CSCI 561 - Foundation for Artificial Intelligence

05 -- Constraint Satisfaction

Professor Wei-Min Shen
University of Southern California

Constraint Satisfaction

- Constraint Satisfaction Problems (CSP)
- Backtracking search for CSPs
- Local search for CSPs

一、1. 概念

Constraint satisfaction problems

约束满足问题包含三个要素：变量、变量的值域和关于变量的约束。因此，我们可以形式化的定义如下：

约束满足问题包含 3 个成分 X、D 和 C：

- X是变量的集合{X1, X2, ..., Xn};
- D是值域的集合{D1, D2, ..., Dn}, 每个变量都有自己的值域;
- C是描述变量取值的约束集合.

Standard search problem:

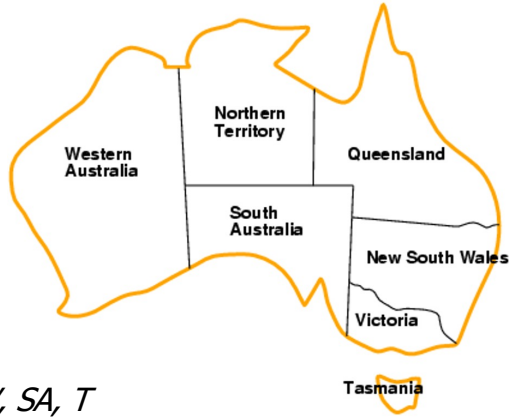
- state is a "black box" – any data structure that supports successor function (actions), heuristic evaluation function, and goal tests

CSP:

- State: is defined by variables X_i with values from domains D_i
- Goal Test: a set of constraints specifying allowable combinations of values for subsets of variables
- Simple example of a formal representation language
- Allows useful general-purpose algorithms with more power than standard search algorithms

求解的主要思想：通过识别违反约束的变量 / 值的组合迅速消除大规模的搜索空间

Example: map coloring problem

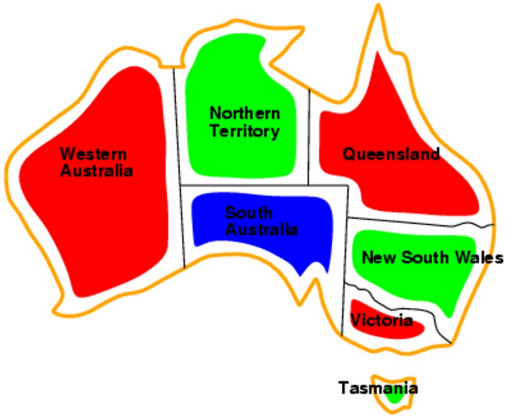


- Variables: *WA, NT, Q, NSW, V, SA, T*
- Domains: D_i = {red, green, blue} (one for each variable)
- Constraints: C_i = <*scope*, *rel*> where *scope* is a tuple of variables and *rel* is the relation over the values of these variables
  - E.g., here, adjacent regions must have different colors
    e.g., WA ≠ NT, or (WA,NT) in {(red,green), (red,blue), (green,red), (green,blue), (blue,red), (blue,green)}

Example: A Map Coloring Problem

- Assignment: values are given to some or all variables
- Consistent (legal) assignment: assigned values do not violate any constraint
- Complete assignment: every variable is assigned a value
- Solution to a CSP: a consistent and complete assignment

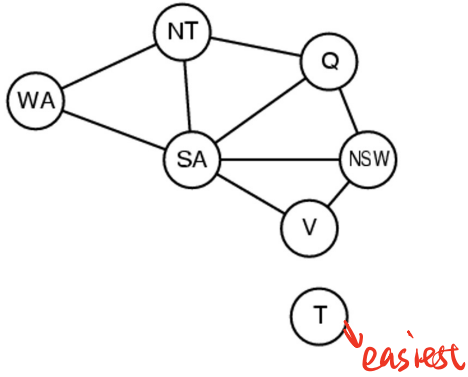
Example: A Map Coloring Problem



- Solutions are complete and consistent assignments,
- e.g., WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green

2、Constraint Graph

- Binary CSP: each constraint relates two variables
- Constraint Graph: nodes are variables, arcs are constraints



Varieties of CSPs

Discrete variables

Finite domains:

- n variables, domain size d $\square$ $O(d^n)$ complete assignments
- e.g., Boolean CSPs, including Boolean Satisfiability (NP-complete)

Infinite domains:

- integers, strings, etc.
- e.g., job scheduling, variables are start/end days for each job
- need a constraint language, e.g., $StartJob_1 + 5 \leq StartJob_3$

Continuous variables

- e.g., start/end times for Hubble Space Telescope observations
- Linear constraints solvable in polynomial time by linear programming

4. Constraint Types and Varieties

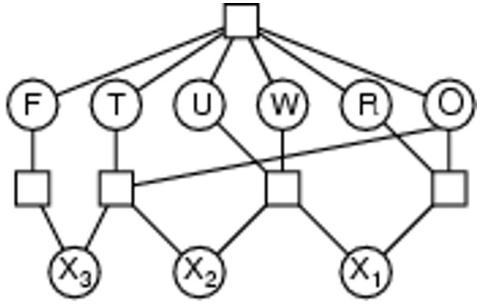
(1) Unary constraints involve a single variable,
- e.g., SA ≠ green

(2) Binary constraints involve pairs of variables,
- e.g., SA ≠ WA

(3) Higher-order (sometimes called global) constraints involve 3 or more variables,
- e.g., cryptarithmetic column constraints

Example: Cryptarithmetic

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



- Variables: $F\ T\ U\ W\ R\ O\ X_1\ X_2\ X_3$
- Domains: $\{0,1,2,3,4,5,6,7,8,9\}$
- Constraints:
  - $Alldiff\ (F,T,U,W,R,O)$
  - $O + O = R + 10 \cdot X_1$
  - $X_1 + W + W = U + 10 \cdot X_2$
  - $X_2 + T + T = O + 10 \cdot X_3$
  - $X_3 = F,\ T \neq 0,\ F \neq 0$

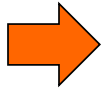
Constraint Hypergraph
Circles: nodes for variable
Squares: hypernodes for n-ary constraints

Real-world CSPs

- Assignment problems
  - e.g., who teaches what class
- Timetabling problems
  - e.g., which class is offered when and where?
- Transportation scheduling
- Factory scheduling
- Notice that many real-world problems involve real-valued variables

Example: Sudoku

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



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

empty

Variables: each square (81 variables)
Domains: [1 .. 9]
Constraints: each column, each row, and each of the nine 3×3 sub-grids that compose the grid contain all of the digits from 1 to 9

b.

Solving CSP Using the "Standard" Search Approach

Let's start with the straightforward approach, then improve it

States are defined by the values assigned so far

- Initial state: the empty assignment { }
- Successor function: assign a value to an unassigned variable that does not conflict with current assignment
  - ☐ fail if no legal assignments
- Goal test: the current assignment is complete

1. This is the same for all CSPs
2. Every solution appears at depth *n* with *n* variables ☐ use depth-first search
3. Path is irrelevant, so can be discarded
4. b = (n - l)d at depth l, hence n! · d^n leaves

Backtracking Search

- Variable assignments are commutative, i.e.,
  [WA = red then NT = green] same as [NT = green then WA = red]

- Only need to consider assignments to a single variable at each node
  - b = d and there are d^n leaves

- Depth-first search for CSPs with single-variable assignments is called backtracking search

- Backtracking search is the basic uninformed algorithm for CSPs

- Can solve *n*-queens for $n \approx 25$

2. pseudocode

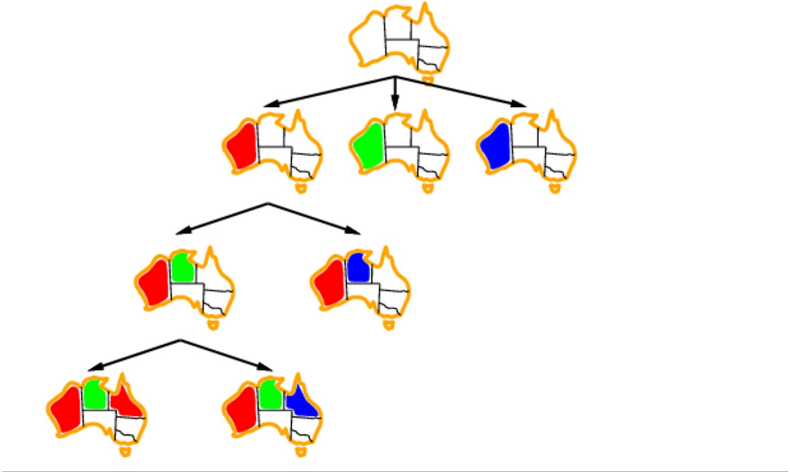
Backtracking search

(note: textbook has a slightly more complex version)

```
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 ← 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 ← RECURSIVE-BACKTRACKING(assignment, csp)
            if result ≠ failue then return result
            remove { var = value } from assignment
    return failure
```

Backtracking example



3. Improving backtracking efficiency

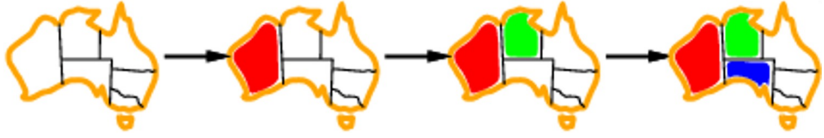
- General-purpose methods can give huge gains in speed (like using heuristics in informed search):
  - Which variable should be assigned next?
  - In what order should its values be tried?
  - Can we detect inevitable failure early?

1) 选择合适的变量

① Selecting the Most Constrained Variable 合法值最少

很快回溯

- Most constrained variable:
  choose the variable with the fewest legal values



- a.k.a. minimum remaining values (MRV) heuristic

如图所示，着色时按照变量与其他变量之间存在的约束数量由多到少的顺序进行着色。

Most constraining variable

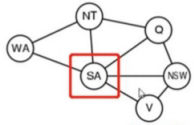
- Tie-breaker among most constrained variables
- Most constraining variable:
  - choose the variable with the most constraints on remaining variables

  - also known as the **degree heuristic**

度启发式：选择涉及对其他未赋值变量的约束数最大的变量,来降低未来选择的分支因子。

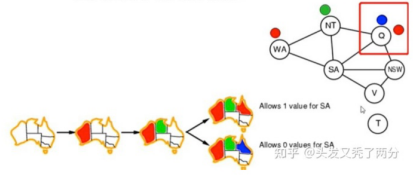
度:变量对其他未赋值变量的约束数

SA的度为5，T的度为0



当进行红色方框圈起来的步骤时，有两种着色选择（蓝色和红色），我们尽量不选择之前着色没有出现过的颜色，将更多是颜色选择机会留给后面的操作，即选红色不选蓝色，因为蓝色之前没有出现过。

最少约束值启发式：优先选择在约束图中排除邻居变量的可选值最少的。这样能给剩下的变量赋值留下最大的灵活性



Least constraining value

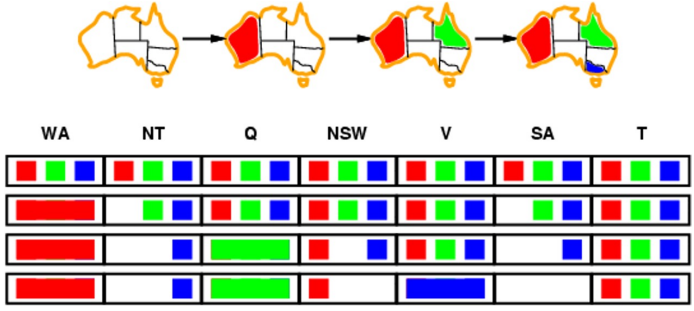
- Given a variable, choose the least constraining value:
  - the one that rules out the fewest values in the remaining variables

- Combining these heuristics makes 1000 queens feasible

(1) 减少搜索空间

① Forward checking

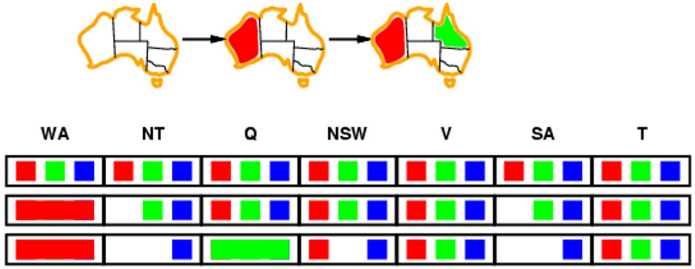
- Idea:
  - Keep track of remaining legal values for unassigned variables (inference step)
  - Terminate search when any variable has no legal values



Constraint propagation

· 约束传播（Constraint propagation）：约束传播的含义是将一个变量的约束内容传播到其他变量。

- Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:



- NT and SA cannot both be blue!

- Constraint propagation repeatedly enforces constraints locally

4. Node and Arc Consistency

① A single variable is node-consistent if all the values in its domain satisfy the variable's unary constraints

② A variable is arc-consistent if every value in its domain satisfies the binary constraints

- i.e., Xi arc-consistent with Xj if for every value in Di there exists a value in Dj that satisfies the binary constraints on arc (Xi, Xj)

③ A network is arc-consistent if every variable is arc-consistent with every other variable.

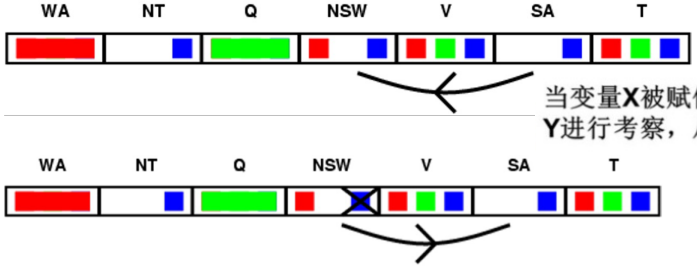
④ Arc-consistency algorithms: reduce domains of some variables to achieve network arc-consistency.

Arc Consistency

- Simplest form of propagation makes each arc consistent
- $X \square Y$ is consistent iff

介绍约束传播之前，要先明白什么是**弧相容（Arc consistency）**：如果对于变量X的每个值x，变量Y都有某个取值能和x保持相容（相容的意思是满足约束），则X->Y的弧是相容的。如图：

for every value x of X there is some allowed y

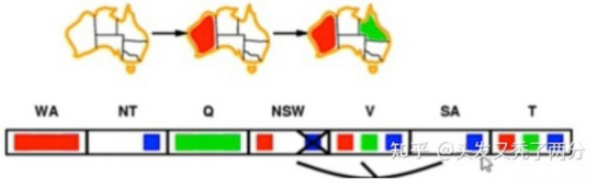


当变量X被赋值，则对每个与X相连的未赋值变量Y进行考察，从Y的值域中删去与X矛盾的值。

由于SA和NSW有约束关系，所以在约束传播的过程中进行弧相容检测，SA的值域只有蓝色，当SA为蓝色时，NSW可以从它的值域中选择红色来应对，所以这个弧是相容的。

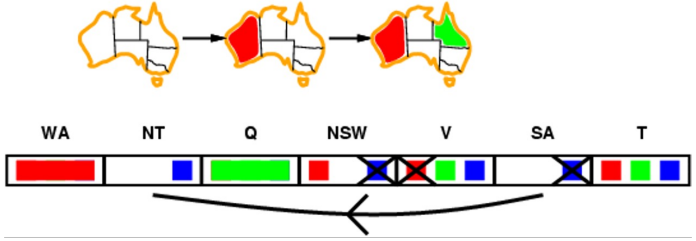
如果反过来，由于NSW的值域中有蓝色，当其选择蓝色时，SA的值域没有相容的选项，所以应该再NSW中删除蓝色。如图：

Arc consistency



- Simplest form of propagation makes ϵ
- $X \square Y$ is consistent iff

for every value x of X there is some allowed y



- If X loses a value, neighbors of X need to be rechecked
- Arc consistency detects failure earlier than forward checking
- After running AC-3, either every arc is arc-consistent or some variable has empty domain, indicating the CSP cannot be solved.
- Can be run as a preprocessor or after each assignment

Arc Consistency Algorithm: AC-3

- Start with a queue that contains all arcs

- Pop one arc (Xi, Xj) and make Xi arc-consistent with respect to Xj
  - If Di was not changed, continue to next arc,
  - Otherwise, Di was revised (domain was reduced), so need to check all arcs connected to Xi again: add all connected arcs (Xk, Xi) to the queue. (this is because the reduction in Di may yield further reductions in Dk)
  - If Di is revised to empty, then the CSP problem has no solution.

Arc Consistency algorithm AC-3

```
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 RM-INCONSISTENT-VALUES(Xi, Xj) then
      for each Xk in NEIGHBORS[Xi] do
        add (Xk, Xi) to queue
```

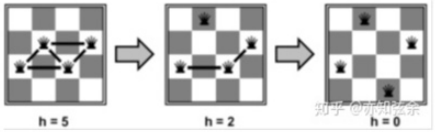
```
function RM-INCONSISTENT-VALUES( Xi, Xj) returns true iff remove a value
  removed ← false
  for each x in DOMAIN[Xi] do
    if no value y in DOMAIN[Xj] allows (x,y) to satisfy constraint(Xi, Xj)
      then delete x from DOMAIN[Xi]; removed ← true
  return removed
```

- Time complexity: $O(n^2d^3)$ (n variables, d values)
- (each arc can be queued only d times, n^2 arcs (at most), checking one arc is $O(d^2)$)

Another Approach: Local Search for CSPs

- Hill-climbing, simulated annealing typically work with "complete" states, i.e., all variables assigned

作为我们感兴趣的最后一个话题，回溯搜索并不是唯一的解决约束满足问题的算法。另一个应用广泛的算法是**本地搜索（local search）**，它的思想非常简单直接，但是相当有效。本地搜索通过迭代改进来工作——从一些随机的变量赋值开始，反复选择违反约束最多的变量并将其重置为违反约束最少的值（这一策略称为**最小冲突启发式（min-conflicts heuristic）**）。通过这一策略，我们能既节省时间又节省空间地解决类似N皇后问题这样的约束满足问题。比如，在下面这个有四个皇后的例子中，我们仅用两步就得到了一个解：



- To apply to CSPs:
  - allow states with unsatisfied constraints
  - operators reassign variable values

- Variable selection: randomly select any conflicted variable

其实，不仅是对有任意大的N的N皇后问题，本地搜索对任意生成的CSP的运行时间几乎是常数并且成功率也非常高！然而，抛开这些优势，本地搜索并不具备完备性和最优性，所以并不一定能得到最优解。另外，有一个关键的比例，在其附近，本地搜索的代价极高。

- Value selection by min-conflicts heuristic:
  - choose value that violates the fewest constraints
  - i.e., hill-climb with *h(n)* = total number of violated constraints

Example: 4-Queens

- States: 4 queens in 4 columns ($4^4 = 256$ states)
- Actions: move queen in column
- Goal test: no attacks
- Evaluation: *h(n)* = number of attacks

- Given random initial state, can solve *n*-queens in almost constant time for arbitrary *n* with high probability (e.g., *n* = 10,000,000)

3. Min-Conflicts Algorithm

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

  $current \leftarrow$ an initial complete assignment for csp
  for i = 1 to max_steps **do**
    if $current$ is a solution for csp **then return** $current$
    $var \leftarrow$ a randomly chosen conflicted variable from csp.VARIABLES
    $value \leftarrow$ the value v for var that minimizes CONFLICTS(var, v, $current$, csp)
    set $var = value$ in $current$
  return *failure*

Figure 6.8 The MIN-CONFLICTS algorithm for solving CSPs by local search. The initial state may be chosen randomly or by a greedy assignment process that chooses a minimal-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.

Example: N-Queens

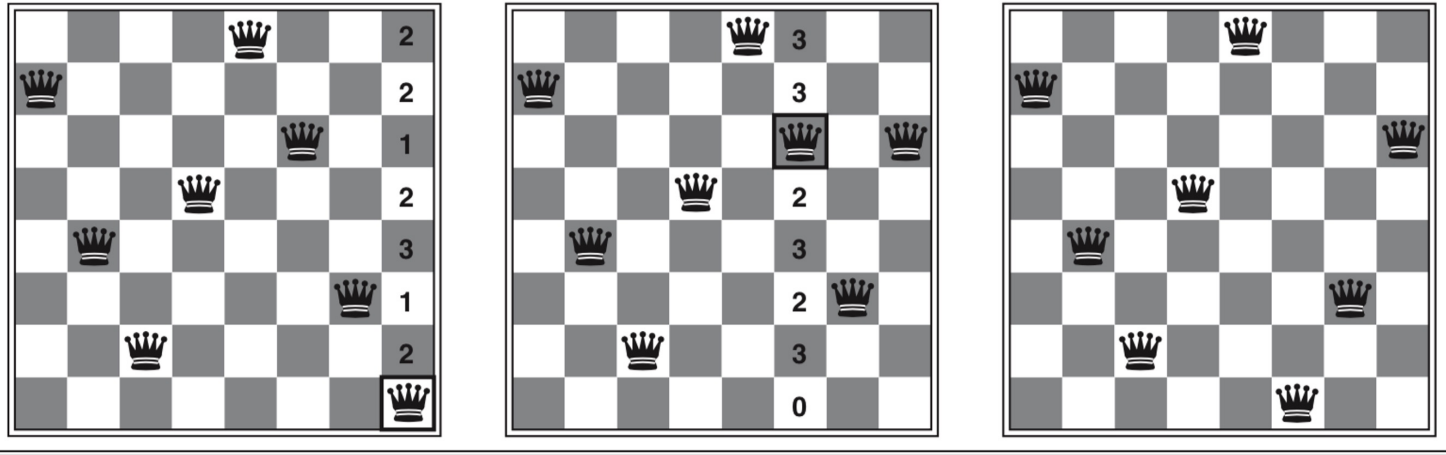


Figure 6.9 A two-step solution using min-conflicts for an 8-queens problem. At each stage, a queen is chosen for reassignment in its column. The number of conflicts (in this case, the number of attacking queens) is shown in each square. The algorithm moves the queen to the min-conflicts square, breaking ties randomly.

Summary

- CSPs are a special kind of search problem:
  - states defined by values of a fixed set of variables
  - goal test defined by constraints on variable values
- Backtracking = depth-first search with one variable assigned per node
- Variable ordering and value selection heuristics help significantly
- Forward checking prevents assignments that guarantee later failure
- Constraint propagation (e.g., arc consistency) does additional work to constrain values and detect inconsistencies
- Iterative min-conflicts is usually effective in practice