Artificial Intelligence

Constraint Satisfaction Problems

Nguyễn Văn Diêu

HO CHI MINH CITY UNIVERSITY OF TRANSPORT

2025

Kiến thức - Kỹ năng - Sáng tạo - Hội nhập

Sứ mệnh - Tầm nhìn

Triết lý Giáo dục - Giá trị cốt lõi

Table of contents

Constraint Satisfaction Problems (CSP)

CSP: 3 components:

- $\mathcal{X} = \left\{X_1, X_2, ..., X_n\right\}$: variables
- $\mathcal{D} = \left\{D_1, D_2, ..., D_n\right\}$: domain; $D_i = \{v_1, v_2, ..., v_k\}$
- $\mathcal{C}$ = set of constraints.

  $\mathcal{C}_j = \langle scope, rel \rangle$
  - *scope*: tuple of variables
  - *rel*: relation of variables

e.g.

$\mathcal{X} = \left\{X_1, X_2\right\}$. $\quad D_1 = D_2 = \{1, 2, 3\}$

constraint: X_1 must be greater than X_2 can be written:

$\left\langle (X_1, X_2), \{(3,1), (3,2), (2,1)\} \right\rangle$ or $\left\langle (X_1, X_2), X_1 > X_2 \right\rangle$

e.g. Map coloring

Australia map coloring problem

e.g. Map coloring

$$\mathcal{X} = \left\{ WA, NT, Q, NSW, V, SA, T \right\}. \qquad D_i = \left\{ red, green, blue \right\}.$$

$$\mathcal{C} = \left\{ SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V, \right.$$
$$\left. WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V \right\}.$$

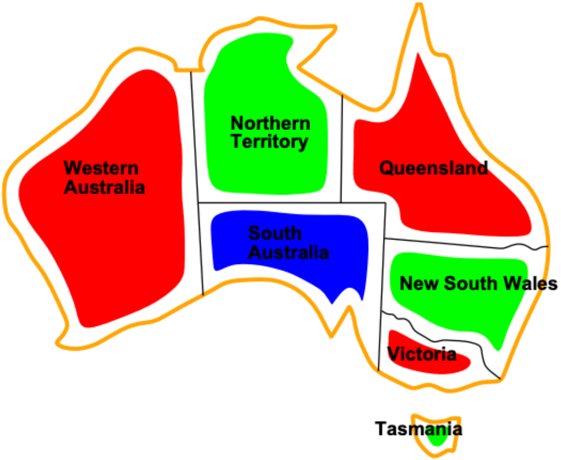
$$SA \neq WA := \left\langle (SA, WA), SA \neq WA \right\rangle.$$

$SA \neq WA$ can be: $\left\{ (red, green), (red, blue), (green, red), \right.$
$\left. (green, blue), (blue, red), (blue, green) \right\}.$

one solutions: $\left\{ WA = red, NT = green, Q = red, NSW = green, V = red, \right.$
$\left. SA = blue, T = green \right\}.$

e.g. Map coloring

One solution for Australia map coloring problem:



e.g. Job-shop scheduling

Scheduling the assembly of a car consisting of 15 tasks :

- install axles (front and back)
- affix all four wheels (right and left, front and back)
- tighten nuts for each wheel
- affix hubcaps
- inspect the final assembly

$$\mathcal{X} = \Big\{ AxleF, AxleB, WheelRF, WheelLF, WheelRB, WheelLB, NutsRF, NutsLF,$$
$$NutsRB, NutsLB, CapRF, CapLF, CapRB, CapLB, Inspect \Big\}$$

task T_1 must occur before task T_2, and task T_1 takes duration d_1 to complete:

$$T_1 + d_1 \leq T_2$$

e.g. Job-shop scheduling

axles in place before wheels, it takes 10 minutes:

$\mathit{AxleF} + 10 \leq \mathit{WheelRF}; \quad \mathit{AxleF} + 10 \leq \mathit{WheelLF};$

$\mathit{AxleB} + 10 \leq \mathit{WheelRB}; \quad \mathit{AxleB} + 10 \leq \mathit{WheelLB}.$

affix the wheel (which takes 1 minute), then tighten the nuts (2 minutes), and finally attach the hubcap:

$\mathit{WheelRF} + 1 \leq \mathit{NutsRF}; \quad \mathit{NutsRF} + 2 \leq \mathit{CapRF};$

$\mathit{WheelLF} + 1 \leq \mathit{NutsLF}; \quad \mathit{NutsLF} + 2 \leq \mathit{CapLF};$

$\mathit{WheelRB} + 1 \leq \mathit{NutsRB}; \quad \mathit{NutsRB} + 2 \leq \mathit{CapRB};$

$\mathit{WheelLB} + 1 \leq \mathit{NutsLB}; \quad \mathit{NutsLB} + 2 \leq \mathit{CapLB}.$

four workers to install wheels, but share one tool that helps put the axle in place:

$$(\mathit{AxleF} + 10 \leq \mathit{AxleB}) \textbf{ or } (\mathit{AxleB} + 10 \leq \mathit{AxleF})$$

e.g. Job-shop scheduling

inspection comes last and takes 3 minutes

For every variable except *Inspect*, add a constraint:

$$X + d_X \leq Inspect.$$

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:

$$D_i = \{0, 1, 2, 3, ..., 30\}.$$

Variations on the CSP

- Variable
  1. **Discrete**
     - finite domains, size $d \Rightarrow O(d^n)$ complete assignments
     - infinite domains (integers, strings, etc.)

       e.g., job scheduling, variables are start/end days for each job
  2. **Continuous domains**, e.g., start/end times for Hubble Telescope observations
- Constraint
  1. **Unary constraint** e.g., $\langle (SA), SA \neq green \rangle$
  2. **Binary constraint** e.g., $\left\langle (SA, WA), SA \neq WA \right\rangle$
  3. **Global constraint** involving an arbitrary number of variables.

     Alldiff: all of the variables involved in the constraint must have different values.

     e.g., Sudoku problem, cryptarithmetic puzzles.

Real-World CSP

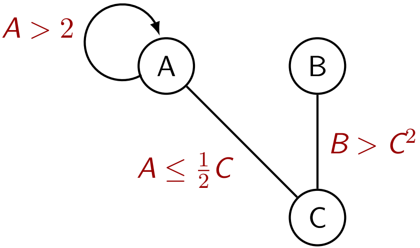
- Assignment problems: e.g., who teaches what class
- Timetabling problems: e.g., which class is offered when and where?
- Hardware configuration
- Transportation scheduling
- Factory scheduling
- Floorplanning
- Fault diagnosis
- ... lots more!
- Many real-world problems involve real-valued variables ...

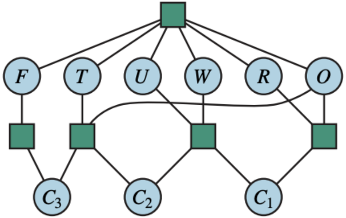
Constraint graph

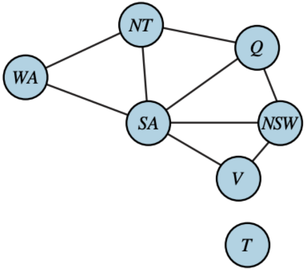
Variables $\rightarrow$ **Vertices**

Constraints $\rightarrow$ **Edges**

- **Unary**: Self-edges
- **Binary**: regular edges
- **n-ary**: hyperedges (hypergraphs)







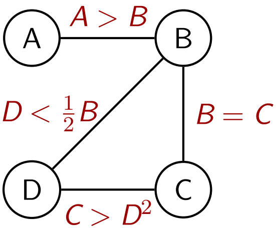
e.g. csp

csp : $\mathcal{X} = A, B, C$

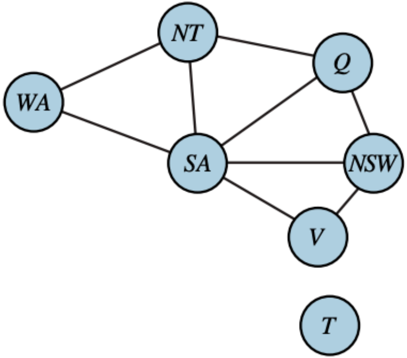
$$\mathcal{D}_{\mathcal{A}} = \{1, 2, 4\} \; ; \; \mathcal{D}_{\mathcal{B}} = \{1, 3, 5\} \; ; \; \mathcal{D}_{\mathcal{C}} = \{3, 5\} \; ; \; \mathcal{D}_{\mathcal{D}} = \{0, 1, 2\}$$

$$\mathcal{C} = \left\{A > B \; ; B = C \; ; D < \frac{1}{2}B \; ; C > D^2\right\}$$

Constraint graph



e.g. Australia Map



e.g. Cryptarithmetic Puzzles

$\mathcal{X} = \{F, T, U, W, R, O\}$

$\mathcal{D} = \{0, 1, 2, ..., 9\}$

global constraint *Alldiff* $\{F, T, U, W, R, O\}$

n-ary constraints

$O + O = R + 10.C_1$

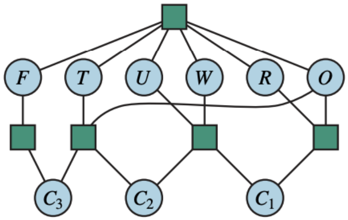
$C_1 + W + W = U + 10.C_2$

$C_2 + T + T = O + 10.C_3$

$C_3 = F$

C_1, C_2, C_3: auxiliary variables.

$$\begin{array}{cccc} & T & W & O \\ + & T & W & O \\ \hline F & O & U & R \end{array}$$



Hypergraph

Ordinary nodes (circles)
Hypernodes (squares)

Constraint Propagation: Inference in CSPs

CSP algorithm:

1. Choosing a new variable assignment.
2. Reduce the number of legal values for a variables.

The key idea is **local consistency**

Node consistency

- A single variable (corresponding to a node in the CSP graph) is node-consistent if all the values in the variable's domain satisfy the variable's unary constraints.
- A graph is node-consistent if every variable in the graph is node-consistent.
- e.g. if South Australians dislike green $\Rightarrow$ SA reduce domain $\{red, blue\}$

Arc consistency

- in binary constraint C; X, Y : variables. (X, Y) is Arc concistency if:
  $\forall x \in Dom(X), \exists y \in Dom(Y) : (x, y) \in C$.
- A graph is arc consistent if every variable is arc consistent with every other variable.
- e.g. Constraint $Y = X^2$ $Dom(X) = \{0, 1, 2, 3\}$, $Dom(Y) = \{0, 1, 4, 9\}$. CSP is arc-consistent.

AC-3 Algorithm

```
function AC-3(csp) returns false if an inconsistency is found and true otherwise
  queue ← a queue of arcs, initially all the arcs in csp

  while queue is not empty do
    (X_i, X_j) ← POP(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 ← 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 ← true
  return revised
```

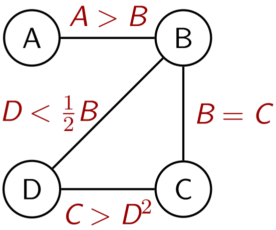
csp : $\mathcal{X} = A, B, C, D$

$$\mathcal{D}_{\mathcal{A}} = \{1, 2, 4\} \; ; \; \mathcal{D}_{\mathcal{B}} = \{1, 3, 5\} \; ; \; \mathcal{D}_{\mathcal{C}} = \{3, 5\} \; ; \; \mathcal{D}_{\mathcal{D}} = \{0, 1, 2\}$$

$$\mathcal{C} = \left\{A > B \; ; B = C \; ; D < \tfrac{1}{2}B \; ; C > D^2\right\}$$

AC-3

1. **Constraint graph**



e.g. AC-3

2. **AC-3**

Revise $\mathcal{D}_X$	(X_i, X_j)	**Queue**	$\cup(X_k, X_i)$
$A\{1,2,4\}$, $B\{1,3,5\}$, $C\{3,5\}, D\{0,1,2\}$		$AB(A > B), BA(B < A), BC(B = C)$, $CB(C = B), CD(C > D^2), DC(D^2 < C)$, $BD(\frac{1}{2}B > D), DB(D < \frac{1}{2}B)$	
$A\{2,4\}$, $B\{1,3,5\}$, $C\{3,5\}, D\{0,1,2\}$	$AB(A > B)$	$BA(B < A), BC(B = C)$, $CB(C = B), CD(C > D^2), DC(D^2 < C)$, $BD(\frac{1}{2}B > D), DB(D < \frac{1}{2}B)$	
$A\{2,4\}$, $B\{1,3\}$, $C\{3,5\}, D\{0,1,2\}$	$BA(B < A)$	$BC(B = C)$, $CB(C = B), CD(C > D^2), DC(D^2 < C)$, $BD(\frac{1}{2}B > D), DB(D < \frac{1}{2}B)$	$CB(C = B)$ $DB(D < \frac{1}{2}B)$

e.g. AC-3

Revise $\mathcal{D}_X$	(X_i, X_j)	Queue	$\cup(X_k, X_i)$
$A\{2,4\}$, $B\{3\}$, $C\{3,5\}$, $D\{0,1,2\}$	$BC(B = C)$	$CB(C = B), CD(C > D^2), DC(D^2 < C)$ $BD(\frac{1}{2}B > D), DB(D < \frac{1}{2}B)$	$AB(A > B)$ $DB(D < \frac{1}{2}B)$
$A\{2,4\}$, $B\{3\}$, $C\{3\}$, $D\{0,1,2\}$	$CB(C = B)$	$CD(C > D^2), DC(D^2 < C), BD(\frac{1}{2}B > D)$, $DB(D < \frac{1}{2}B), AB(A > B)$	$DC(D^2 < C)$
$A\{2,4\}$, $B\{3\}$, $C\{3\}$, $D\{0,1,2\}$	$CD(C > D^2)$	$DC(D^2 < C), BD(\frac{1}{2}B > D)$, $DB(D < \frac{1}{2}B), AB(A > B)$	
$A\{2,4\}$, $B\{3\}$, $C\{3\}$, $D\{0,1\}$	$DC(D^2 < C)$	$BD(\frac{1}{2}B > D)$, $DB(D < \frac{1}{2}B), AB(A > B)$	$BD(\frac{1}{2}B > D)$

e.g. AC-3

Revise $\mathcal{D}_X$	(X_i, X_j)	Queue	$\cup(X_k, X_i)$
$A\{2,4\}$, $B\{3\}$, $C\{3\}$, $D\{0,1\}$	$BD(\frac{1}{2}B > D)$	$DB(D < \frac{1}{2}B), AB(A > B)$	
$A\{2,4\}$, $B\{3\}$, $C\{3\}$, $D\{0\}$	$DB(D < \frac{1}{2}B)$	$AB(A > B)$	$CD(C > D^2)$
$A\{4\}$, $B\{3\}$, $C\{3\}$, $D\{0\}$	$AB(A > B)$	$CD(C > D^2)$	
$A\{4\}$, $B\{3\}$, $C\{3\}$, $D\{0\}$	$CD(C > D^2)$		
$\mathcal{D}_\mathcal{A} = \{4\}$, $\mathcal{D}_\mathcal{B} = \{3\}$, $\mathcal{D}_\mathcal{C} = \{3\}$, $\mathcal{D}_\mathcal{D} = \{0\}$ $\mathcal{C} = \left\{A > B \,;\, B = C \,;\, D < \frac{1}{2}B \,;\, C > D^2\right\}$			

Path consistency

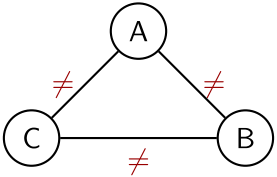
set $\{X_i, X_j\}$ is path-consistent with X_m if:

$\{X_i, X_j\}$ is Arc-consistency and $\{X_i, X_m\}$, $\{X_m, X_j\}$ is also Arc-consistency.

—

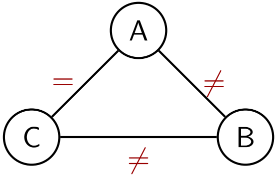
CSP: $\mathcal{X} = \{A, B, C\}$; $\mathcal{D} = \{1, 2\}$;
$\mathcal{C} = \{A \neq B, B \neq C, C \neq A\}$

Arc-consistency but **not Path-consistency**



CSP: $\mathcal{X} = \{A, B, C\}$; $\mathcal{D} = \{1, 2\}$;
$\mathcal{C} = \{A \neq B, B \neq C, C = A\}$

Path-consistency



K-consistency

- 1-consistency: Node-consistency
- 2-consistency: Arc-consistency
- 3-consistency: Path-consistency

strongly k-consistent:

if it is **k-consistent**, **(k—1)-consistent**, **(k—2)-consistent**, ... , **1-consistent**

Global constraints

- Global constraints occur frequently in real problems.

- Can handled by special-purpose algorithms that are more efficient than the general-purpose methods.

e.g. **Alldiff** constraint says that all the variables involved must have distinct values.

Sudoku

- Sudoku board: 81 squares, some of which are initially filled with digits from 1 to 9.
- The puzzle is to fill in all the remaining squares such that no digit appears twice in any row, column, or 3×3 box.
- A row, column, or box is called a unit.

	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		

(a)

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

(b)

Standard search formulation

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.

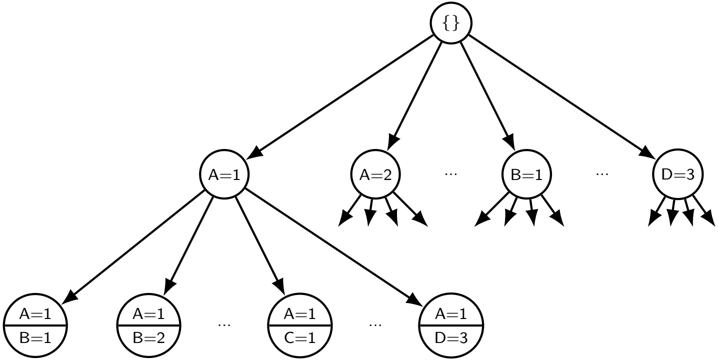
  $\Rightarrow$ fail if no legal assignments (not fixable!)
- **Goal test**: the current assignment is complete

1. This is the same for all CSPs
2. Every solution appears at depth n with n variables

   $\Rightarrow$ use **depth-first search**

e.g. DFS

$\mathcal{X} = \{A, B, C, D\}$; $\mathcal{D} = \{1, 2, 3\}$



Depth 1: 4 variable $\times$ 3

$=$ 12 states.

Depth 2: 3 variable $\times$ 3

$=$ 9 states.

Depth 3: 2 variable $\times$ 3

$=$ 6 states.

Depth 4: 1 variable $\times$ 3

$=$ 3 states (leaf level).

Backtracking Search

- **depth-limited search** could solve **CSP**
- **state**: partial assignment; **action**: extend the assignment
- **Order of assignments list**: no effect on the final outcome
- **CSP are commutative**: Regardless of the assignment order
- Don't care about path!
- Only a single variable at each node in the search tree needs to be considered
- number of leaves in the search tree: d^n

Backtracking Search

func `BACKTRACKING-SEARCH`(csp) **return** a solution or failure

  return `BACKTRACK`($csp, \{\}$)

Backtracking Search

```
func BACKTRACK(csp, assignment) return a solution or failure
    if assignment is complete then
        return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
    foreach value ∈ 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
            remove { var = value } from assignment
    return failure
```

Backtracking Search

- $assignment = \{variable = value, \ldots\}$
- $var \leftarrow$ **SELECT-UNASSIGNED-VARIABLE**$(csp, assignment)$

  minimum-remaining-values (MRV) heuristic: *var* was choosed so that remain with fewest "legal" values. It base on **degree heuristic**, so we choose with **highest degree**.

- $value \in$ **ORDER-DOMAIN-VALUES**$(csp, var, assignment)$

  least-constraining-value heuristic: *value* was choosed so that it make highest choices for the neighboring variables in the constraint

- **INFERENCE**$(csp, var, assignment)$

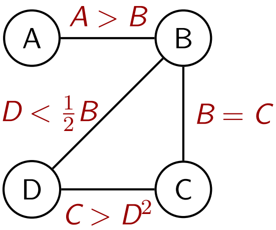
  forward checking: After $X_i \leftarrow value$, the INFERENCE calls AC-3, but instead of a queue of all arcs in the CSP, we start with only the arcs (X_j, X_i) for all X_j that are unassigned variables that are neighbors of X_i

e.g. csp

csp : $\mathcal{X} = A, B, C$

$$\mathcal{D}_{\mathcal{A}} = \{1, 2, 4\} \; ; \; \mathcal{D}_{\mathcal{B}} = \{1, 3, 5\} \; ; \; \mathcal{D}_{\mathcal{C}} = \{3, 5\} \; ; \; \mathcal{D}_{\mathcal{D}} = \{0, 1, 2\}$$

$$\mathcal{C} = \left\{A > B \; ; B = C \; ; D < \tfrac{1}{2}B \; ; C > D^2\right\}$$



e.g. backtrack without hueristic

$\{A = 1\}$

$\{A = 1, B = 1\}$ inconsistent with $A > B$

$\{A = 1, B = 3\}$ inconsistent with $A > B$

$\{A = 1, B = 5\}$ inconsistent with $A > B$

No valid for B, return $result = failure$, Backtrack to $\{A = next\ value\}$

$\{A = 2\}$

$\{A = 2, B = 1\}$

$\{A = 2, B = 1, C = 3\}$ inconsistent with $B = C$

$\{A = 2, B = 1, C = 5\}$ inconsistent with $B = C$

$\{A = 2, B = 1, C = 7\}$ inconsistent with $B = C$

No valid for C, return $result = failure$, Backtrack to $\{A = 2, B = next\ value\}$

e.g. backtrack without hueristic

$\{A = 2, B = 3\}$ inconsistent with $A > B$

$\{A = 2, B = 5\}$ inconsistent with $A > B$

No valid for B, return $result = failure$, Backtrack to $\{A = next\ value\}$

$\{A = 4\}$

$\{A = 4, B = 1\}$

$\{A = 4, B = 1, C = 3\}$ inconsistent with $B = C$

$\{A = 4, B = 1, C = 5\}$ inconsistent with $B = C$

$\{A = 4, B = 1, C = 7\}$ inconsistent with $B = C$

No valid for C, return $result = failure$, Backtrack to $\{A = 4, B = next\ value\}$

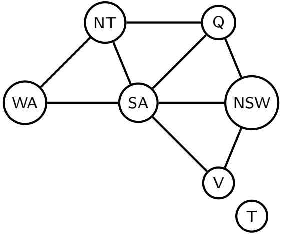
e.g. backtrack without hueristic

$\{A = 4, B = 3\}$

$\{A = 4, B = 3, C = 3\}$

$\{A = 4, B = 3, C = 3, D = 0\}$ *assignment* **Full**

e.g. Australia map



$\{WA = R\}$

$\{WA = R, NT = R\}$ inconsistent with $WA \neq NT$

$\{WA = R, NT = G\}$

$\{WA = R, NT = G, SA = R\}$ inconsistent with $WA \neq SA$

$\{WA = R, NT = G, SA = G\}$ inconsistent with $NT \neq SA$

e.g. backtrack without hueristic

$\{WA = R, NT = G, SA = B\}$

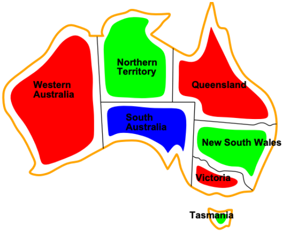
$\{WA = R, NT = G, SA = B, Q = R\}$

$\{WA = R, NT = G, SA = B, Q = R, NSW = R\}$ inconsistent with $Q \neq NSW$

$\{WA = R, NT = G, SA = B, Q = R, NSW = G\}$

$\{WA = R, NT = G, SA = B, Q = R, NSW = G, V = R\}$

$\{WA = R, NT = G, SA = B, Q = R, NSW = G, V = R, T = R|G|B\}$ *assignment* **Full**

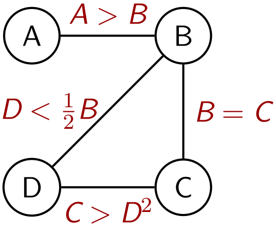


e.g. csp

csp : $\mathcal{X} = A, B, C$

$$\mathcal{D}_{\mathcal{A}} = \{1, 2, 4\} \; ; \; \mathcal{D}_{\mathcal{B}} = \{1, 3, 5\} \; ; \; \mathcal{D}_{\mathcal{C}} = \{3, 5\} \; ; \; \mathcal{D}_{\mathcal{D}} = \{0, 1, 2\}$$

$$\mathcal{C} = \left\{A > B \; ; B = C \; ; D < \tfrac{1}{2}B \; ; C > D^2\right\}$$



e.g. backtrack with hueristic

$\{B = 3\}$ **minimum-remaining-values (MRV), least-constraining-value**

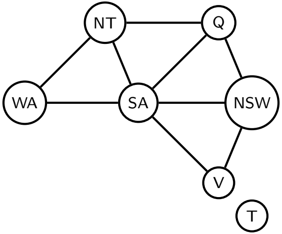
$\Rightarrow dom(A) = \{4\}$, $dom(B) = \{3\}$, $dom(D) = \{0, 1\}$

$\{B = 3, A = 4\}$

$\{B = 3, A = 4, C = 3\}$

$\{B = 3, A = 4, C = 3, D = 0\}$ *assignment* **Full**

e.g. Australia map



$\{SA = R\}$. heuristic: **minimum-remaining-values (MRV), least-constraining-value**

$\Rightarrow SA\{R\}$ $WA\{GB\}$, $NT\{GB\}$, $Q\{GB\}$, $NSW\{GB\}$, $V\{GB\}$, $T\{RGB\}$

$\{SA = R, WA = R\}$ inconsistent with $SA \neq WA$

e.g. backtrack with hueristic

$\{SA = R, WA = G\}$, heuristic: **(MRV), least-constraining-value**

$\Rightarrow SA\{R\}\ WA\{G\}$, $NT\{B\}$, $Q\{G\}$, $NSW\{B\}$, $V\{G\}$, $T\{RGB\}$

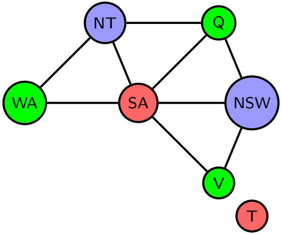
$\{SA = R, WA = G, NT = B\}$, heuristic: **(MRV), least-constraining-value**

$\{SA = R, WA = G, NT = B, Q = G\}$, heuristic: ...

$\{SA = R, WA = G, NT = B, Q = G, NSW = B\}$, heuristic: ...

$\{SA = R, WA = G, NT = B, Q = G, NSW = B, V = G\}$, heuristic: ...

$\{SA = R, WA = G, NT = B, Q = G, NSW = B, V = G, T = R|G|B\}$ *assignment* **Full**



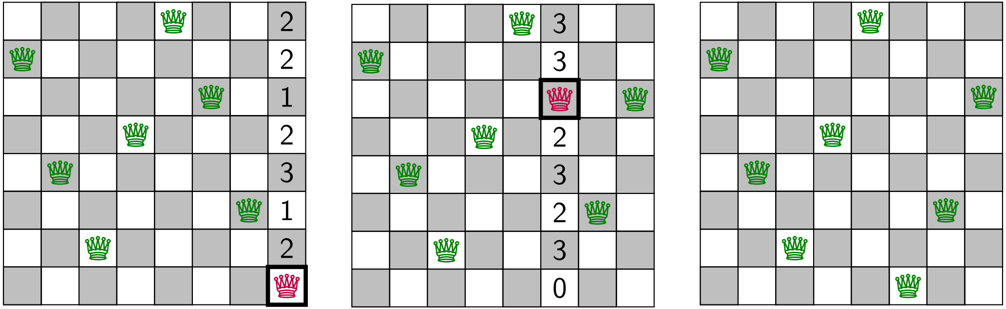
Local Search for CSP

- all variables assigned
- To apply to CSP
  - Allow states with unsatisfied constraints
  - operators **reassign** variable values
- Select a variable: random conflicted variable
- Select a value: **min-conflicts heuristic**
  - Value that violates the fewest constraints
  - Hill-climbing like algorithm with the objective function being the number of violated constraints
- Works surprisingly well in problem like n-Queens

Min-Conflicts

```
func MIN-CONFLICTS(csp, max_steps) return a solution or failure
    input: 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(csp, var, v, current)
        set var = value ∈ current
    return failure
```

e.g. 8-Queens



MIN-CONFLICTS local search algorithm for CSP.

initial state: chosen randomly or by a greedy assignment minimal-conflict value.

each variable in turn: **CONFLICTS** function counts the number of constraints violated.