Constraint Satisfaction Problems (CSPs)

School of Electronic and Computer Engineering
Peking University

Wang Wenmin

Contents

Contents

What are Constraint Satisfaction Problem (CSPs) 什么是约束满足问题

- CSPs are mathematical problems, defined as a set of objects whose state must satisfy a number of constraints or limitations.

  CSPs是数学问题，被定义为其状态必须满足若干约束和限制的一组对象。

- CSPs represent the entities in a problem as a homogeneous collection of finite constraints over variables, which is solved by constraint satisfaction methods.

  CSPs将问题中的实体表示为对变量进行有限约束的同质集合，通过约束满足方法加以解决。

- CSPs are the research subject in both artificial intelligence and operations research,

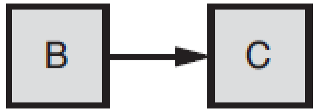
  CSPs是人工智能和运筹学共同的研究课题，

  - since the regularity in their formulation provides a common basis to analyze and solve problems of many unrelated families.

    这是因为其形式化的规则性提供了用于分析和解决许多不相关问题的共同基础。

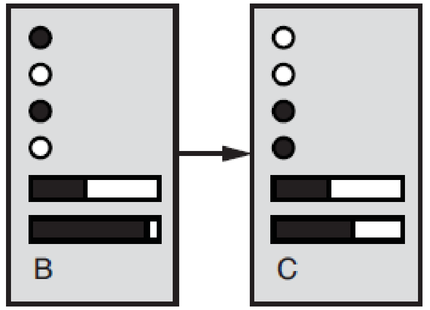
Standard Search vs. Constraint Satisfaction Problem 标准搜索与约束满足问题

- Standard search problem 标准搜索问题
  - The state is atomic or indivisible, a "black box" with no internal structure.
    其状态是原子的或不可分割的，一个没有内部结构的黑盒子。



Atomic

- Constraint satisfaction problem 约束满足问题
  - The state is a factored representation, a set of variables, each of which has a value.
    状态采用因子表示，是一系列变量，每个有相应的值。
  - Take advantage of the structure of states.
    发挥状态结构的长处。
  - Use *general-purpose* rather than *problem-specific* heuristics.
    采用一般用途而不是问题特有的启发式。



Factored

Why Study Constraint Satisfaction Problem 为什么研究约束满足问题

- CSPs often exhibit high complexity, requiring a combination of heuristics and combinatorial search methods.

  CSP通常呈现出高复杂性，需要将启发式与组合搜索方法相结合。

- Some certain forms of the CSP: 某些形式的CSP：

Boolean Satisfiability Problem (SAT)	■	布尔可满足性问题
Satisfiability Modulo Theories (SMT)	■	可满足性模理论
Answer Set Programming (ASP)	■	答案集编排

- Examples that can be modeled as a CSP: 可建模为CSP的例子：

8-queens puzzle	■	8皇后问题
Map coloring problem	■	地图着色问题
Cryptarithmetic	■	密码算术
Sudoku	■	数独

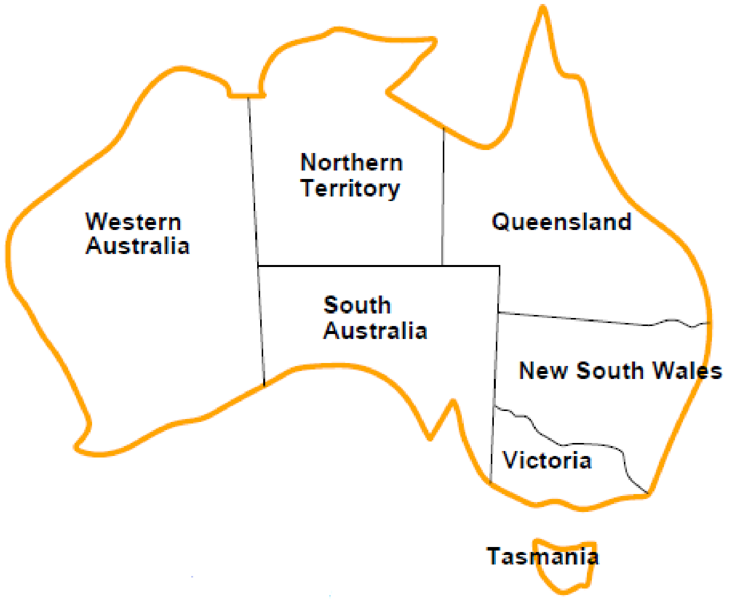
Formal Definition of CSP CSP的形式化定义

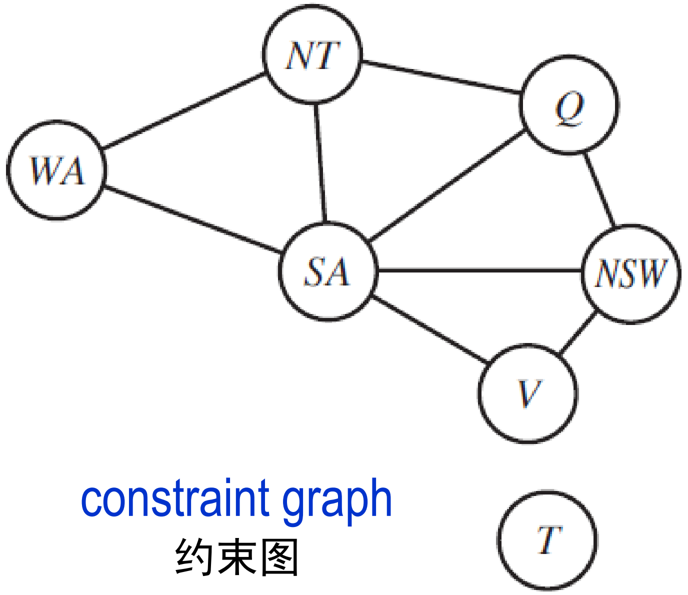
- A CPS is defined as a triple <X, D, C>,where:
  一个CSP被定义为一个三元组<X, D, C>，其中：
  - X is a set of variables, $\{X_1,\ldots, X_n\}$
    X为变量的集合，$\{X_1,\ldots, X_n\}$。
  - D is a set of domains, $\{D_1,\ldots, D_n\}$. One D_i for each variable X_i ,consisting of a set of values $\{V_1,\ldots, V_k\}$
    D为范畴的集合，$\{D_1,\ldots, D_n\}$。每个D_1对应一个变量X_1，包含一组值$\{V_1,\ldots, V_k\}$
  - C is a set of constrains, $\{C_1,\ldots, C_m\}$. Each C_i consists of a pair <scope, rel> where, scope is a tuple of variables that participate in the constraint, rel is a relation that defines the values that those variables can take on.
    C为约束的集合，$\{C_1,\ldots, C_m\}$。每个约束C_1包含一个二元组<scope, rel>，其中，scope为参与约束的一组变量，rel是一个关系，定义这些变量可以接受的值。

Formal Definition of CSP CSP的形式化定义

- To solve a CSP, we need to define a state space and the notion of a solution.
  求解一个CSP，我们需要定义一个状态空间和解的概念。
- Each state is defined by assignment of values to variables: $\{X_i = v_i, X_j = v_j , \dots\}$.
  每个状态都是通过对变量赋值的形式来定义的：$\{X_i = v_i, X_j = v_j , \cdots\}$。
  - Consistent assignment 一致性赋值
    a legal assignment that does not violate any constraints.
    一种不违反任何约束的合法赋值。
  - Complete assignment 完整性赋值
    every variable is assigned, and the assignment is consistent, complete.
    每个变量都被赋值，并且该赋值是一致的、完整的。
  - Partial assignment 局部性赋值
    assigns values to only some of the variables.
    仅对某些变量赋进行赋值。

Example: Map Coloring 地图着色

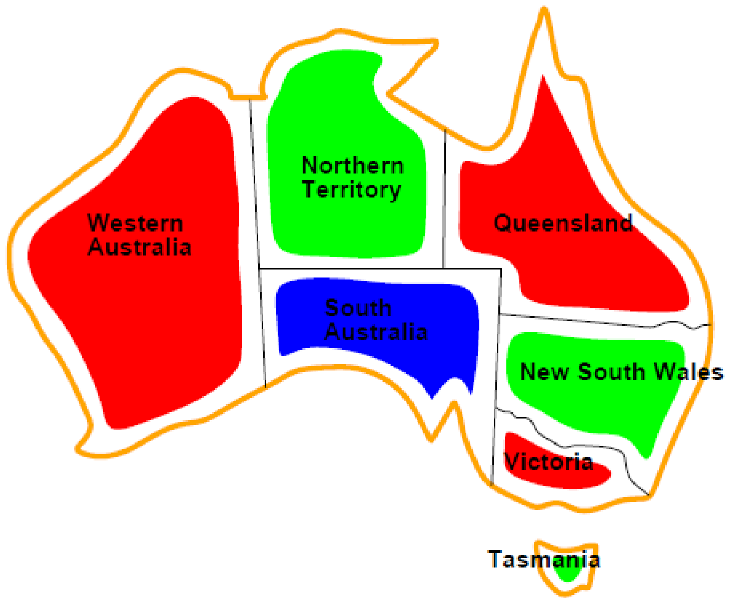


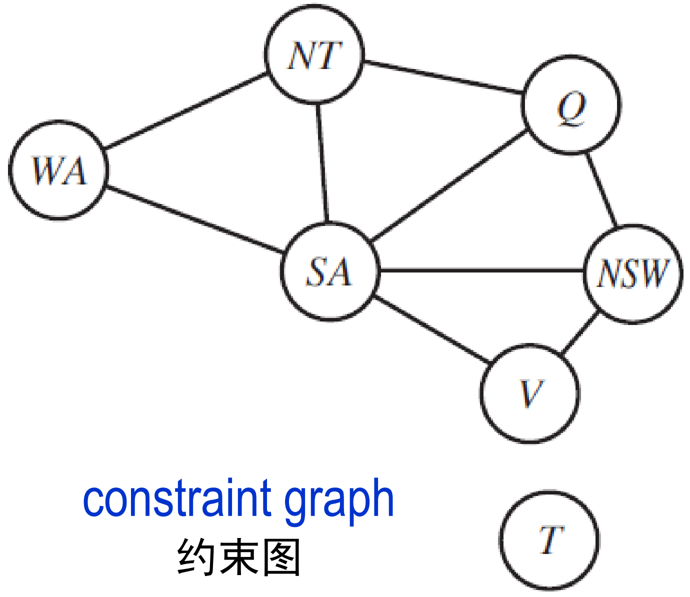


constraint graph
约束图

- Variables: 变量：
  $X = \{WA, NT, Q, NSW, V, SA, T\}$
- Domains: 范畴：
  $D_i = \{red, green, blue\}$.
- Constraints: neighboring regions should have different colors, i.e., 约束：相邻区域必须有不同的颜色，例如，
  $C = \{SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V, WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V\}$

Example: Map Coloring 地图着色





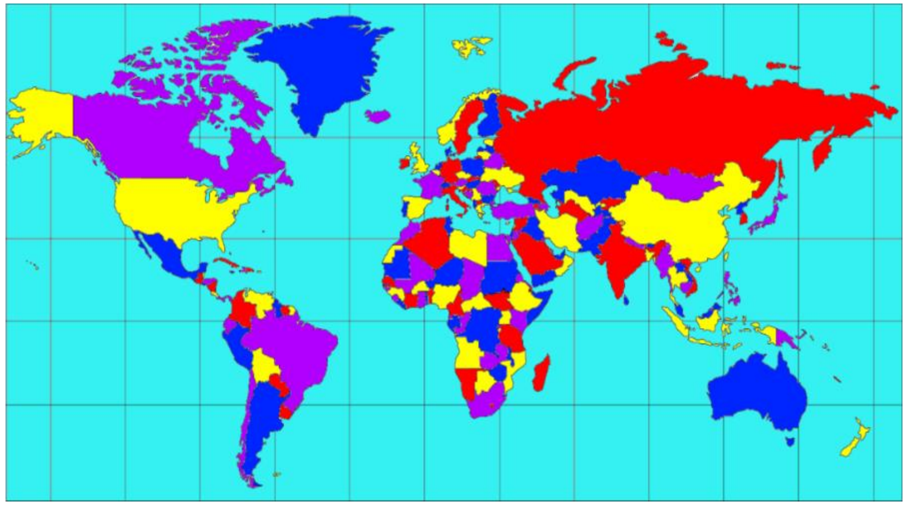
constraint graph
约束图

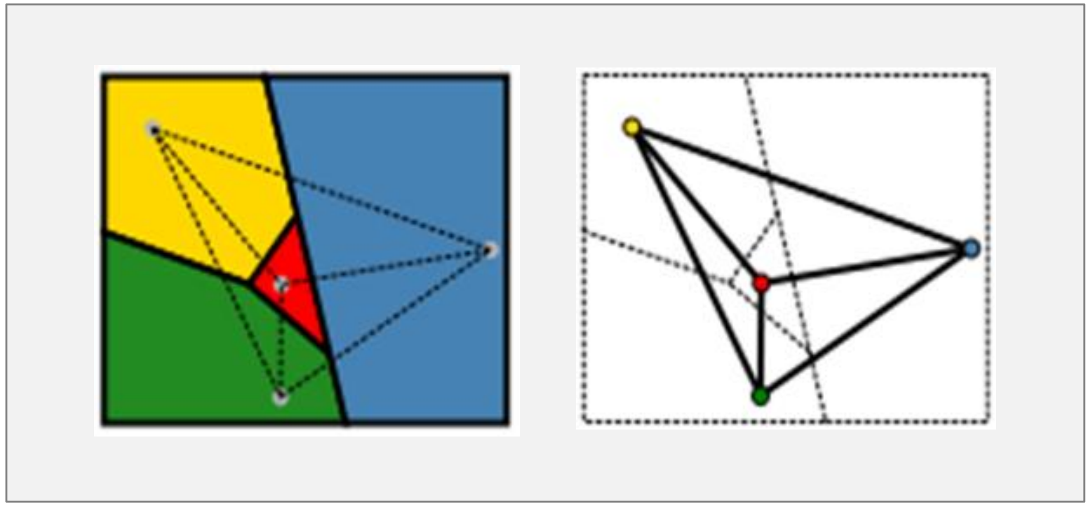
- *SA ≠ WA*, a shortcut for <*(SA, WA), SA ≠ WA*>, can be enumerated in turn as:
  SA ≠ *WA*，是<*(SA, WA), SA ≠ WA*>简写，可以有如下6种组合：
  {(red, green),(red,blue),(green, red),......,(blue,green)}
- Many possible solutions to this problem, e.g.,
  这个问题有许多种可能的解，例如：
  {(WA=red, NT=green,Q=red, NSW=green, V=red, SA=blue, T=green).}

Appendix: Four Color Theorem 四色定理

- Given any separation of a plane into contiguous regions, called a map, the regions can be colored using at most four colors so that no two adjacent regions have the same color.

  给定任意平面分割而成的相邻区域，称为地图，这些区域可以使用至多四种颜色加以着色，以致于不存在两个邻近区域具有同样的颜色。


Map of the world using just four colors
仅用四种颜色的世界地图


Precise formulation of the theorem
该定理的精确表述

Example: Job-shop scheduling 作业车间调度

- Factories have the problem of scheduling jobs, subject to various constraints.
  工厂有调度作业的问题，受各种因素制约。
- Consider the problem of scheduling the assembly of a car:
  考虑调度汽车装配的问题：
  - whole job is composed of tasks, each task is as a variable, where the value of each variable is the time that the task starts.
    整个工作由任务组成，每个任务作为一个变量，其中每个变量的值为该任务开始的时间。
  - Constraints: 约束条件：
    - one task must occur before another,
      一个任务必须在其它任务之前发生，
    - a task takes a certain amount of time to complete.
      任务需要一定的时间来完成。

Example: Job-shop scheduling 作业车间调度

- Consider a part of the car assembly, consisting of 15 tasks:
  设有一部分汽车装配问题，包含15个任务：
  - install two axles (front and back), 安装两个车轴（前、后），
  - affix all four wheels (right, left, front and back), 装上四个车轮（左、右、前、后），
  - tighten nuts for each wheel, 拧紧每个车轮的螺母，
  - affix four hubcaps, and 装上四个轮毂罩，
  - inspect the final assembly. 检查最后的组装。
- The tasks can be represented with 15 variables:
  该任务可以用15个变量表示：

$$X = \{Axle_F, Axle_B, Wheel_{RF}, Wheel_{LF}, Wheel_{RB}, Wheel_{LB}, \\ Nuts_{RF}, Nuts_{LF}, Nuts_{RB}, Nuts_{LB}, Cap_{RF}, Cap_{LF}, Cap_{RB}, Cap_{LB}, Inspect\}$$

Example: Job-shop scheduling 作业车间调度

- Next represent precedence constraints between tasks.
  下面表示任务间优先级约束条件：
  - The axles have to be in place before wheels are put on (10 minutes), so:
    在车轮安装之前（10分钟）车轴必须就位：

$$Axle_F + 10 \leq Wheel_{RF}\,; \quad Axle_F + 10 \leq Wheel_{LF}\,;$$
$$Axle_B + 10 \leq Wheel_{RB}\,; \quad Axle_B + 10 \leq Wheel_{LB}\,.$$

  - For each wheel, must affix the wheel (1 minute), then tighten nuts (2 minutes), and finally attach hubcap:
    对每个车轮，必须先装上该车轮（1分钟），再拧紧螺丝（2分钟），最后再盖上轮毂罩：

$$Wheel_{RF} + 1 \leq Nuts_{RF}\,; \quad Nuts_{RF} + 2 \leq Cap_{RF}\,;$$
$$Wheel_{LF} + 1 \leq Nuts_{LF}\,; \quad Nuts_{LF} + 2 \leq Cap_{LF}\,;$$
$$Wheel_{RB} + 1 \leq Nuts_{RB}\,; \quad Nuts_{RB} + 2 \leq Cap_{RB}\,;$$
$$Wheel_{LB} + 1 \leq Nuts_{LB}\,; \quad Nuts_{LB} + 2 \leq Cap_{LB}\,.$$

Example: Job-shop scheduling 作业车间调度

- Need a disjunctive constraint that $Axle_F$ and $Axle_B$ must not overlap in time; either one comes first or the other does:
  需要一个析取约束条件，即$Axle_F$ 和$Axle_B$ 时间上不能重叠；要么一个先做、或者另一个做。
  $$Axle_F + 10 \le Axle_B \quad \text{or} \quad Axle_B + 10 \le Axle_F$$
- Also need to assert that the inspection comes last and takes 3 minutes. For every variable except inspect we add a constraint of the form.
  还需要确保检验放在最后并且需要3分钟。除了检查每个变量，还要增加如下形式的约束：
  $$X + d_X \le \ Inspect$$
- Finally, suppose there is a requirement to get the whole assembly done in 30 minutes. We can achieve that by limiting the domain of all variables:
  最后，假设需要全部装配在30分钟完成。我们可以通过限制所有变量的范畴来实现：
  $$D_i = \{1, 2, 3, \ldots, 27\}$$

Why Formulate a Problem as a CSP

- ☐ CSP is a natural representation for a wide variety of problems.
  CSP是对很多种问题的一种自然表示。
- ☐ CSP-solving system is easier to solve a problem than another search technique.
  与其它搜索技术相比，CSP求解系统更容易解决问题。
- ☐ CSP solver can be faster than state-space searchers, since it can quickly eliminate large swatches of the search space.
  CSP求解系统比状态空间搜索更快，因为它可以快速排除大的搜索空间样本。

Example:

Once we chose $\{SA = blue\}$ in the Australia problem, we can conclude that none of the 5 neighboring variables can take on the value $blue$, therefore:
在澳大利亚问题中一经选择{*SA = blue*}，就可以得到结论，5个相邻变量不能再选择*Blue*，因此：

$$3^5 = 243 \text{ assignments} \quad \Longrightarrow \quad 2^5 = 32, \text{ a reduction of } 87\%.$$

Why Formulate a Problem as a CSP

- In regular state-space search 常规的状态空间搜索
  - (We can only ask) is this specific state a goal? No? What about this one?
    （我们仅能查询）这个特定状态是目标吗？ 不是？那么这一个是什么？
- With CSP 采用CSP
  - Once we find out that a partial assignment is not a solution, we can immediately discard further refinements.
    一旦我们发现某个局部赋值不是解的话，我们可以迅速放弃进一步的努力。
  - And, we can see *why* the assignment is not a solution —— we see which variables violate a constraint.
    并且，我们能看到为什么该赋值不是解 —— 我们查看哪个变量违反约束。
  - So, many problems can be solved quickly when formulated as a CSP.
    故，当形式化为一个CSP的话，许多问题都可以快速得到解决。

Variations on the CSP Formalism CSP形式的变体

Domains 范畴

	Discrete 离散	Continuous 连续
Finite 有限	Map coloring problem 地图着色问题	none
Infinite 无限	Set of integers or strings 整数或字符串集合	none

Constraints 约束

	unary 1元	*binary* 2元	*n-ary* n元
Linear 线性	$\langle (SA), SA \neq green \rangle$	$SA \neq WA$	$Alldiff\,(F, T, U, W, R, O)$
Nonlinear 非线性	no algorithm exists 算法不存在		

Cryptarithmetic 算式迷

- It is a type of mathematical game consisting of a mathematical equation among unknown numbers, whose digits are represented by letters.
  是一种数学游戏，由未知数字之间的数学等式组成，这些数字被表示成字母。
  OO
- The goal is to identify the value of each letter.
  其目标是识别出每个字母的值。

Cryptarithmetic is also known as:
算式迷还被称为：

Alphametics	■	字母算数
Verbal arithmetic	■	覆面算
Word addition	■	文字加法
Cryptarithm	■	隐算数

Example: Some Cryptarithmetic Problems 某些算式迷问题

- SEND + MORE = MONEY
- HALF + HALF = WHOLE
- THREE + THREE + ONE = SEVEN
- ONE + THREE + FOUR = EIGHT

- 客上天然居×4 = 居然天上客
- 海上明月×9 = 月明海上
- 车马炮×炮 = 相马炮
- 谜 + 字谜 + 数字谜 + 解数字谜 + 善解数字谜 = 巧解数字谜

Example: A Cryptarithmetic Formatted as a CSP

TWO + TWO = FOUR

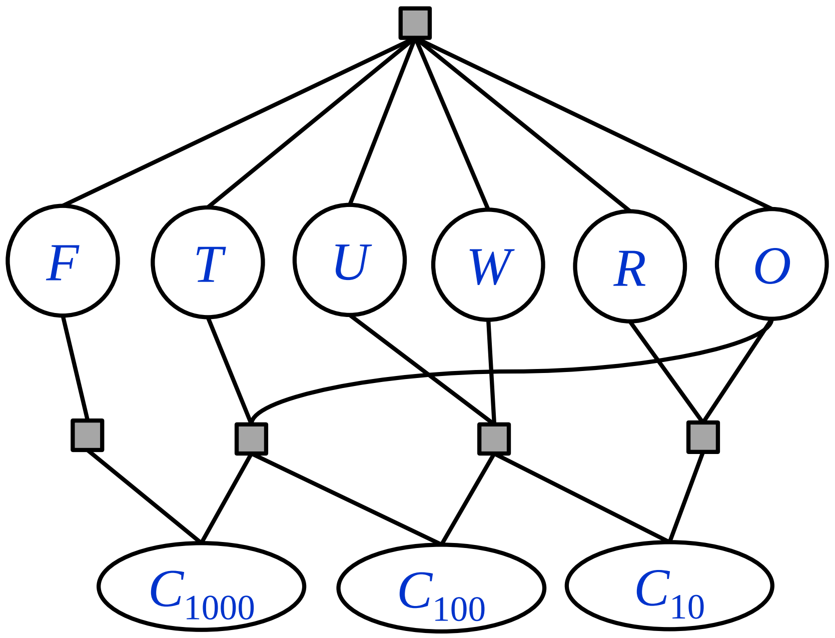
a solution 一个解：938 + 938 = 1876

- Variables 变量： $X = \{F, T, U, W, R, O, C_{10}, C_{100}, C_{1000}\}$
- Domains 范畴：$D_i = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- Global Constraints 全局约束： *Alldiff* (F, T, U, W, R, O)
- Addition constraints 附加约束：(*n-ary*)

$$O + O = R + 10 \cdot C_{10}$$
$$C_{10} + W + W = U + 10 \cdot C_{100}$$
$$C_{100} + T + T = O + 10 \cdot C_{1000}$$
$$C_{1000} = F$$

where C_{10}, C_{100}, and C_{1000} are auxiliary variables, representing the digit carried over into the tens, hundreds, or thousands column.

其中C_{10}、C_{100}、以及C_{1000} 是辅助变量，表示该数字进位到十位、百位、或者千位。



Constraint hypergraph
约束超图

A Theoretical Aspect of CSPs: Decision Problems CSPs的理论方面：决策问题

- CSPs are also studied in
  - *computational complexity theory* and *finite model theory*.

  CSPs也是计算复杂性理论和有限模型论所要研究的问题。
- Most classes of CSPs that are known to be tractable are those

  大多数被认为是易处理的CSPs是这样一些问题：
  - where the hypergraph of constraints has bounded tree width, or

    约束的超图具有有界的树宽，或者
  - where the constraints exist essentially *non-unary polymorphisms* of the set of constraint relations.

    约束本质上存在约束关系集的非一元多态性。
- Every CSP can also be considered as a *conjunctive query containment problem*.

  每个CSP也可以被看作是合取查询包含问题。

Real-World CSPs 现实世界的CSPs

- Many real-world problems involve real-valued variables:
  许多现实世界的问题涉及实值变量：

Assignment problems	■	分配问题
Timetabling problems	■	时间表问题
Hardware configuration	■	硬件配置
Transportation scheduling	■	运输调度
Factory scheduling	■	工厂调度
Circuit layout	■	电路布线
Floor planning	■	楼层规划
Fault diagnosis	■	故障诊断

Resolution of CSPs CSPs的解

- CSPs on finite domains are typically solved using a form of search.
  有限范畴中的CSPs问题通常采用某种形式的搜索来解决。
- The most used techniques are variants of
  最常用的技法如下：
  - constraint propagation,
    约束传播
  - backtracking,
    回溯
  - local search.
    局部搜索。

We will discuss the three techniques in next three sections

Thank you for your attention!

PoAI

Constraint Propagation: Inference in CSPs

School of Electronic and Computer Engineering
Peking University

Wang Wenmin

Contents

Overview of Constraint Propagation 约束传播概述

- Regular state-space search: 常规的状态空间搜索
  - can do only one thing, search.
    只能做一件事，搜索。
- CSPs: 约束满足问题
  - can do search, choose a new variable assignment from several possibilities,
    能做搜索，从若干可能性中选择一个新的变量赋值，
  - can also do a specific type of inference, called constraint propagation.
    还能做一种特殊类型的推理，称为约束传播。
- Constraint Propagation: 约束传播
  - uses the constraints to *reduce the number of legal values for a variable*, which in turn *can reduce the legal values for another variable*, and so on.
    采用约束来减少一个变量的合法值的数量，相应地可以减少其它变量的合法值，以此类推。

Overview of Constraint Propagation 约束传播概述

- ☐ Those techniques are used to modify a constraint satisfaction problem.
  这些技术被用于改变一个约束满足问题。
- ☐ More precisely, they enforce a form of local consistency, which are conditions related to the consistency of a group of variables and/or constraints.
  更精确地说，它们严格实施局部一致性，这种形式是一组变量和约束一致性相关的条件。
- ☐ Constraint propagation has various uses: 约束传播有各种用途：
  - ■ 1) turns a problem into one that is equivalent but is usually simpler to solve,
    将其转转化成等价但通常更易于求解的问题。
  - ■ 2) may prove satisfiability/unsatisfiability of problems.
    可以证明问题的可满足性/不可满足性。

  This always happens for some certain kinds of problems.
  对于某些特定类型的问题，这种情况总是发生。

Overview of Constraint Propagation 约束传播概述

- There are different types of local consistency:
  有不同类型的局部一致性：
  - Node consistency
    节点一致性
  - Arc consistency
    弧一致性
  - Path consistency
    路径一致性
  - *k*-consistency
    k一致性
- Most popular method is AC-3 algorithm which enforces arc consistency.
  最流行的约束传播方法是AC-3算法，它支持弧一致性。

  The name “AC-3” was used because it's the 3rd version.

Node Consistency 节点一致性

- Definition 定义

  A single variable (node) is node-consistent, if all the values in the variable's domain satisfy the variable's unary constraints.

  如果变量的范畴中所有值满足变量的一元约束，则单个变量（节点）是节点一致的。

Example:

- in the variant of the Australia map-coloring, where South Australians (SA) dislike green, i.e.,

  在简化的澳大利亚着色问题中，南澳大利亚人（SA）不喜欢绿色，即：

$$\langle (SA), SA \neq green \rangle$$

leaving SA with the reduced domain $\{red, blue\}$.

留给SA的为缩减范畴$\{red,\ blue\}$。

Arc Consistency 弧一致性

- Definition 定义
  A variable is arc-consistent, if every value in its domain satisfies the variable's binary constraints.
  如果范畴中的每个值满足变量的二元约束，则该变量是弧一致的。

Example:

- in the variant of the Australia map-coloring, the constraint $SA \neq WA$, we can write this constraint explicitly as,
  在简化的澳大利亚着色问题中，该约束SA ≠ WA，我们可将这个约束显式地写成：

$$\langle (SA, WA), \{(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)\}\rangle$$

Arc Consistency Algorithm AC-3 弧一致性算法AC-3

```
function AC-3(csp) returns false if an inconsistency is found and true otherwise
    inputs: csp, a binary CSP with components $(X, D, C)$
    local variables: queue, a queue of arcs, initially all the arcs in csp
    while queue is not empty do
        $(X_i, X_j) \leftarrow$ REMOVE-FIRST(queue)
        if REVISE(csp, $X_i$, $X_j$) then
            if size of $D_i = 0$ then return false
            for each $X_k$ in $X_i$.NEIGHBORS - $\{X_j\}$ do
                add $(X_k, X_i)$ to queue
    return true
```

```
function REVISE(csp, $X_i$, $X_j$) returns true iff we revise the domain of $X_i$
    revised $\leftarrow$ false
    for each $x$ in $D_i$ do
        if no value $y$ in $D_j$ allows $(x, y)$ to satisfy the constraint between $X_i$ and $X_j$ then
            delete $x$ from $D_i$
    revised $\leftarrow$ true
    return revised
```

Path Consistency 路径一致性

- Definition 定义

  A two-variable set $\{X_i, X_j\}$ is path-consistent with respect to a third variable X_m if, for every assignment $\{X_i = a, X_j = b\}$ consistent with the constraints on $\{X_i, X_j\}$, there is an assignment to X_m that satisfies the constraints on $\{X_i, X_m\}$ and $\{X_m, X_j\}$.

  对于与 $\{X_i, X_j\}$ 约束一致的每个赋值 $\{X_i = a, X_j = b\}$ ，如果存在一个满足对$\{X_i, X_m\}$和$\{X_m, X_j\}$约束的X_m 赋值，则二元变量集 $\{X_i, X_j\}$ 对于第三个变量X_m 是路径一致的。

- Why called path consistency

  为什么称为路径一致

  - because it as looking at a path from X_i to X_j with X_m in the middle.

    因为它看起来是一条从X_i到X_j、中间为X_m的路径。

k-Consistency k一致性

- Definition 定义

  A CSP is k-consistent if, for any set of $k-1$ variables and for any consistent assignment to those variables, a consistent value can always be assigned to any k^{th} variable.

  对于任意k – 1个变量的集合以及对于这些变量的任意一致性赋值，如果某个一致性值总是能够被赋值于任意第k个变量，则该CSP是k一致性的。

- The notion of k-consistency is the stronger forms of propagation.

  k一致性的概念更强的传播形式。

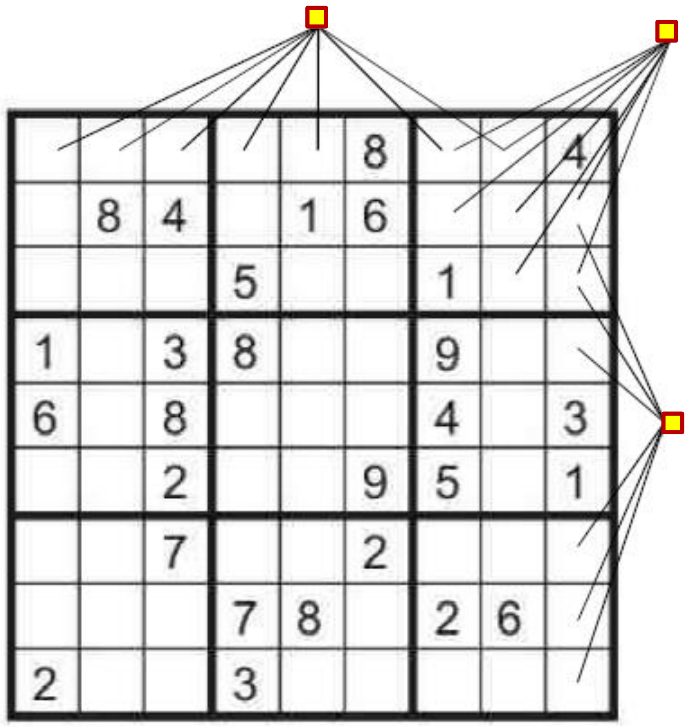
- Features of k-consistency k一致性的特点
  - $k = 1$: same as *node consistency*. k=1： 等同于节点一致性
  - $k = 2$: same as *arc consistency*. k=2： 等同于弧一致性
  - $k = 3$: same as *path consistency*. k=3： 等同于路径一致性

Example: Sudoku 数独

- Sudoku is a logic based, combinatorial number placement puzzle.
  数独是一种基于逻辑的组合数填空难题。
- The objective is to fill a 9×9 grid with digits, so that each column, each row, and each of the nine 3×3 sub-grids that compose the grid contains all of the digits from 1 to 9.
  其目的是用数字填满9x9个网格，使得每一行、每一列、以及9个3x3的子网格的每一个都包含从1到9的所有数字。

History

- Number puzzles appeared in French in late 19th century.
  19世纪后叶，数字谜题出现在法国
- It was introduced by Japan in April 1984.
  日本于1984年4月引入该数字谜题。
- London began featuring Sudoku in 2004.
  伦敦于2004年开始关注数独。



Example: Sudoku 数独

- Variables: 变量：

$$A_1, A_2, \dots, A_9$$
$$B_1, B_2, \dots, B_9$$
$$\dots$$
$$I_1, I_2, \dots, I_9$$

- Domains: 范畴：

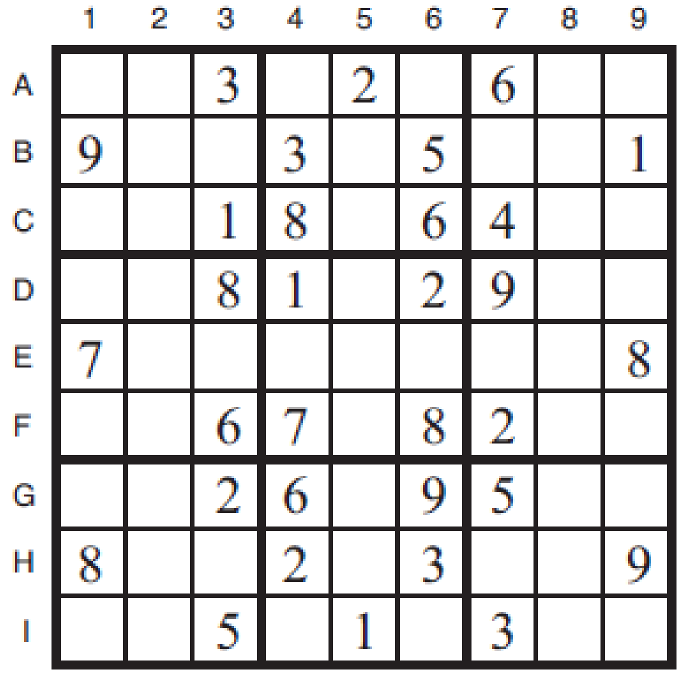
$$\{1, 2, \dots, 9\}$$

- Constraints: 约束：

row ： 行： $Alldiff\,(A_1, A_2, \dots, A_9)$, …, $Alldiff\,(I_1, I_2, \dots, I_9)$

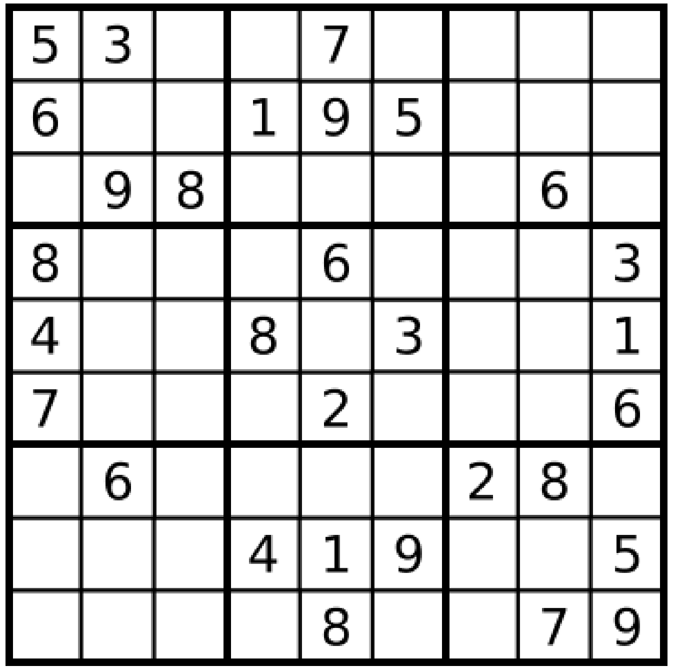
column : 列： $Alldiff\,(A_1, B_1, \dots, I_1)$, …, $Alldiff\,(A_9, B_9, \dots, I_9)$

region ： 区域： $Alldiff\,(A_1, A_2, A_3, \dots, C_3)$, …, $Alldiff\,(G_7, G_8, G_9, \dots, I_9)$

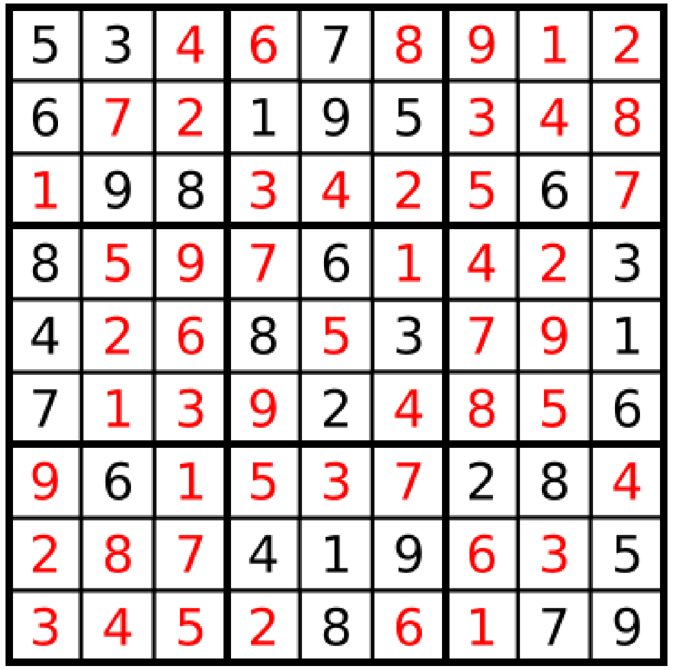


	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

Example: Sudoku 数独



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

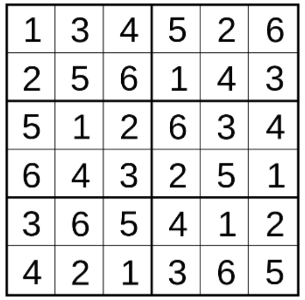


5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

A Sudoku puzzle (9×9) and its solution

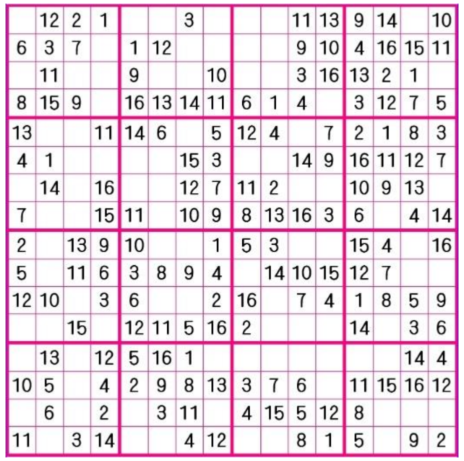
一个数独的谜题（9×9）和它的谜底

Example: Variants of Sudoku 数独的变体

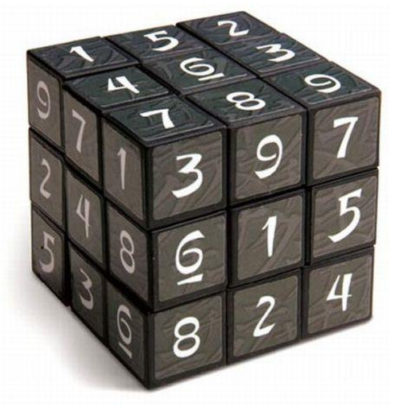


1	3	4	5	2	6
2	5	6	1	4	3
5	1	2	6	3	4
6	4	3	2	5	1
3	6	5	4	1	2
4	2	1	3	6	5

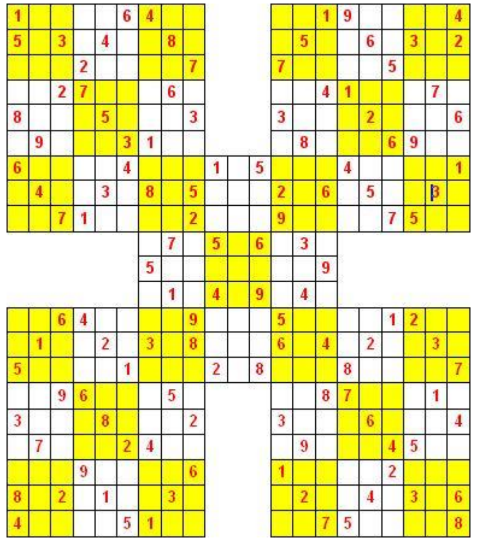
6×6



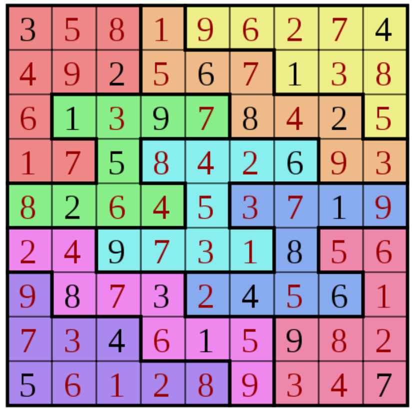
16×16



9×9

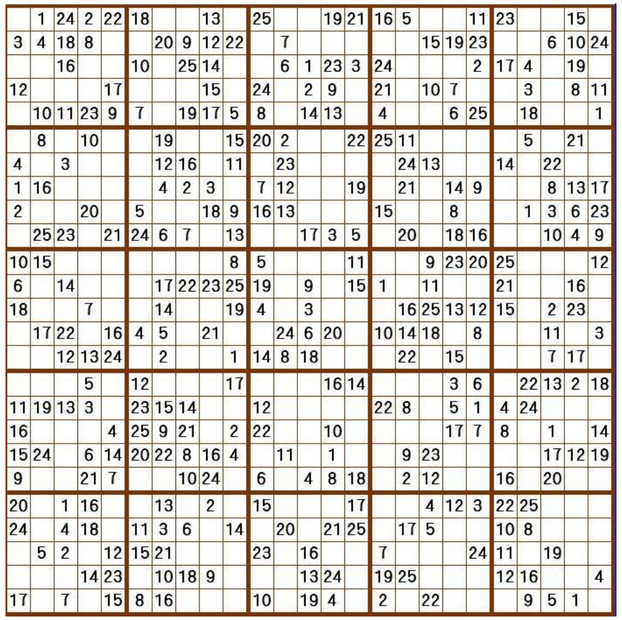


9×9×5



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

9×9



25×25

Thank you for your attention!

PoAI

Backtracking Search for CSPs

School of Electronic and Computer Engineering
Peking University

Wang Wenmin

Contents

Overview of Backtracking Search 回溯搜索概述

- It is a general algorithm on depth-first search, used for finding solutions to some computational problems, notably CSPs.

  是一种深度优先搜索的通用算法，用于查找某些计算问题的答案，尤其是CSPs。

- Backtracking search incrementally builds candidates to the solutions, and abandons each *partial candidate* c (backtracks), as soon as it determines that c cannot possibly be completed to a valid solution.

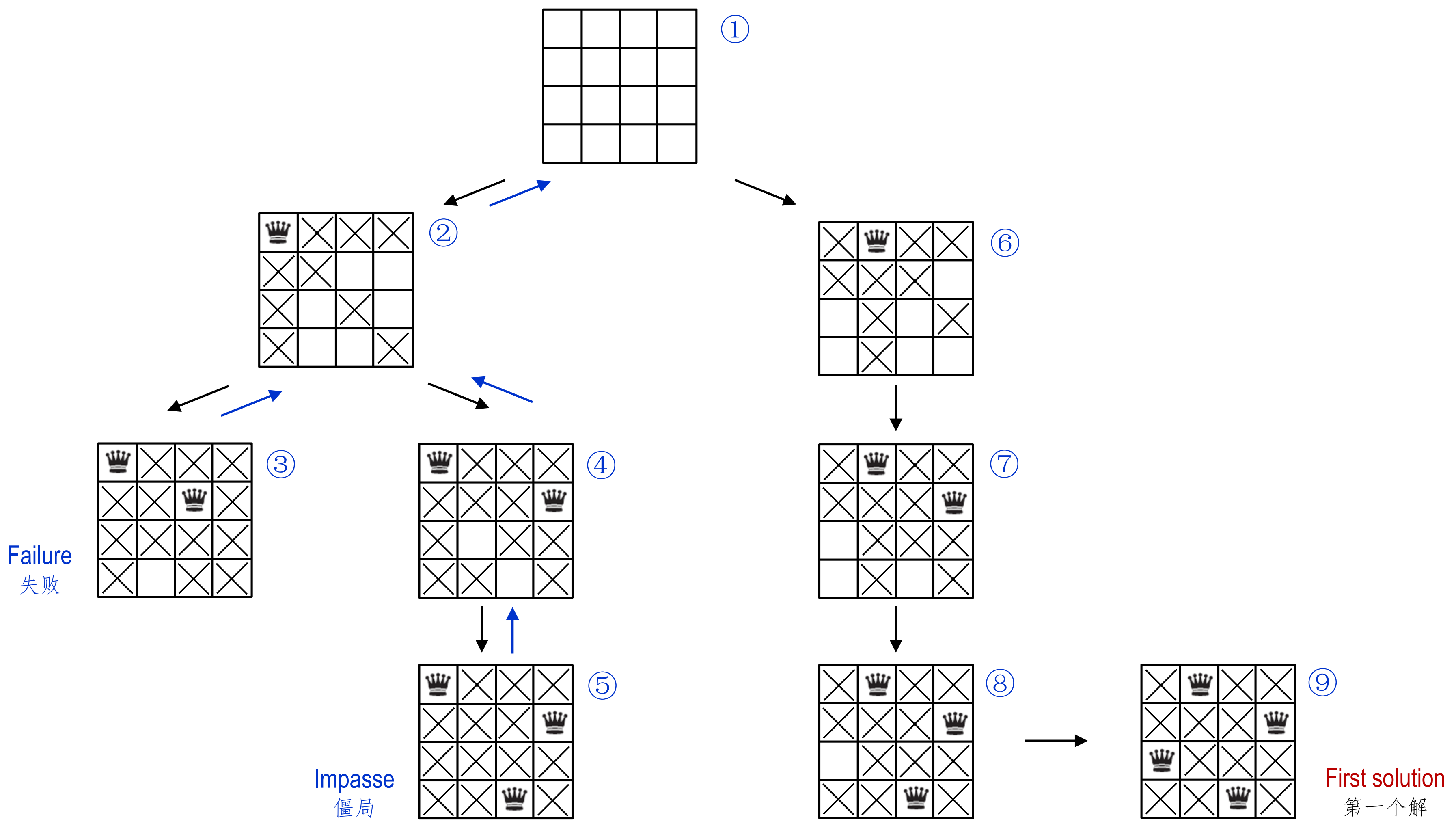
  回溯搜索递增地构建解的候选，而且一旦确定部分候选c不能成为一个合法的解，就将c抛弃(回溯）。

Example: 8-queens puzzle 8皇后难题

- In the common backtracking approach, the partial candidates are arrangements of k queens in the first k rows of the board, all in different rows and columns.

  常见的回溯方法，部分候选是在棋盘的在前k行上布局k个皇后，所有这些要在不同的行与列上。

Example: 4-queen problem 4皇后问题



Overview of Backtracking Search 回溯搜索概述

- It is the basic uninformed algorithm, a depth-first search with two improvements, for solving CSPs.
  是基本的无信息算法，一种具有两种改进的深度优先搜索，用于求解CSPs问题。
- *Idea* 1: One variable at a time 构思1：每次一个变量
  - Variable assignments are commutative, i.e.,
    变量赋值是可交换的，例如
    $[WA = red$ then $NT = green]$, same as $[NT = green$ then $WA = red]$
- *Idea* 2: Check constraints as you go 构思2：检查所需约束
  - I.e., consider only values which do not conflict previous assignments
    即，仅需考虑与前面赋值不发生冲突的值
  - Might have to check the constraints. “Incremental goal test”
    也许需要检查该约束。递增式目标检测

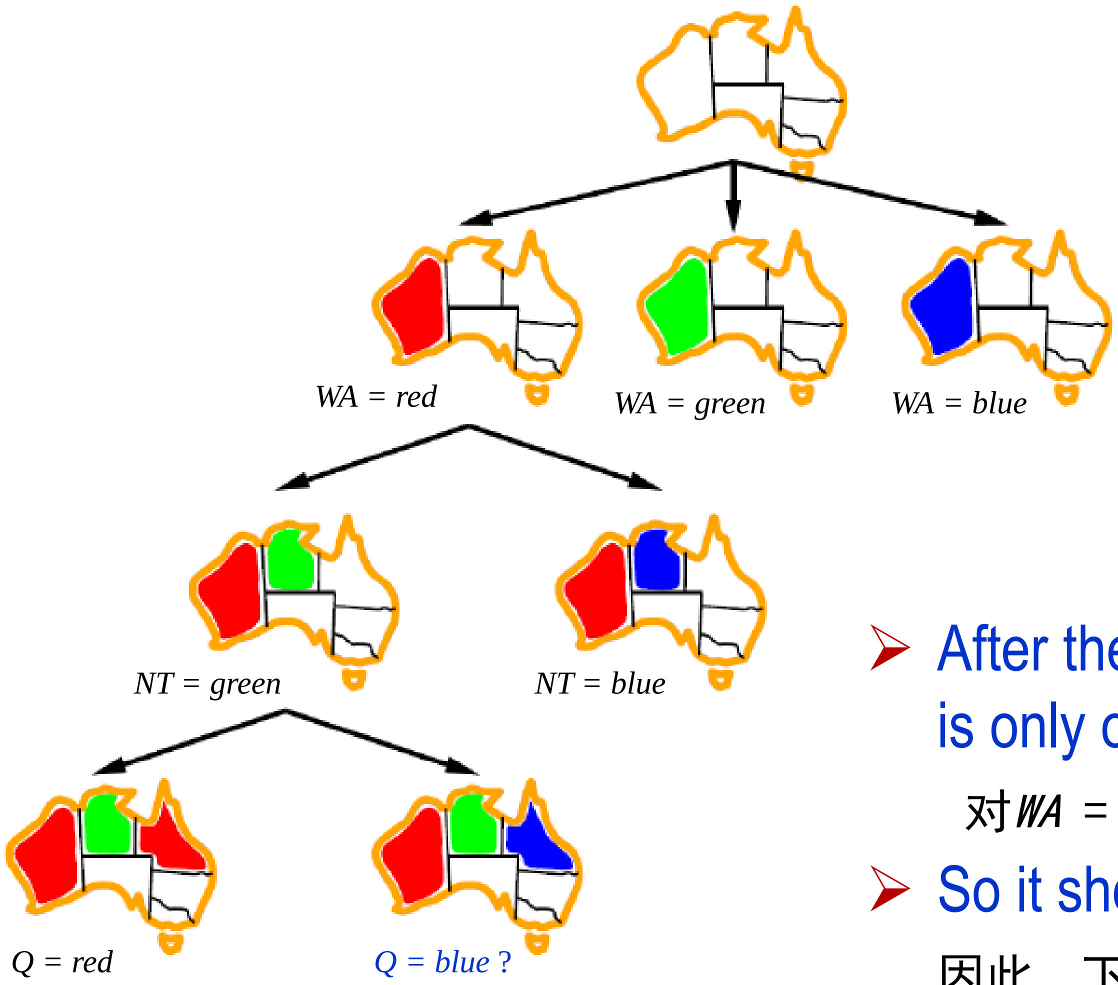
A Simple Backtracking Algorithm for CSPs 一个简单的CSPs回溯算法

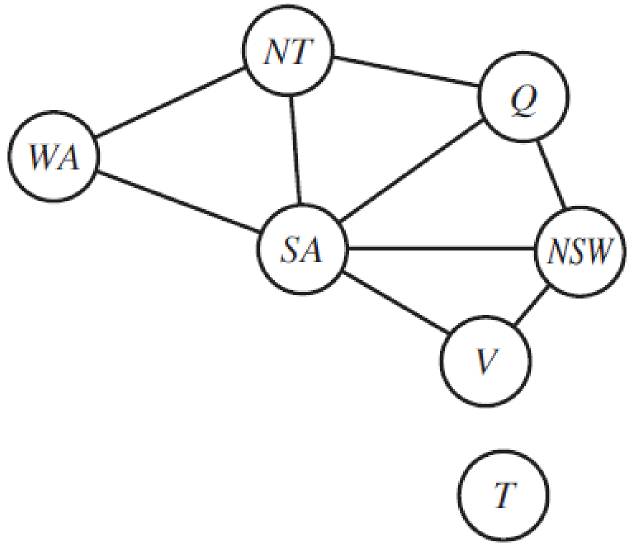
```
function BACKTRACK-SEARCH(csp) returns a solution, or failure
    return BACKTRACK({ }, csp)
function BACKTRACK(assignment, csp) returns a solution, or failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(csp)
    for each value in ORDER-DOMAIN-VALUES(var,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

Questions to Improve Backtracking 改善回溯的若干问题

- Question 1: 问题1：
  - Which variable should be assigned next? 下一步哪个变量应该被赋值?
    (SELECT-UNASSIGNED-VARIABLE)
  - And, in what order should its values be tried? 并且，尝试值应该以什么顺序?
    (ORDER-DOMAIN-VALUES)
- Question 2: 问题2：
  - What inferences should be performed at each step? 每一步应该完成什么推理?
    (INFERENCE)
- Question 3: 问题3：
  - When the search arrives at an assignment that violates a constraint, can the search avoid repeating this failure? 当搜索到一个违反约束的赋值时，该搜索是否能避免重复这个失败?

Discussion 1: Variable and Value Ordering 变量与值的排序





- After the assignments for *WA* = *red* and *NT* = *green*, there is only one possible value for *SA*.

  对*WA* = *red*和*NT* = *green*赋值之后，对*SA*来说仅剩一个可能的值。

- So it should assign *SA* = *blue* next rather than assigning *Q*.

  因此，下一步应该赋值*SA* = *blue* 而不是赋值*Q*。

Discussion 1: Variable and Value Ordering 变量与值的排序

The heuristic strategies of ordering 排序的启发式策略

- ☐ Minimum-remaining-values (MRV) 最小剩余值

  to choose the variable with the fewest “legal” values, also has been called the “most constrained variable”.

  选择具有最少“合法”值的变量，也被称为“最受约束变量”。

- ☐ Degree heuristic 程度启发式

  to reduce the branching factor on future choices by selecting the variable that is involved in the largest number of constraints on other unassigned variables.

  通过选择参与其它未分配变量的最大约束数，来减少未来选择的分支因子。

- ☐ Least-constraining-value heuristic 最少约束值启发式

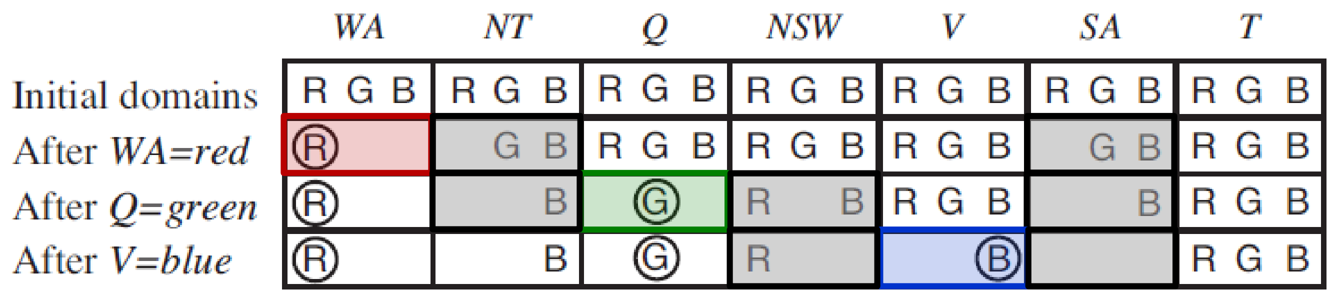
  to prefer the value that rules out the fewest choices for the neighboring variables in the constraint graph.

  尽量选取能排除约束图中相邻变量最少选择的值。

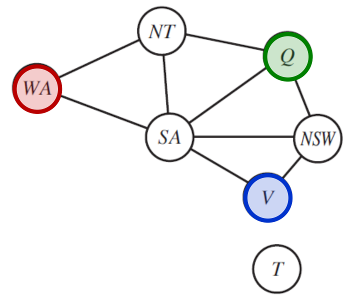
Discussion 2: Inference in Search 搜索中的推理

- The inference forward checking can be powerful in a search.
  前向检查推理在搜索中会很有用。

Example: Backtracking search with forward checking 具有前向检查的回溯搜索



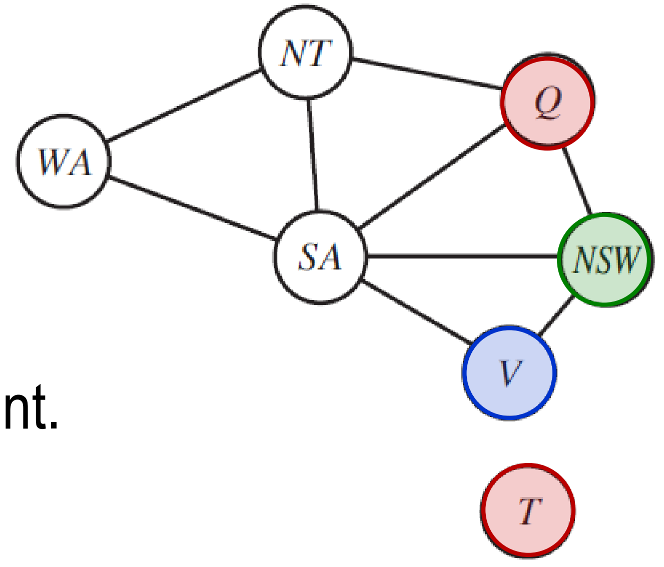
	WA	*NT*	*Q*	*NSW*	*V*	*SA*	*T*
Initial domains	R G B	R G B	R G B	R G B	R G B	R G B	R G B
After *WA=red*	Ⓡ	G B	R G B	R G B	R G B	G B	R G B
After *Q=green*	Ⓡ	B	Ⓖ	R B	R G B	B	R G B
After *V=blue*	Ⓡ	B	Ⓖ	R	Ⓑ		R G B



- *WA=red* is assigned first; then it deletes “R” from the domains of the neighboring variables *NT* and *SA*.
  WA=red先被赋值；然后从相邻变量NT和SA的范畴中删除“R”。
- After *Q=green* is assigned, “G” is deleted from the domains of *NT*, *SA*, and *NSW*.
  Q=green被赋值后，从NSW、SA以及NSW的范畴中删除“G”。
- After *V=blue* is assigned, “B” is deleted from the domains of *NSW* and *SA*, leaving *SA* with no legal values.
  V=blue被赋值后，从NSW和SA的范畴中删除“B”，剩下SA没有合法值。

Discussion 3: Intelligent Backtracking 智能回溯

- ☐ BACKTRACKING-SEARCH algorithm has a policy when a search fails: back up to the preceding variable and try a different value.
  BACKTRACKING-SEARCH算法搜索失败时有一个策略：回到先前的变量并尝试不同的值。
  - ■ This is called chronological backtracking.
    这被称为按时间回溯。
- ☐ E.g., for a variable ordering 例如，对于一个变量排序
  $\{Q, NSW, V, T, SA, WA, NT\}$,
  the partial assignment 该部分赋值
  $\{Q=red, NSW=green, V=blue, T=red\}$.
  - ■ Try next variable *SA*, we see that every value violates a constraint.
    试图对下一个变量*SA*赋值时，我们看到每个值都违反约束。
  - ■ Backjumping would jump over *T* and try a new value for *V*.
    后退跳跃法将跳过*T*并尝试将一个新的值赋给*V*.



Thank you for your attention!

PoAI

Local Search for CSPs

School of Electronic and Computer Engineering
Peking University

Wang Wenmin

PoAI

Contents

Local Search for CSPs CSPs的局部搜索

- Local search algorithms are also effective in solving many CSPs, using complete-state formulation:
  局部搜索算法在求解许多CSPs问题中也很有效，采用完整状态形式化：
  - Initial state: assign a value for each variable
    初始状态：为每个变量赋值
  - Search: change the value of one variable at a time
    搜索：一次改变一个变量的值

Example: 8-queens problem

- Initial state is a random configuration of 8 queens in 8 columns, and each step moves a single queen to a new position in its column.
  初始状态是在8列上随机地摆放8个皇后，然后每一步将一个皇后移动到该列上的新位置。
- Typically, the initial guess violates several constraints, point is to eliminate the violated constraints.
  通常，初始状态会违反若干约束。要点是消除这些违反的约束。

Min-conflicts Heuristic 最小冲突启发式

- A most obvious heuristic is to select a new value for a variable.
  一个最明显的启发式是对一个变量选择一个新的值。
- Features 特点
  - Variable selection: randomly select any conflicted variable
    变量选择：随机选择任意一个冲突变量
  - Value selection: select new value that results in a minimum number of conflicts with others
    值的选择：选择导致与其它变量呈现最少冲突的新值
- Surprisingly effective for many CSPs 对许多CSPs出奇的有效
  - on *n*-queens problem, the run time of min-conflicts is roughly *independent of problem size*.
    在*n*皇后问题上，最小冲突的运行时间与问题大小基本上无关。
  - even on *million*-queens problem, it solves in an average of 50 steps, after the initial assignment!
    甚至对于百万皇后问题，在初始赋值后，平均50步左右就能得到解！

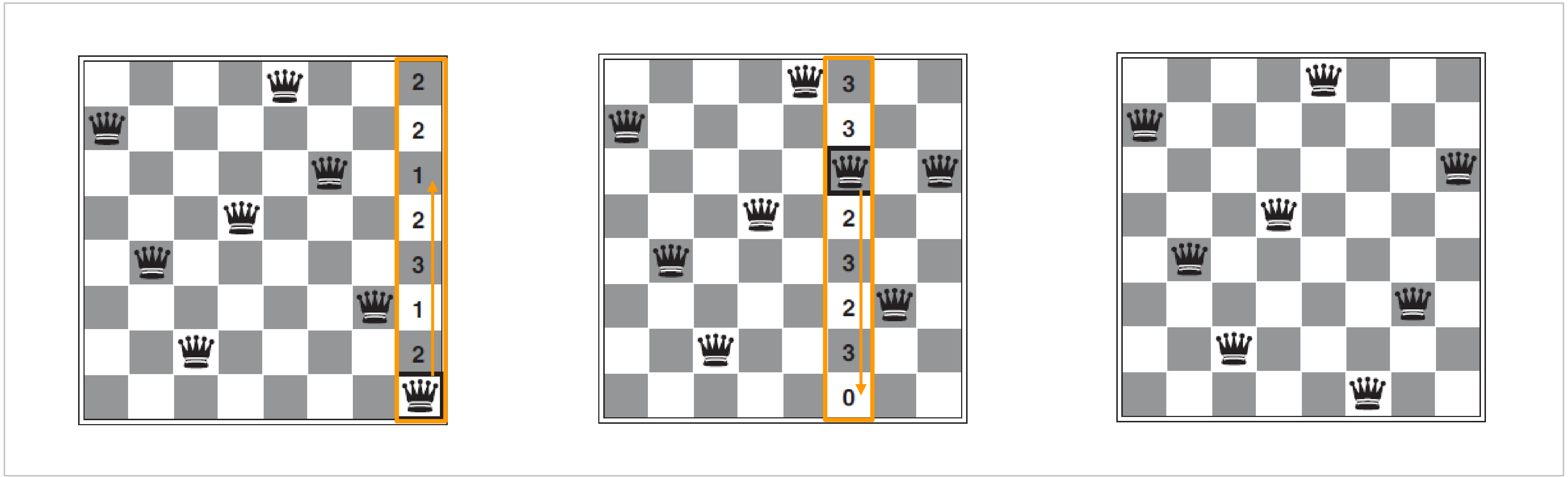
Algorithm Solving CSPs by Local Search 局部搜索求解CSPs算法

```
function MIN-CONFLICTS(csp, max_steps) returns a solution or failure
    inputs: csp, a constraint satisfaction problem
                max_steps, the number of steps allowed before giving up
    current ← an initial complete assignment for csp
    for i = 1 to max_steps do
        if current is a solution for csp then return current
        var ← a randomly chosen conflicted variable from csp.VARIABLES
        value ← the value v for var that minimizes CONFLICTS(var, v, current, csp)
        set var = value in current
    return failure
```

The initial state may be chosen randomly that chooses a min-conflict value for each variable in turn.
The CONFLICTS function counts the number of constraints violated by a particular value, given the rest of the current assignment.

初始状态可以随机选择，依次为每个变量选择一个最小冲突值。
给定当前赋值的其余部分，CONFLICTS函数计算特定值违反约束的个数。

Example: Min-conflicts for an 8-Queens Problem 8皇后问题的最小冲突法



A two-step solution using min-conflicts 用最小冲突法的两部解决方案

At each stage, a queen is chosen for reassignment in its column. The number of attacking queens is shown in each square. The algorithm moves the queen to the min-conflicts square, breaking ties randomly.

每一阶段，选择一个皇后在该列上重新分配。每个方格显示皇后攻击的数量。
算法将皇后移到最小冲突的方格内，随机地切断束缚。

Constraint Weighting 约束加权

- Can focus on the important constraints 能聚焦于重要的约束
  - Give each constraint a numeric weight, W_i, initially all 1.
    给定每个约束一个数值权重，W_i，初始值都为1。
  - At each step, choose a variable/value pair to change, that will result in lowest total weight of all violated constraints.
    每一步，选择一个变量/值对加以改变，这将导致所有违反约束的总权重为最低。
  - Then, weights are adjusted by incrementing the weight of each constraint that is violated by current assignment.
    然后，通过增加当前分配所违反的每个约束的权重来调整权重。
- Two benefits 两个益处
  - Add topography to plateau, making sure that it is possible to improve from the current state
    对高原增加了地势，确保能够改善当前的状态。
  - Also add weight to the constraints that are proving difficult to solve.
    对证明难以求解的约束也增加权重。

Thank you for your attention!

PoAI

The Structure of Problems

School of Electronic and Computer Engineering
Peking University

Wang Wenmin

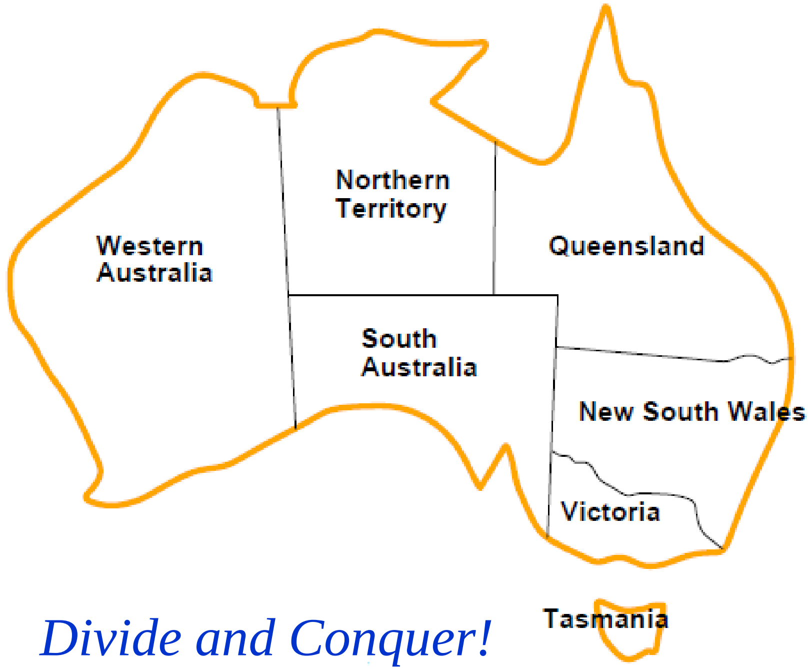
Contents

Decomposing Problem 问题分解

- The structure of problem as represented by constraint graph can be used to find solutions.
  由约束图所表征的问题结构，可以用于寻找解。
- The complexity of solving a CSP is strongly related to the structure of its constraint graph.
  求解一个CSP问题的复杂性，与约束图的结构密切相关。
- The problem in the real world can be decomposed into many sub-problems.
  现实世界的问题可以被分解为许多子问题。

Example:

Coloring Tasmania and coloring the mainland are independent sub-problems.
对塔斯曼尼亚着色与澳洲大陆着色是相互独立的子问题。



Divide and Conquer!

Independent Sub-problems 独立子问题

- They are identifiable as connected components of constraint graph.

  独立子问题可被标识为约束图的联接组件。

- Suppose a graph of n variables can be broken into sub-problems of only c variables: each worst-case solution cost is $O((n/c)\cdot d^c)$, linear in n.

  设n个变量的图可分解为仅有c个变量的子问题：每个最坏解的代价是O((n/c) · dᶜ)， n的线性关系。

- Without the decomposition, the total work is $O(d^n)$.

  如果不分解，则总的运行是O(dn)。

  - E.g., assuming $n = 80$, $d = 2$, $c = 20$, search 10 *million nodes/sec*.

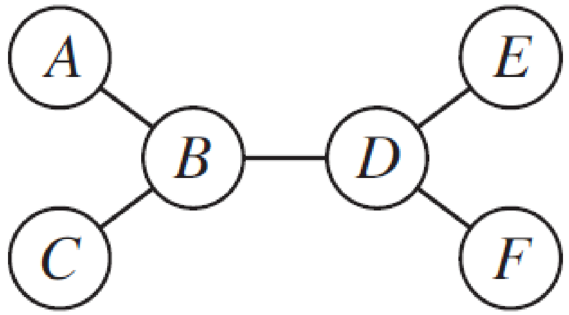
    例如， 假如 n = 80, d = 2, c = 20, 每秒搜索1千万个节点

    - Original problem: $d^n = 2^{80} = 4$ *billion years*;
    - 4 sub-problems : $(n/c)\cdot d^c = (80/20)\cdot 2^{20} = 0.4$ *seconds*.

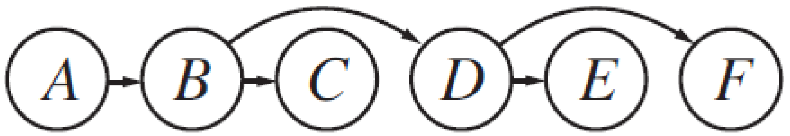
    原始问题：dn = 280 = 40亿年；4个子问题：(n/c) · dc = (80/20) · 220 = 0.4秒。

Tree-structured Problems 树结构问题

- Any *tree-structured CSP* can be solved in *time linear* in the number of variables.
  任何树结构的CSP都可以用变量数中的时间线性加以解决。
- The method to solve a tree-structured CSP: first pick any variable to be root of tree, and choose an ordering (called a topological sort).
  求解树结构CSP的方法：先挑选任意变量作为树的根，然后再选择一个排列（称为拓扑排序）。



The constraint graph of a tree-structured CSP
树结构CSP的约束图



A linear ordering of the variables consistent with the tree with *A* as the root
一种以*A*为根与该树的变量一致的线性排列

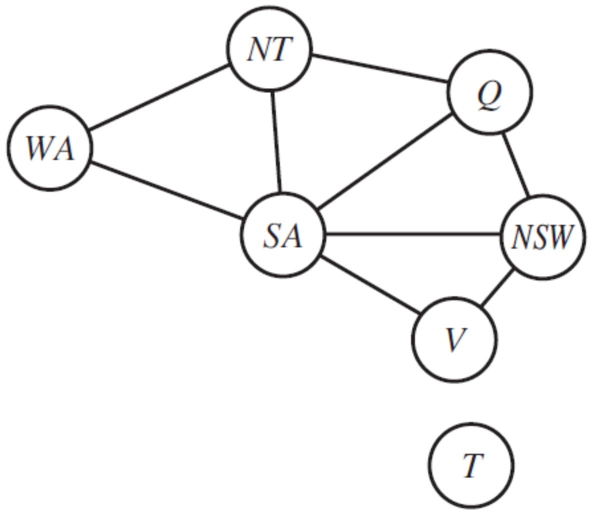
For a tree-structured CSP, it can be solved in $O(n \cdot d^2)$ time.
对于一个树结构的CSP，可以在$O(n \cdot d^2)$时间内得到解。

Algorithm to Solve Tree-structured CSPs 求解树结构CSPs的算法

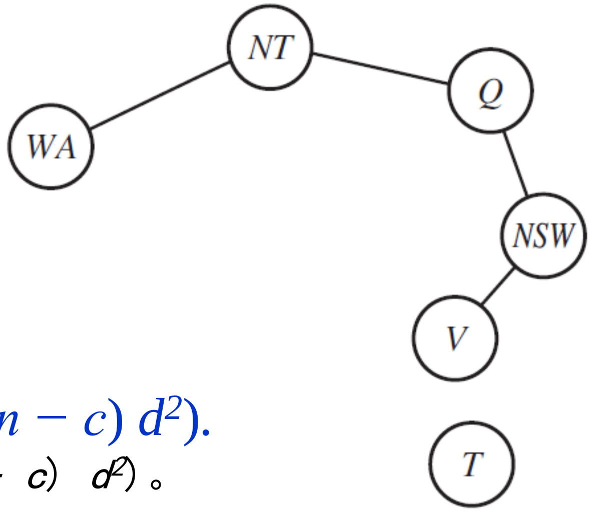
```
function TREE-CSP-SOLVER(csp) returns a solution, or failure
    inputs: csp, a CSP with components $X$, $D$, $C$
    $n \leftarrow$ number of variables in $X$
    assignment ← an empty assignment
    root ← any variable in $X$
    $X \leftarrow$ TOPOLOGICAL-SORT($X$, root)
    for $j = n$ down to 2 do
        MAKE-ARC-CONSISTENT(PARENT($X_j$), $X_j$)
        if it cannot be made consistent then return failure
    for $i = 1$ to $n$ do
        assignment[$X_i$] ← any consistent value from $D_i$
        if there is no consistent value then return failure
    return assignment
```

Reduce Constraint Graphs to Tree Structures 简化约束图为树结构

- 1st approach: cutset conditioning 第1种途径：割集调节
  - It can reduce a general CSP to a tree-structured one, and is quite efficient if a small cutset can be found.
    可以将一个通用CSP问题简化为一个树结构CSP，并且若能够找到一个小割集则相当有效。
- Conditioning: Instantiate a variable, prune its neighbors' domains.
  调节：对一个变量进行实例化，剪去它的相邻范畴。



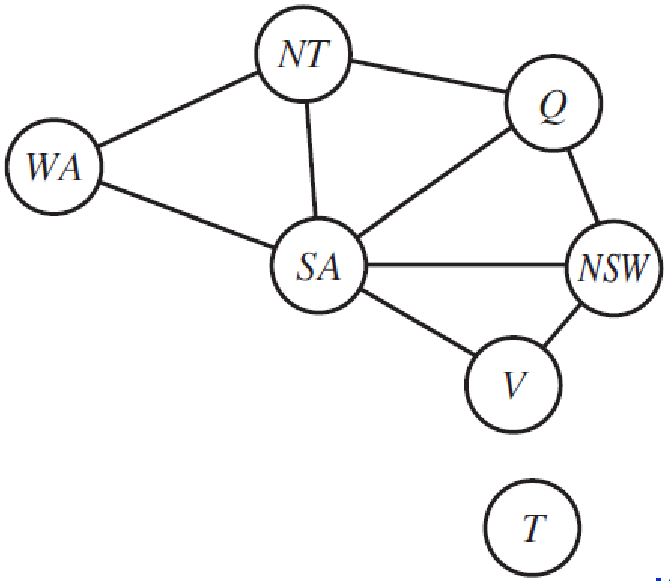
Remove *SA*
移除*SA*

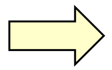


If cutset has size c, then run time is $O(d^c\,(n-c)\,d^2)$.
若割集的尺寸为c，则运行时间是$O(d^c\,(n-c)\,d^2)$。

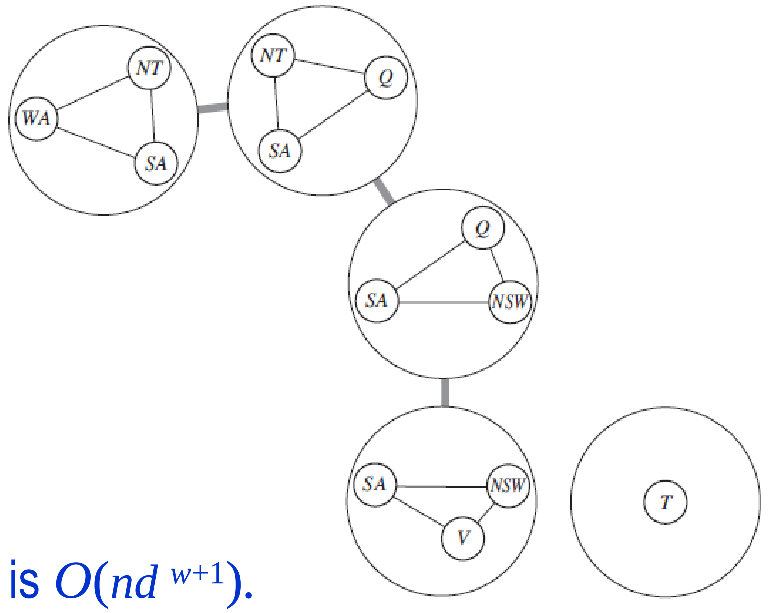
Reduce Constraint Graphs to Tree Structures 简化约束图为树结构

- 2nd approach: tree decomposition 第2途种径：树分解
  - This techniques transform CSP into a tree of sub-problems and are efficient if the tree width of the constraint graph is small.
    这种技法将CSP转换成一棵子问题树，并且当该约束图的树宽较小时很有效。





Decomposition
分解



If a graph has tree width w, then run time is $O(nd^{\,w+1})$.
若一个图的树宽为w，则运行时间是$O(nd^{\,w+1})$。

Thank you for your attention!

PoAI