 一元约束
  - 哪些约束是一元的?
  - 应用一元约束以后的各变量的值域是什么?
- 弧的一致性检查（AC-3算法）
  - 从以上的值域结果开始，强化弧的一致性。初始时，队列里包括所有的弧（按字母顺序排序）
  - 检查弧A→B的一致性后，A,B的值域是什么?
  - 从以上结果继续弧一致性的检查，证实检查A->C,A->D,B->A,B->C,B->D，C->A 不改变任何值域
  - 接下来检查C->B, 各变量值域如何变化？哪些弧被加入弧队列中?
  - 继续该算法，直至弧队列为空，此时各变量的值域是什么?

举例：校园规划（继续）

- 开始搜索
  - 使用最小剩余值启发信息来选择从哪个变量开始赋值?
  - 使用最少约束值启发信息来选择该变量的值?
  - 赋值后，再递归检查弧的一致性，然后各变量的值域是什么?
  - 此时是否找到了一个解？