CONSTRAINT SATISFACTION PROBLEMS

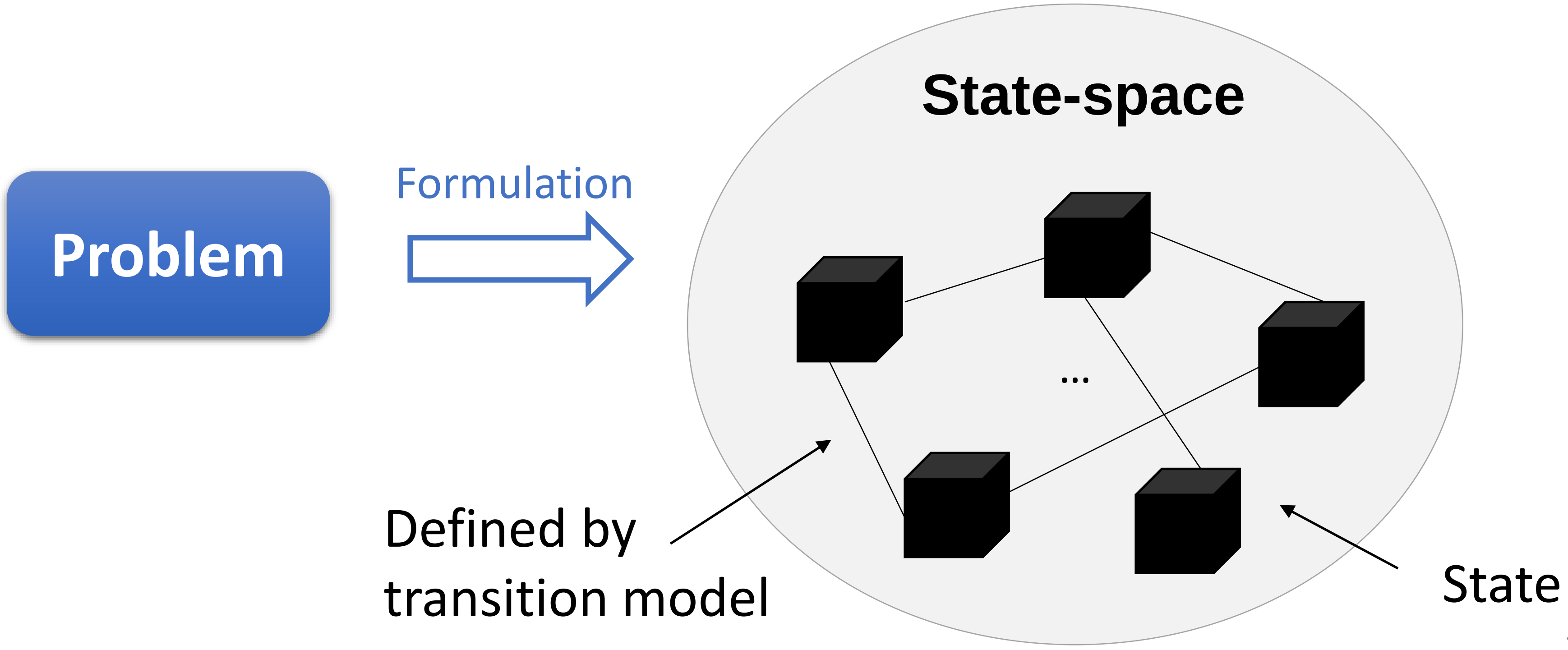
Nguyễn Ngọc Thảo – Nguyễn Hải Minh
{nnthao, nhminh}@fit.hcmus.edu.vn

Outline

- Constraint satisfaction problems (CSPs)
- Constraint propagation: Inference in CSPs
- Backtracking search for CSPs
- Local search for CSPs (Self-study)
- The structure of problems (Self-study)

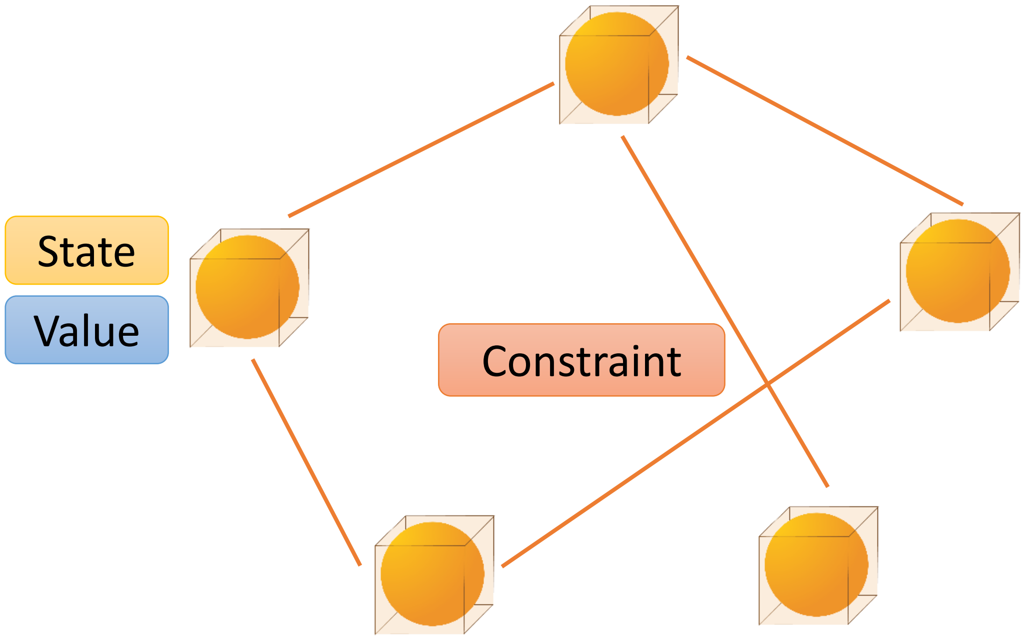
State-space search problems

- Each state is atomic, or indivisible, from the point of view of the search algorithm.
- Domain-specific code describing transitions between states is required for each problem.



Factored representation

- CSP factorizes each state into a set of variables, each of which has a value.
- A problem is solved when every **variable** has a **value** that satisfies all the **constraints** on that variable.



Constraint satisfaction problem

Problem formulation

- A CSP formulation has three main components.

$\boldsymbol{X} = \{X_1, .., X_n\}$: a set of variables

$\boldsymbol{D} = \{D_1, .., D_n\}$: a set of domains, one for each variable.

  - $D_i = \{v_1, .., v_k\}$: set of allowable values for variable X_i

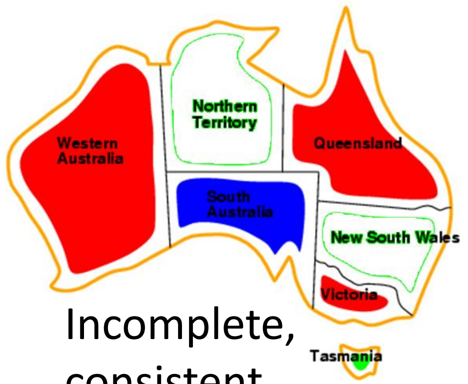
$\boldsymbol{C}$: a set of constraints, $C_j = \langle scope, rel \rangle$, that state allowable combinations of values.

  - $scope$: a tuple of variables that participate in the constraint
  - A relation rel defines the values that participated variables can take

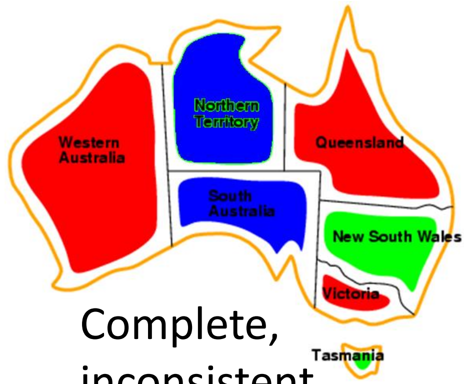
- A CSP is an NP-complete problem in general.

Assignments in CSP

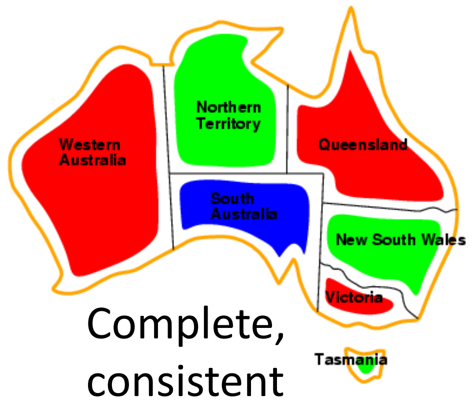
- CSPs assigns values to variables, $\{X_i = v_i, X_j = v_j, \ldots\}$.
- A solution is a consistent and complete assignment.
    - A consistent assignment does not violate any constraints.
    - A complete assignment has every variable assigned a value.
- A partial solution is a partial assignment that is consistent.
    - A partial assignment is one that leaves some variables unassigned.



Incomplete, consistent assignment



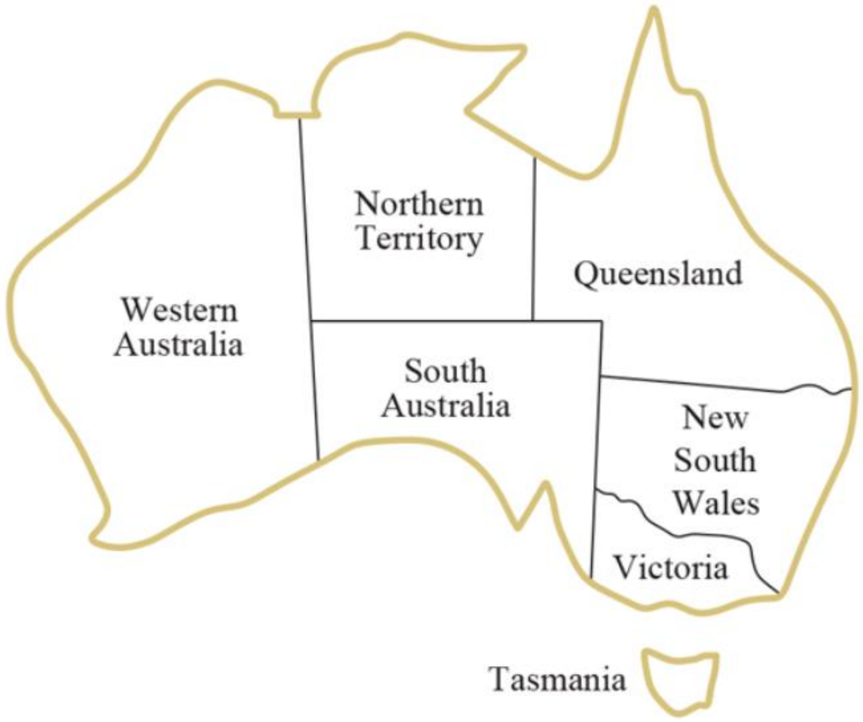
Complete, inconsistent assignment

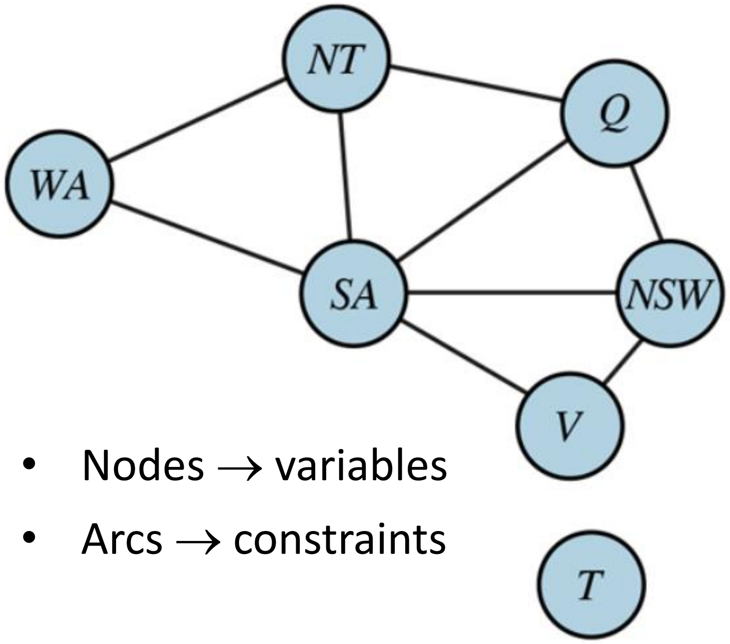


Complete, consistent assignment

Example problem: Map coloring

- **Objective:** Color each region either **red**, **green**, or **blue** in such a way that no neighboring regions have the same color





- Nodes $\rightarrow$ variables
- Arcs $\rightarrow$ constraints

(a) The principal states and territories of Australia.
(b) The map-coloring problem represented as a constraint graph.

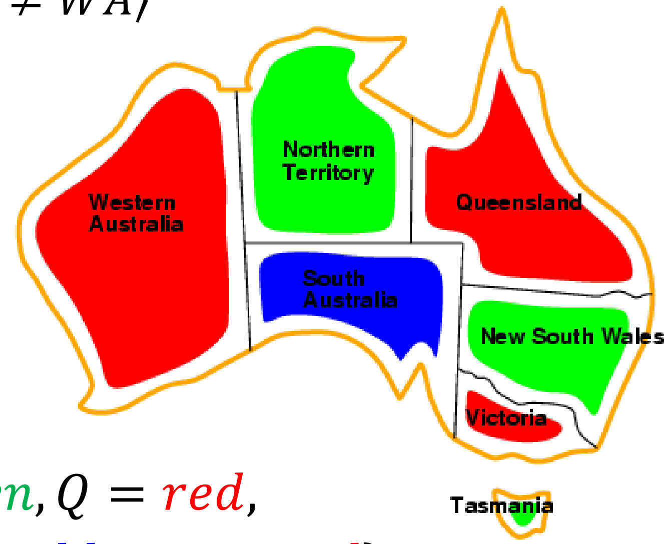
Example problem: Map coloring

- Variables: $X = \{WA, NT, Q, NSW, V, SA, T\}$
- Domains: $D_i = \{red, green, blue\}$
- Constraints: Neighboring regions have distinct colors.

$$C = \begin{Bmatrix} SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V, \\ WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V \end{Bmatrix}$$

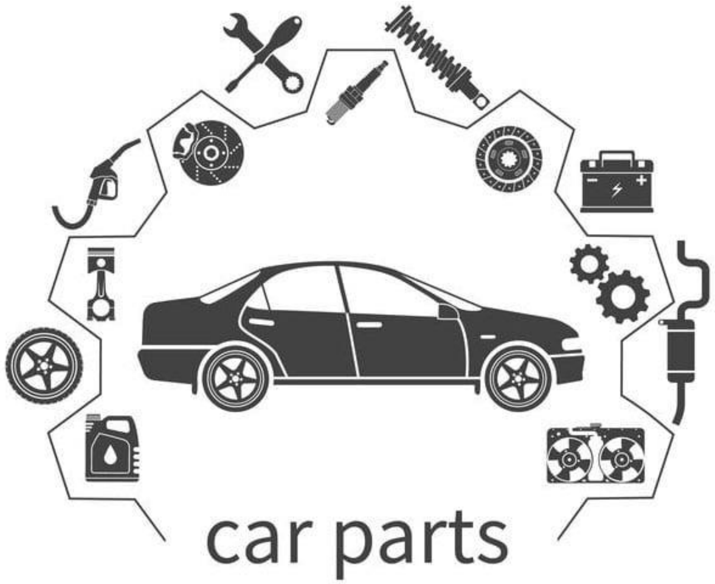
  - $SA \neq WA$ is a shortcut of $\langle (SA, WA), SA \neq WA \rangle$

- There are many possible solutions.



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

Example problem: Job-shop scheduling



15 tasks

- Install axles (front and back)
- Affix all four wheels (right and left, front and back)
- Tighten nuts for each wheel
- Affix hubcaps, and
- Inspect the final assembly

- Some tasks must occur before another, and some tasks can go on at once
  - E.g., a wheel must be installed before the hubcap is put on
- A task takes a certain amount of time to complete.

Example problem: Job-shop scheduling

- Variables: $X = \{Axle_F, Axle_B, Wheel_{RF}, Wheel_{LF}, Wheel_{RB}, Wheel_{LB},$
  $Nuts_{RF}, Nuts_{LF}, Nuts_{RB}, Nuts_{LB},$
  $Cap_{RF}, Cap_{LF}, Cap_{RB}, Cap_{LB}, Inpsect\}$
- Domains: The time that the task starts
- Assume that the tasks, T_1 and T_2, take duration d_1 and d_2 to complete, respectively
- Precedence constraints: The task T_1 must occur before the task T_2, i.e., $\boldsymbol{T_1 + d_1 \leq T_2}$
- Disjunctive constraints: The tasks T_1 and T_2 must not overlap in time, i.e., $\boldsymbol{T_1 + d_1 \leq T_2}$ or $\boldsymbol{T_2 + d_2 \leq T_1}$

Example problem: Job-shop scheduling

- The axles must be in place before the wheels are put on. Installing an axle takes 10 minutes.

$$Axle_F + 10 \leq Wheel_{RF} \qquad Axle_F + 10 \leq Wheel_{LF}$$
$$Axle_B + 10 \leq Wheel_{RB} \qquad Axle_B + 10 \leq Wheel_{LB}$$

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

$$Wheel_{RF} + 1 \leq Nut_{RF} \qquad Nuts_{RF} + 2 \leq Cap_{RF}$$
$$Wheel_{LF} + 1 \leq Nut_{LF} \qquad Wheel_{RB} + 1 \leq Nut_{RB} \qquad Nuts_{LF} + 2 \leq Cap_{LF} \qquad Nuts_{RB} + 2 \leq Cap_{RB}$$
$$Wheel_{LB} + 1 \leq Nut_{LB} \qquad Nuts_{LB} + 2 \leq Cap_{LB}$$

- Suppose we have four workers to install wheels, but they must share one tool that helps put the axle in place. $Axle_F + 10 \leq Axle_B \;\; or \;\; Axle_B + 10 \leq Axle_F$

- The inspection comes last and takes 3 minutes $\rightarrow$ for every variable except $Inspect$, add a constraint of the form $X + d_X \leq Inspect$.
- Finally, suppose there is a requirement to get the whole assembly done in 30 minutes $\rightarrow$ limit the domain of all variables to $D_i = \{1, 2, 3, \dots, 27\}$.

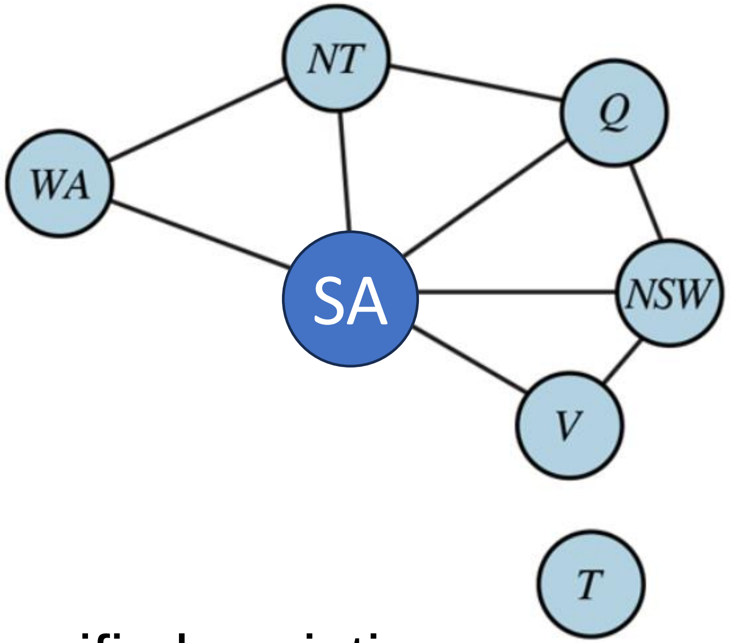
Why formulate a problem as a CSP?

- CSP gives better insights to the problems and their solutions.
- A CSP solver can quickly prune large swathes of the search space that an atomic state-space searcher cannot.

Assume that we have chosen {SA = blue}.

Search: 3^5 = 243 assignments

CSP: 2^5 = 32 assignments ↓ 87%



- It is more general than problem-specific heuristics.

Types of variables in CSP

- The simplest CSP involves variables that have discrete and finite domains.
  - E.g., map coloring problem, job-shop scheduling with time limits
- A discrete domain can be infinite.
  - E.g., set of integers problem, job-shop scheduling without time limits
- CSPs with continuous domains are common in practice
  - E.g., the scheduling of experiments on the Hubble Space Telescope uses real-values variables for observation timings.
  - They are widely studied in the field of operations research, e.g., linear programming.

Types of constraints in CSP

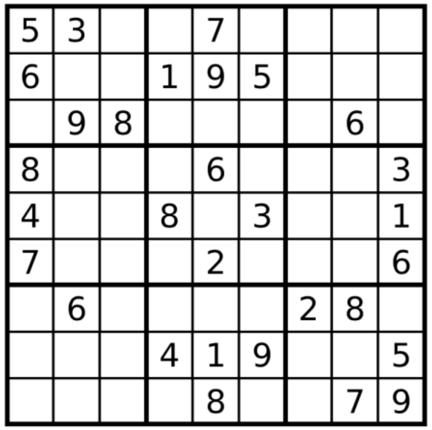
- A unary constraint restricts the value of a single variable.
  - E.g., the South Australians hates green $\rightarrow \langle (SA), SA \neq green \rangle$
- A binary constraint relates two variables
  - E.g., adjacent regions are of different colors, $\langle (SA, WA), SA \neq WA \rangle$
- Any n-ary $(n > 2)$ constraint can be turned into a binary one.

- A binary CSP is one with only unary and binary constraints, which can be represented as a constraint graph.

Types of constraints in CSP

- A higher-order constraint involves three or more variables.
  - E.g., the ternary constraint $Between(X, Y, Z)$ can be defined as $\langle (X, Y, Z), X < Y < Z \text{ or } X > Y > Z \rangle$
- A global constraint relates an arbitrary number of variables.

$Alldiff$: all variables involved must have different values

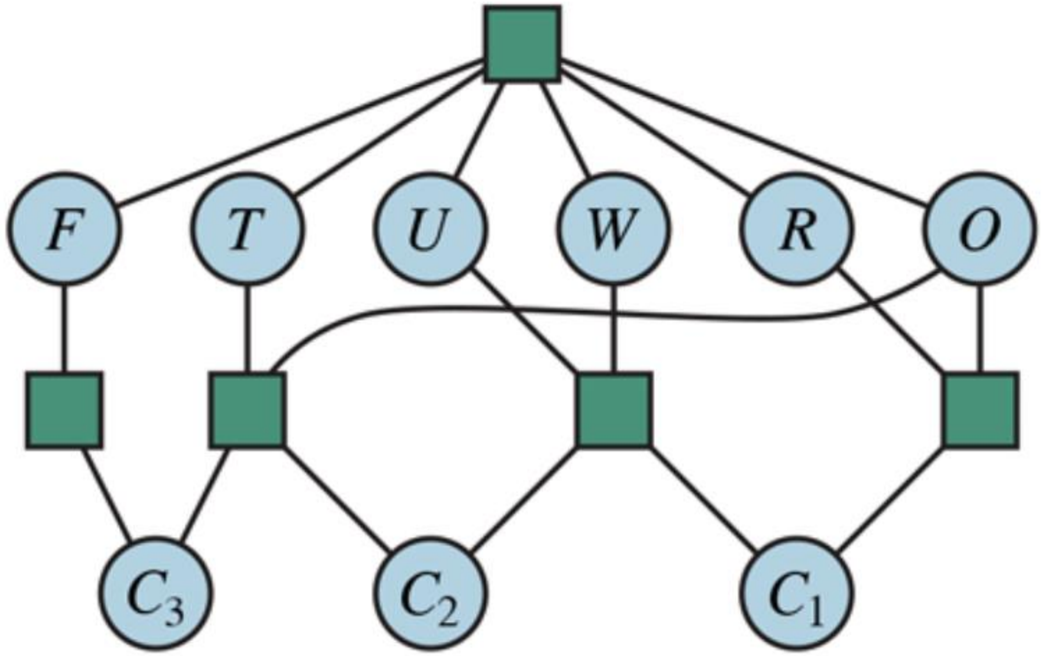


5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

$$\begin{array}{r} E\ G\ G \\ +\ E\ G\ G \\ \hline =P\ A\ G\ E \end{array}$$

Example problem: Cryptarithmetic

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



(Left) A cryptarithmetic problem assumes that each letter stands for a distinct digit and find a substitution of digits for letters such that the resulting sum is arithmetically correct. (Right) The constraint hypergraph for the cryptarithmetic problem, showing the $Alldiff$ constraint (square box at the top) and the column addition constraints (four square boxes in the middle). The variables C_1, C_2, and C_3 represent the carry digits for the three columns from right to left.

Example problem: Cryptarithmetic

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

- Variables: $F\ T\ U\ W\ R\ O\ C_1\ C_2\ C_3$
- Domains: $\{0,1,2,3,4,5,6,7,8,9\}$
- Constraints:
    - $Alldiff(F, T, U, W, R, O)$
    - $O + O = R + 10 \cdot C_1$
    - $C_1 + W + W = U + 10 \cdot C_2$
    - $C_2 + T + T = O + 10 \cdot C_3$
    - $C_3 = F$
    - $T \neq 0, F \neq 0$

Column addition constraints

$\longrightarrow$ No leading zeroes are allowed.

Preference constraints

- The constraints described so far have all been absolute constraints, violation of which rules out a potential solution.
- Many real-world CSPs involve preference constraints telling which solutions are preferred.
  - E.g., Prof. R prefers teaching in the morning. A schedule that has Prof. R’s class at 2 p.m. would still be an allowable but not optimal.
- These constraints are often encoded as costs on variable assignments $\rightarrow$ Constrained optimization problem (COP)
  - E.g., an afternoon slot for Prof. R costs 2 points, whereas a morning slot costs only 1 points.

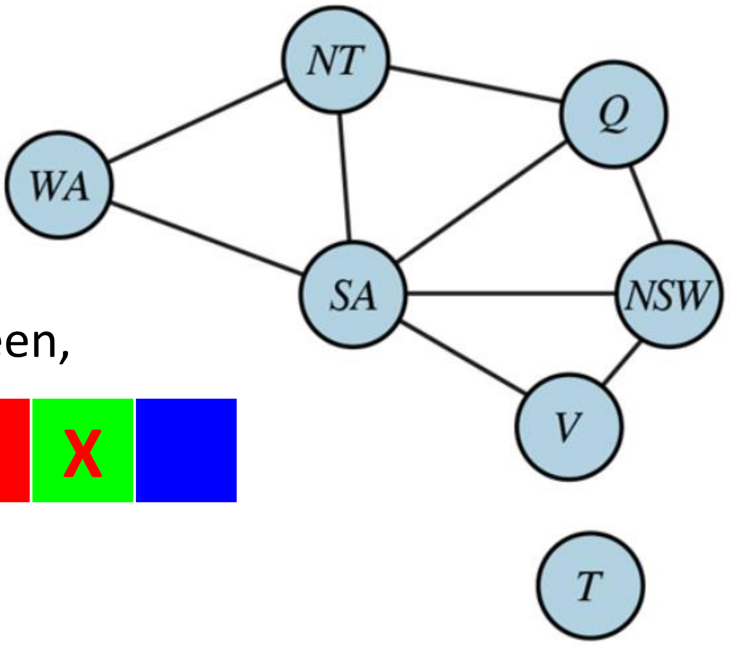
Constraint Propagation: Inference in CSPs

Constraint propagation

- Reduce the number of legal values for a variable by using constraints $\rightarrow$ lower the legal values for other variables, etc.
  - This will leave fewer choices to consider when we make the next choice of a variable assignment.
- It may be intertwined with search or done as a preprocessing step, i.e., before search starts.
  - The preprocessing can sometimes solve the whole problem without any further search.

Node consistency

- A single variable is node-consistent if all the values in the variable’s domain satisfy the variable’s unary constraints.



The South Australians dislike green,

The domain of $\{SA\}$ will be 🟥 ~~🟩~~ 🟦

Arc consistency (AC)

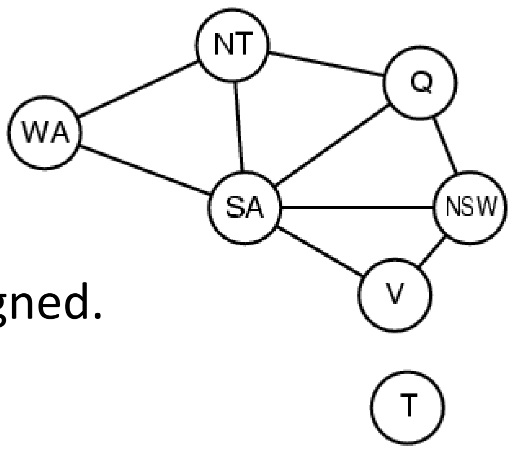
- Given two variables, X_i and X_j, whose domains are D_i and D_j, respectively.
- X_i is arc-consistent with X_j if for every value in D_i there is some value in D_j that satisfies the binary constraint (X_i, X_j).
  - E.g., $\langle (X, Y), Y = X^2 \rangle$, both domains are sets of digits $\rightarrow$ reduce X's domain to {0, 1, 2, 3} and Y's to {0, 1, 4, 9}
- Arc consistency may have no effect in several cases.
  - E.g., in the constraint $SA \neq WA$, no matter what value chosen for SA, there is a valid value for WA, following $SA \neq WA$

{(red,green), (red,blue), (green,red), (green,blue), (blue,red), (blue,green)}

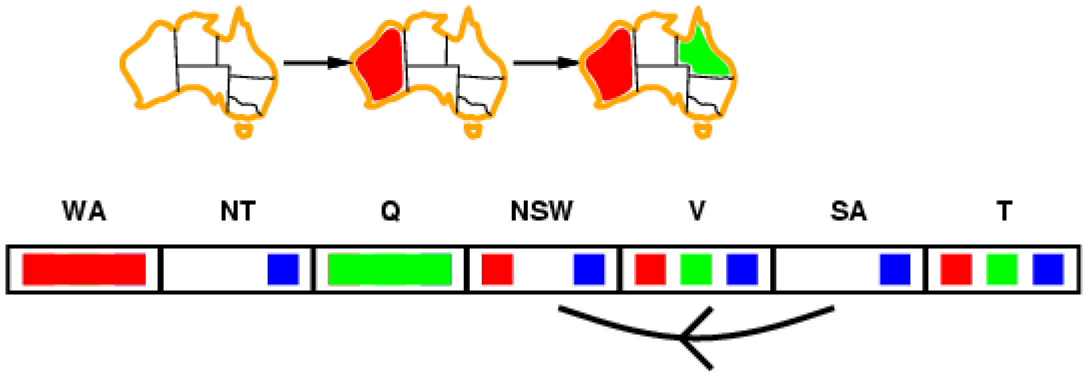
```
function AC-3(csp) returns false if an inconsistency is found and true otherwise
  queue ← a queue of arcs, initially all the arcs in csp
  while queue is not empty do
    $(X_i, X_j)$ ← POP(queue)
    if REVISE(csp, $X_i$, $X_j$) then
      if size of $D_i = 0$ then return false
      for each $X_k$ in $X_i$.NEIGHBORS - {$X_j$} do
        add ($X_k$, $X_i$) to queue
  return true
```

- The worst-case complexity is $O(cd^3)$
- c: number of binary constraints (arc)
- Each variable has domain size d

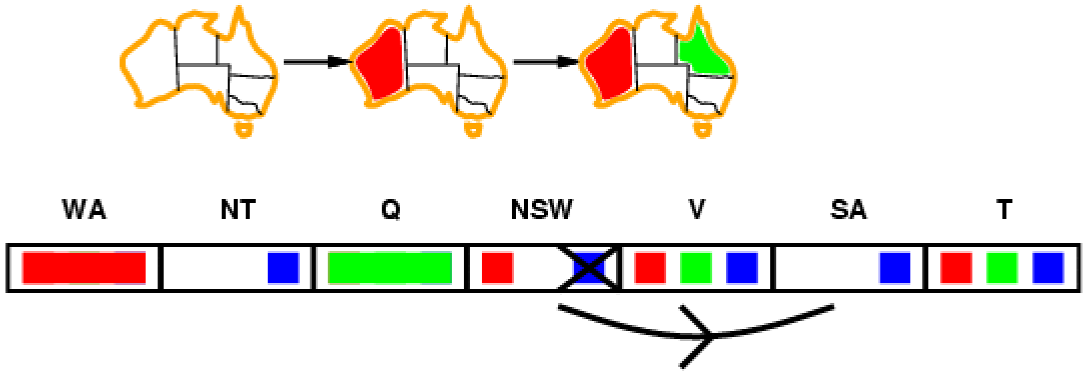
```
function REVISE(csp, $X_i$, $X_j$) returns true iff we revise the domain of $X_i$
  revised ← false
  for each $x$ in $D_i$ do
    if no value $y$ in $D_j$ allows $(x, y)$ to satisfy the constraint between $X_i$ and $X_j$
      delete $x$ from $D_i$
      revised ← true
  return revised
```



Consider state of search after WA and Q are already assigned.

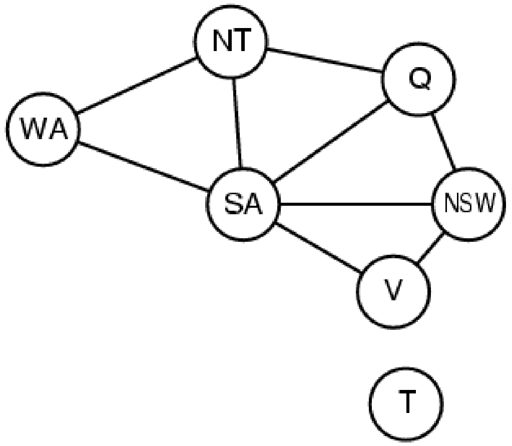


- $SA \rightarrow NSW$ is consistent if $SA = blue$ and $NSW = red$

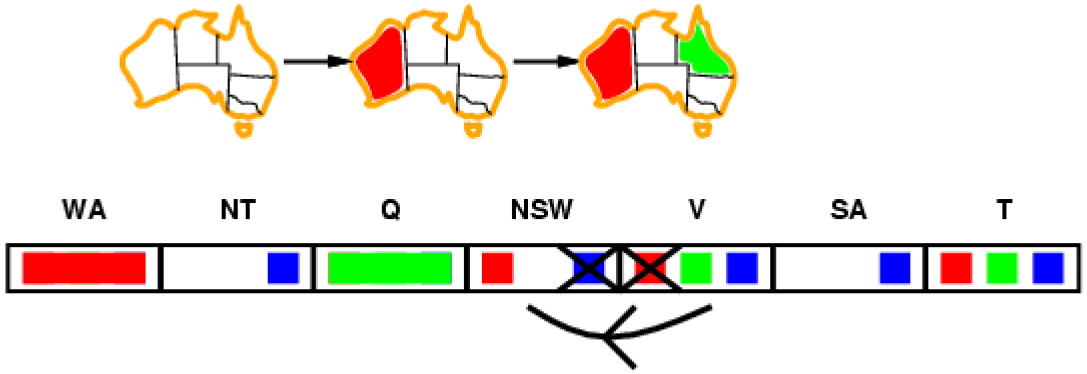


- $NSW \rightarrow SA$ is consistent if $NSW = red$ and $SA = blue$
  $NSW = blue \text{ and } SA = ???$

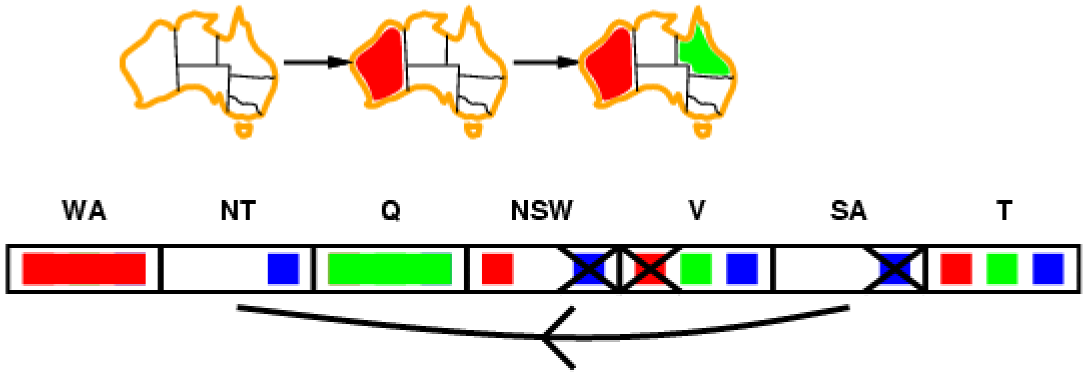
Arc-consistency can be made by removing $blue$ from NSW



If X loses a value, neighbors of X need to be rechecked



- Check $V \rightarrow NSW$
- Not consistent for $V = red$
  $\rightarrow$ remove red from V



- Check $SA \rightarrow NT$
- Not consistent for $SA = blue$
  $\rightarrow$ remove $blue$ from SA
- The domain of SA is empty.

Arc consistency detects failure earlier than forward checking.

Arc consistency: An evaluation

- Arc consistency may run before the search starts or after each assignment, repeatedly until no inconsistency remains.
    - Trade-off: Eliminate large inconsistent parts of the state space while requiring some overhead to do
- AC performs a systematic method for arc-checking.
    - Incoming arcs can become inconsistent, while outgoing arcs stay still.

Quiz 01: Timetable scheduling

- You are scheduling for computer science classes that meet on Mondays, Wednesdays and Fridays .
- There are 5 classes and 3 professors who will be teaching these classes.
- You are constrained that each professor can only teach one class at a time.
- The classes are:
    - Class 1 - Intro to Programming: meets from 8:00-9:00am
    - Class 2 - Intro to Artificial Intelligence: meets from 8:30-9:30am
    - Class 3 - Natural Language Processing: meets from 9:00-10:00am
    - Class 4 - Computer Vision: meets from 9:00-10:00am
    - Class 5 - Machine Learning: meets from 9:30-10:30am
- The professors are:
    - Professor A, who is available to teach Classes 3 and 4.
    - Professor B, who is available to teach Classes 2, 3, 4, and 5.
    - Professor C, who is available to teach Classes 1, 2, 3, 4, and 5.

Quiz 01: Timetable scheduling

- Formulate this problem as a CSP in which there is one variable per class, stating the domains (i.e., available professors) and constraints.
    - Constraints should be specified formally and precisely but may be implicit rather than explicit.
- Draw the constraint graph associated with your CSP.
- Show the domains of the variables after running arc-consistency on this initial graph (after having already enforced any unary constraints).
- Give one solution to this CSP.

Quiz 02: Magic square of order 3

- A magic square of order n is an arrangement of n^2 distinct positive integers, in a square, such that the n numbers in all rows, all columns, and both diagonals sum to the same constant.
- For order-3 magic squares, there is a general formula devised by Édouard Lucas.

The aside table is made of positive integers, **p**, **q**, and **z**.

These nine numbers will form a magic square with the magic constant **3(p+q)** so long as **0 < q < z < p** and **z ≠ 2q**.

$p+q-z$	$p+2q+z$	p
$p+z$	$p+q$	$p+2q-z$
$p+2q$	$p-z$	$p+q+z$

- Formulate the above magic square construction method as a CSP over the variables p, q and z, each of which has its initial domain in {1,...,9}.
- Let the magic constant be 15. Solve the CSP.

Backtracking Search for CSPs

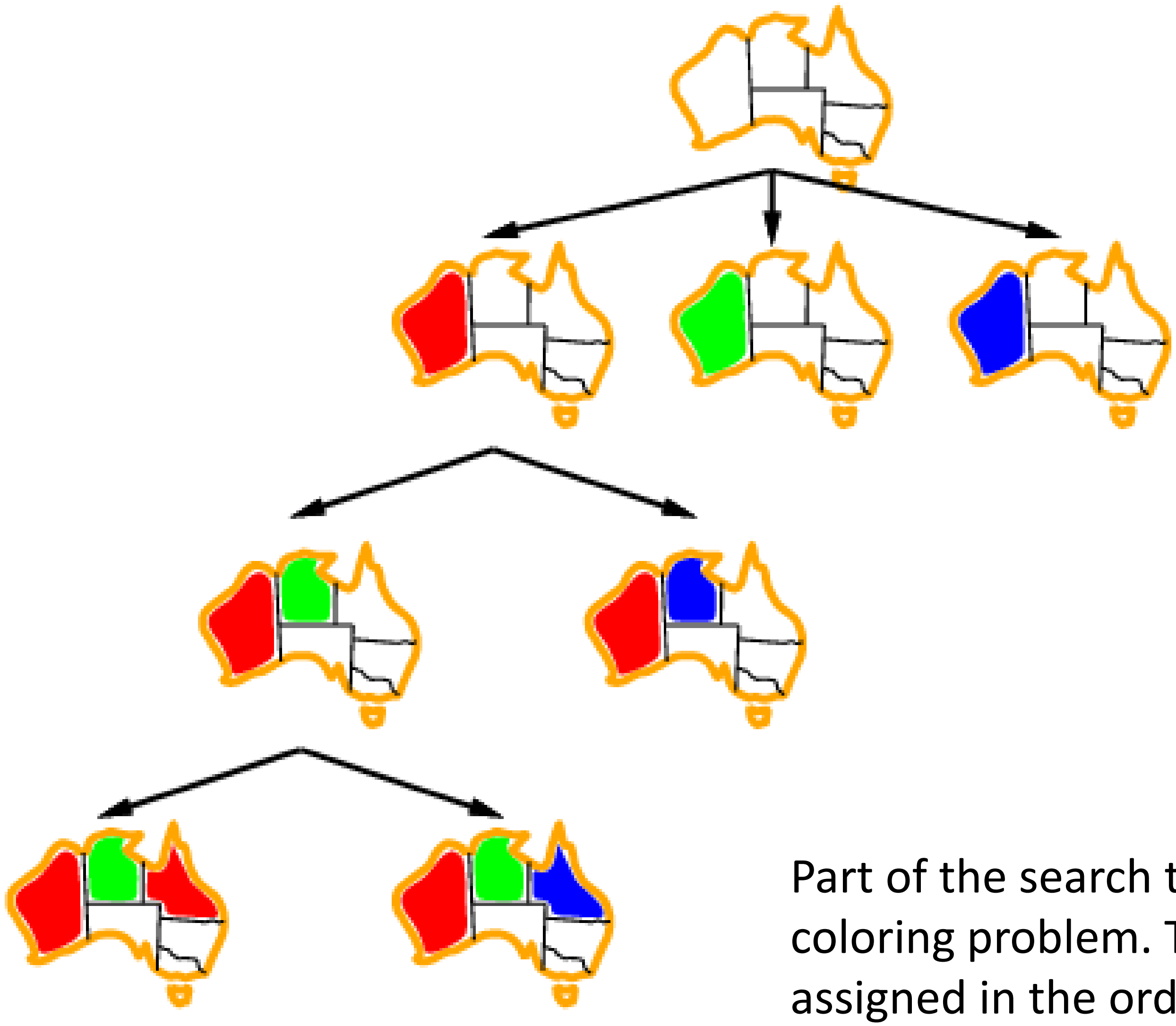
CSP as a Search problem

- Let a state be a partial assignment and an action extend the assignment.
- A standard depth-limited search can be used solve CSPs.
  - Initial state: empty assignment { }
  - Successor function: assign a value to an unassigned variable that agrees with the current assignment $\rightarrow$ fail if no legal assignments
  - Goal test: the current assignment is complete
- All the complete assignments of n variables of domain size d appears at depth n on the search tree.
  - The branching factor $b = (n - l)d$ at depth l, there are $n! \cdot d^n$ leaves with only d^n complete assignments!

The commutativity of CSPs

- A problem is commutative if the order of application of any given set of actions does not matter.
- Variable assignments in CPS are commutative.
  - E.g., [$WA = red$ then $NT = green$] $\equiv$ [$NT = green$ then $WA = red$]
- We need only consider a single variable at each node in the search tree.
  - The branching factor $b = d$, and there are d^n leaves.

Backtracking search: An example



Part of the search tree for the map-coloring problem. The variables are assigned in the order WA, NT, Q,…

```
function BACKTRACKING-SEARCH(csp) returns a solution or failure
    return BACKTRACK(csp, { })
```

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

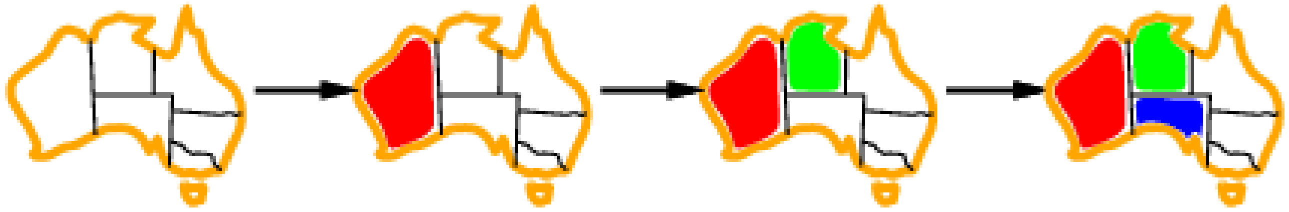
Which variable should be assigned next?

In what order should its values be tried?

What inferences should be performed?

Minimum-remaining-values heuristic

- MRV heuristic takes the variable with the fewest legal values.
  - It picks a variable that is most likely to cause a failure soon, thereby pruning the search tree.

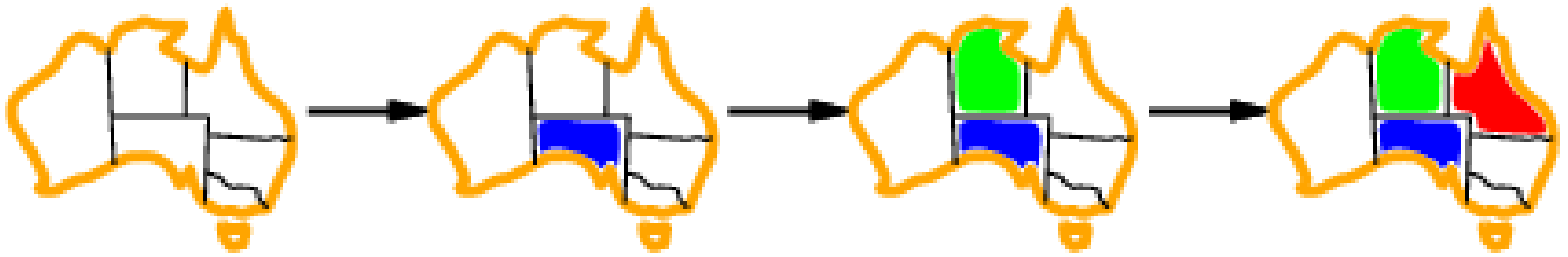


There is only one possible value for SA after $[WA = red, NT = green]$.

- It usually performs better than a random or static ordering.
  - The advantage is sometimes by orders of magnitude, yet the results vary depending on the problem.

Degree heuristic

- DH heuristic selects the variable that involves in the largest number of constraints on other unassigned variables.
  - It attempts to reduce the branching factor on future choices.

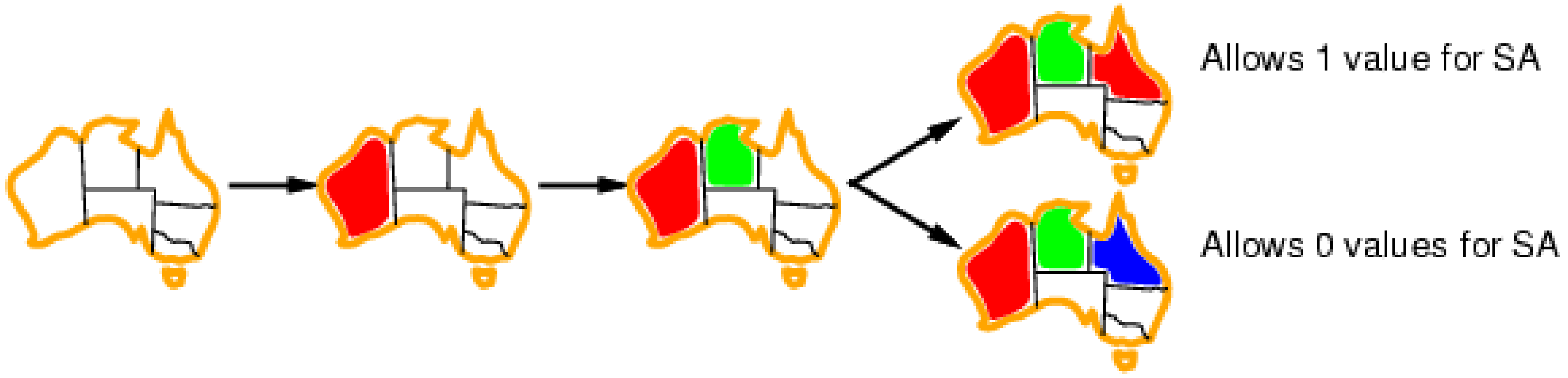


SA has a highest degree of 5, other variables except T have degrees of 2 or 3.

- DH is the tie-breaker among most constrained variables.

Least constraining value heuristic

- LCV heuristic prefers the value that rules out the fewest choices for neighboring variables in the constraint graph.
  - It tries to leave the maximum flexibility for subsequent variable assignments.

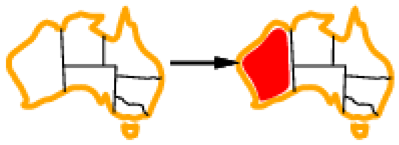


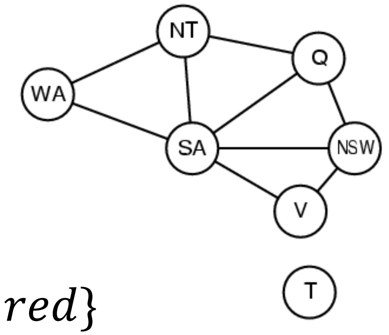
- In conclusion, variable selection should be fail-first, while value choice should be fail-last. Why?

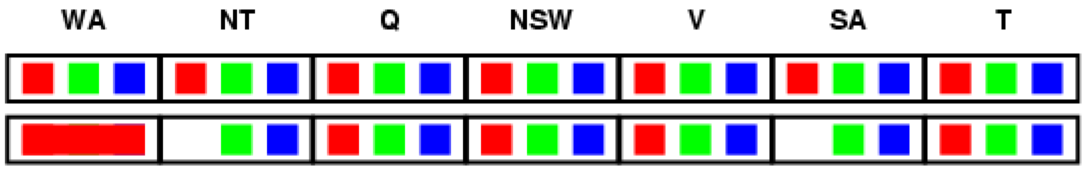
Interleaving search and inference

- Let X be the variable just been assigned and Y be each unassigned variable that connects X by some constraint.
- Forward checking (FC) deletes from Y's domain any value that is inconsistent with the value chosen for X.

- Combining MRV heuristic with forward checking effectively improves the search in many problems.
- FC detects many inconsistencies, but not all of them.
  - It makes the current variable arc-consistent but does not look ahead.

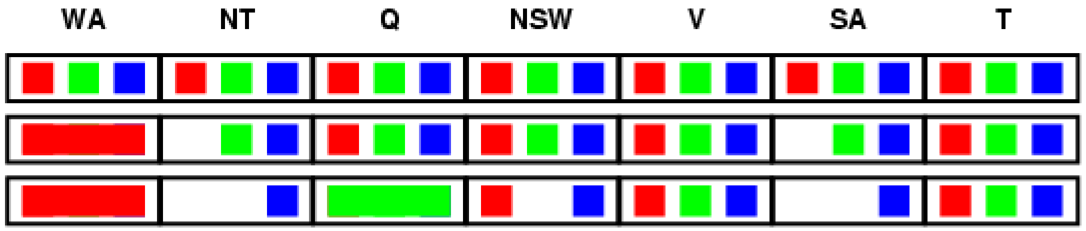






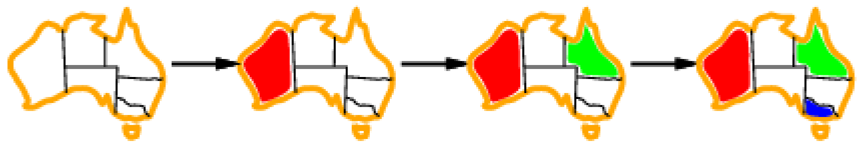
WA	NT	Q	NSW	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦

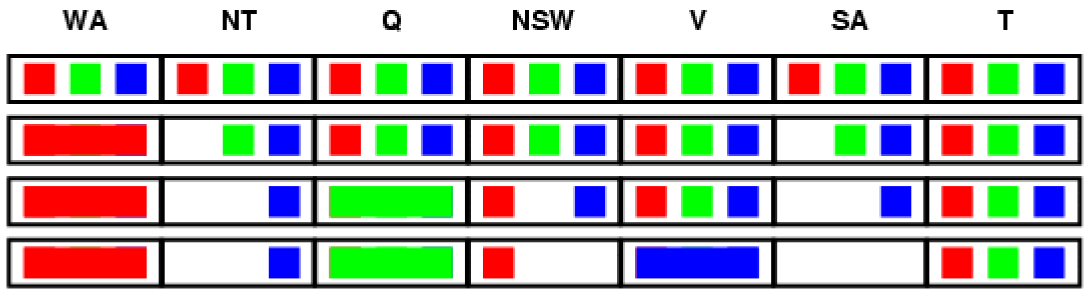
- Assign $\{WA = red\}$
- NT and SA can no longer be red



WA	NT	Q	NSW	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦
🟥	🟦	🟩	🟥🟦	🟥🟩🟦	🟦	🟥🟩🟦

- Assign $\{Q = green\}$
- NT, NSW and SA can no longer be green.



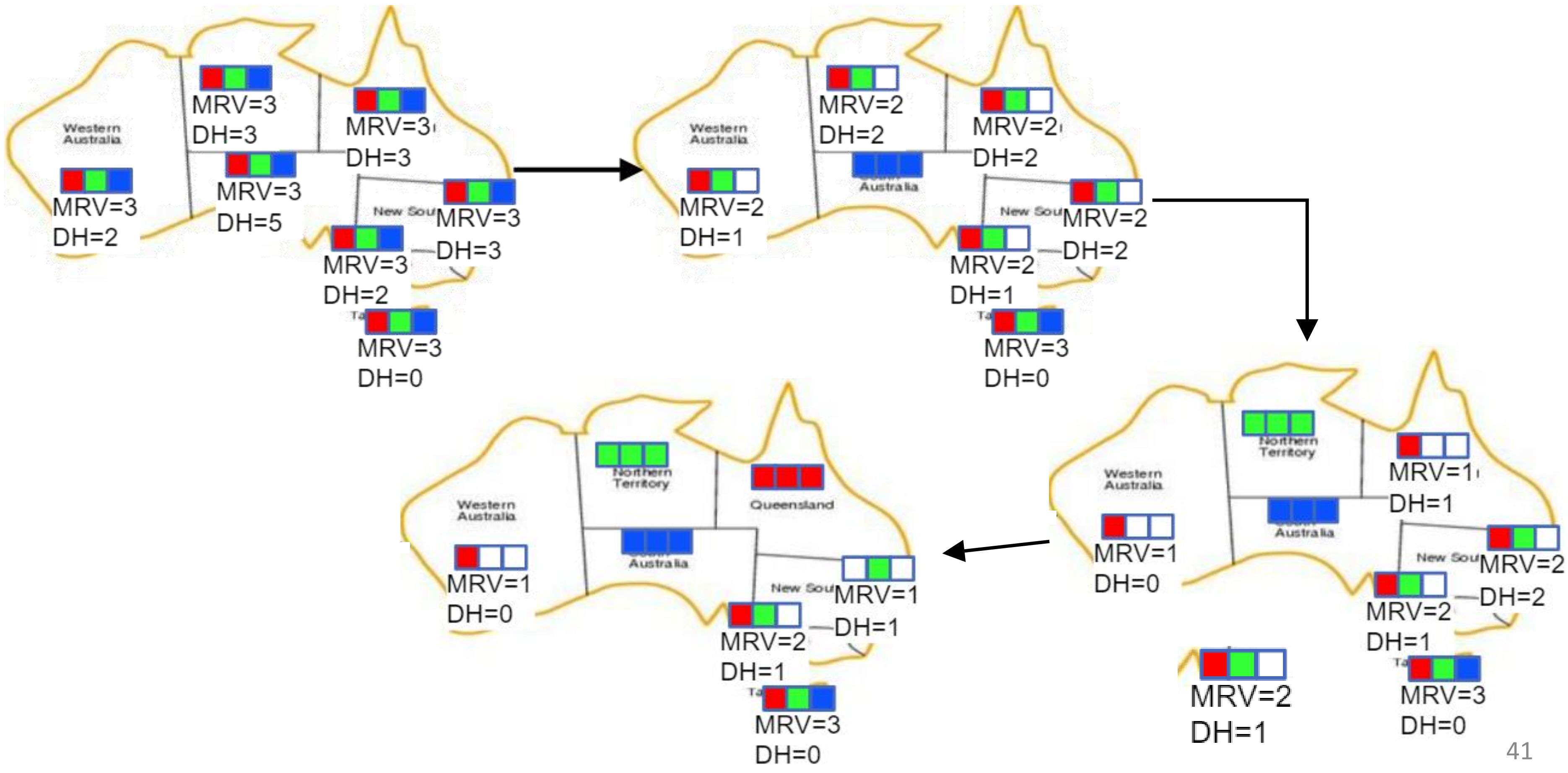


WA	NT	Q	NSW	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦
🟥	🟦	🟩	🟥🟦	🟥🟩🟦	🟦	🟥🟩🟦
🟥	🟦	🟩	🟥	🟦		🟥🟩🟦

- Assign $\{V = blue\}$
- NSW can no longer be blue.
- SA is empty

Example: Australian map coloring

- The map coloring problem can be easily solved by using forward checking and MRV heuristic.

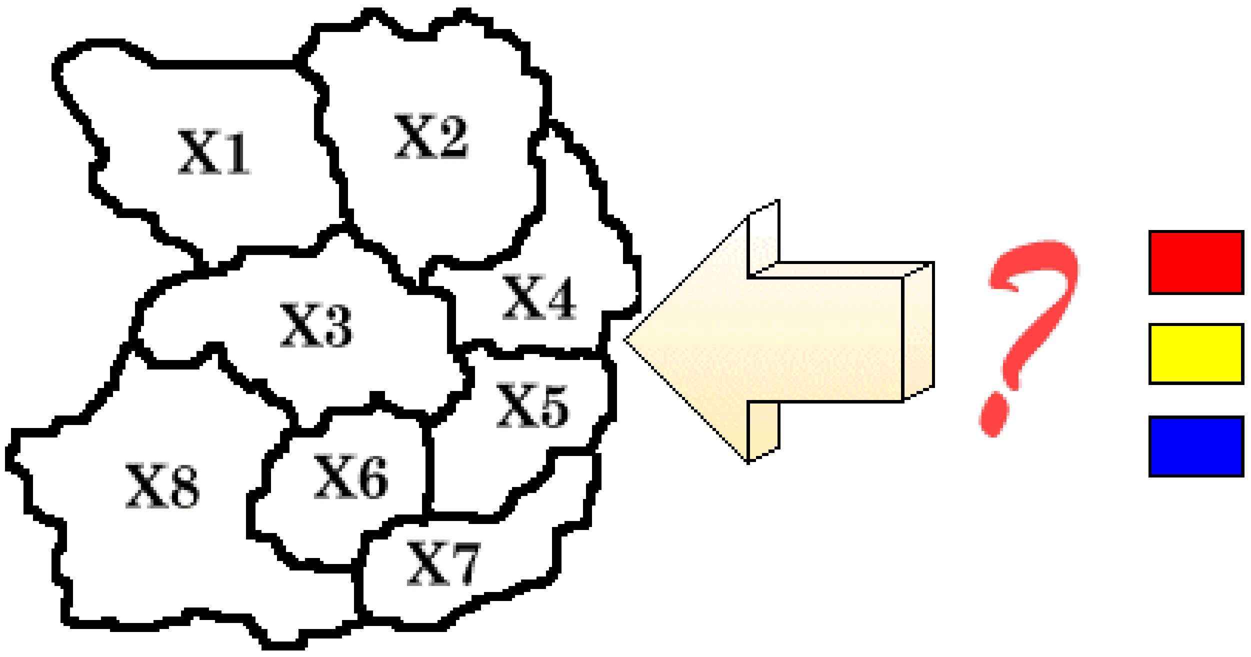


Maintaining Arc Consistency (MAC)

- Consider the (directed) arc (X_j, X_i), where X_i is the assigned variable and X_j is each unassigned neighbor of X_i.
- The MAC algorithm starts with only (X_j, X_i) and applies AC-3 for constraint propagation.
- If any variable has its domain reduced to the empty set, AC-3 fails and backtrack occurs immediately.

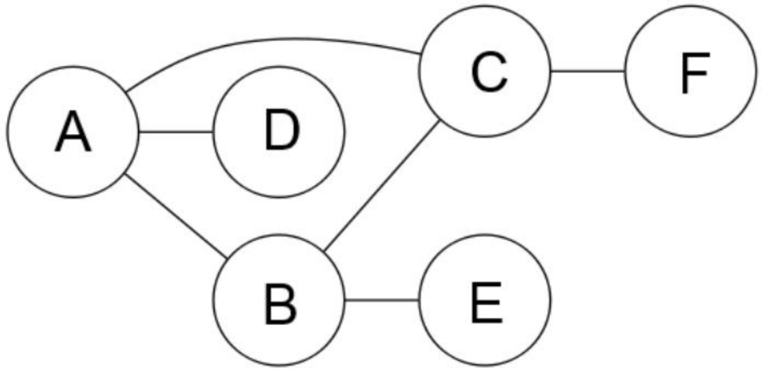
Quiz 03: Map coloring problem

- Coloring each region either **red**, **yellow**, or **blue** in such a way that no neighboring regions have the same color



Quiz 04: AC vs. FC

- The graph shown aside is a constraint graph for a CSP that has only binary constraints. Initially, no variables have been assigned.



- For each of the given scenarios, mark all variables for which the specified filtering might result in their domain being changed. Note that every scenario is independent from the others.

Quiz 04: AC vs. FC

- A value is assigned to A. Which domains might be changed as a result of running **forward checking** for A?

  ☐ A ☐ B ☐ C ☐ D ☐ E ☐ F

- A value is assigned to A, and then forward checking is run for A. Then a value is assigned to B. Which domains might be changed as a result of running **forward checking** for B?

  ☐ A ☐ B ☐ C ☐ D ☐ E ☐ F

- A value is assigned to A. Which domains might be changed as a result of enforcing **arc consistency** after this assignment?

  ☐ A ☐ B ☐ C ☐ D ☐ E ☐ F

- A value is assigned to A, and then arc consistency is enforced. Then a value is assigned to B. Which domains might be changed as a result of enforcing **arc consistency** after the assignment to B?

  ☐ A ☐ B ☐ C ☐ D ☐ E ☐ F

Local search for CSP

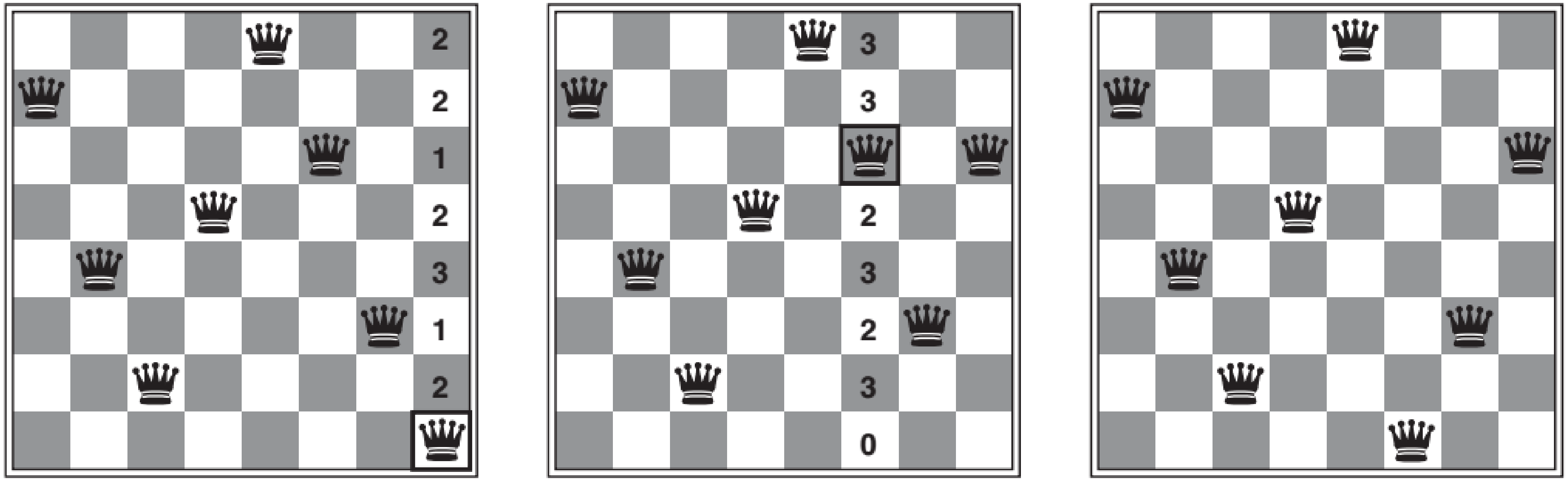
Local search for CSPs

- Complete-state formulation
  - The initial state assigns a value to every variable $\rightarrow$ violate constraints
  - The search changes the value of one variable at a time $\rightarrow$ resolve the confliction
- ***Min-conflicts heuristic:*** *the minimum number of conflicts with other variables*
- Min-conflicts is surprisingly effective for many CSPs.
  - Million-queens problem can be solved ~ 50 steps
  - Hubble Space Telescope: the time taken to schedule a week of observations down from 3 weeks (!) to ~10 minutes

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 ← 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
```

MIN-CONFLICTS: 8-queens



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 attacking queens (i.e., conflicts) is shown in each square.
The algorithm moves the queen to the min-conflicts square, breaking ties randomly.

Local search for CSPs

- The landscape of a CSP under the min-conflicts heuristic usually has a series of plateau.
    - There are millions of variable assignments that are only one conflict away from a solution.
- Plateau search: allow sideways moves to another state with the same score
- Tabu search: keep a small list of recently visited states and forbid the algorithm to return to those states
- Simulated annealing can also be used

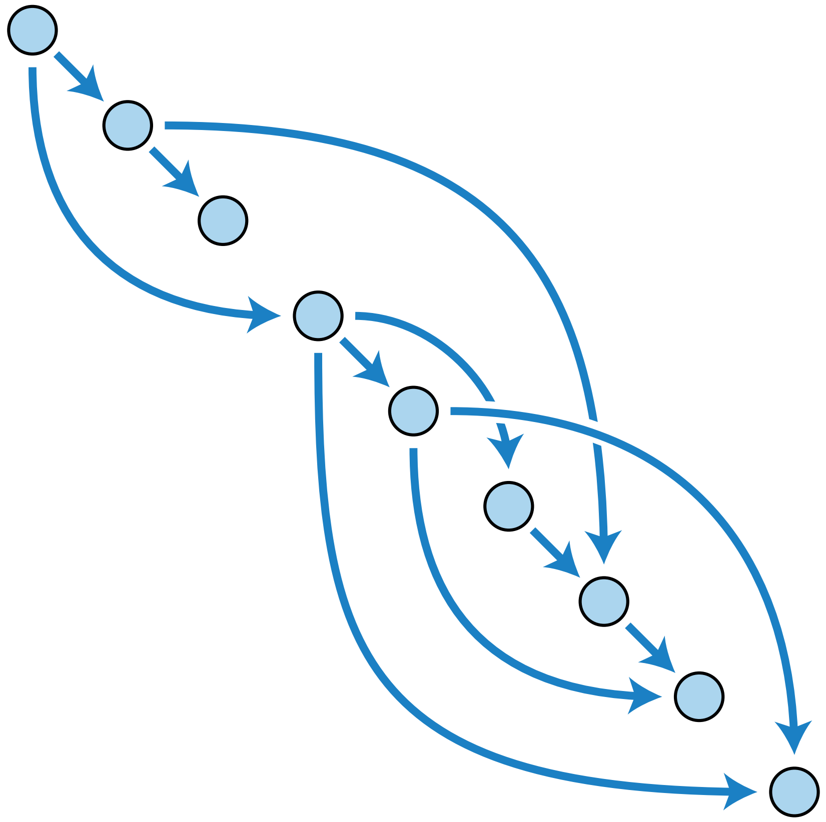
Constraint weighting

- Concentrate the search on the important constraints
- Each constraint is given a numeric weight, W_i, initially all 1.
- At each step, choose a variable/value pair to change that has the lowest total weight of all violated constraints
- Increase the weight of each constraint that is violated by the current assignment

Local search in online setting

- Scheduling problems: online setting
  - A weekly airline schedule may involve thousands of flights and tens of thousands of personnel assignments
  - The bad weather at one airport can render the schedule infeasible.
- The schedule should be repaired with a minimum number of changes.
  - Done easily with a local search starting from the current schedule
  - A backtracking search with the new set of constraints usually requires much more time and might find a solution with many changes from the current schedule

The structure of problems



Independent subproblems

- *If assignment S_i is a solution of CSP_i, then $\bigcup_i S_i$ is a solution of $\bigcup_i CSP_i$.*
  - For example, the Australia map coloring: Tasmania and the mainland

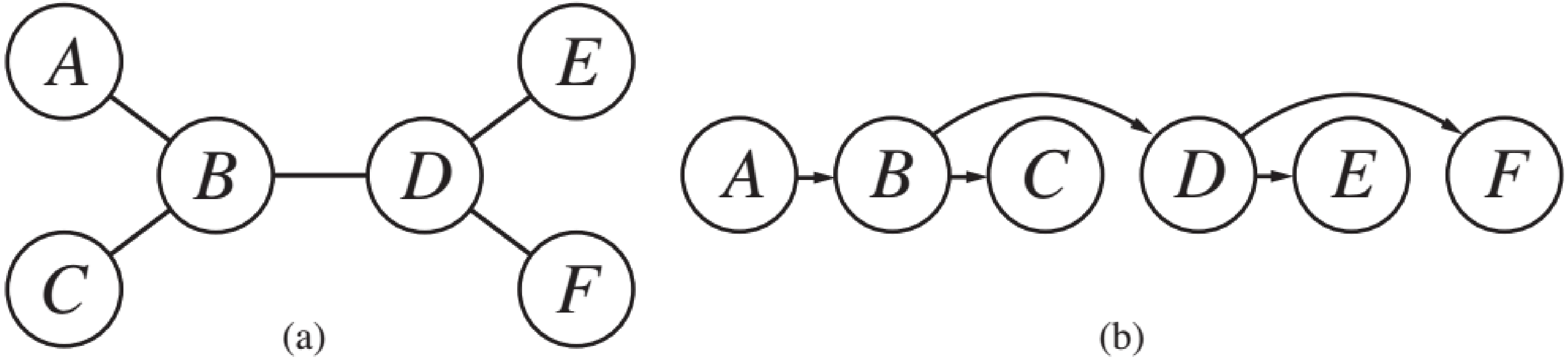
- Suppose each CSP_i has c variables from n variables.
- Then there are n/c subproblems, each of which takes at most d^c work to solve.
  - where c is a constant and d is the size of the domain.
- Hence, the total work is $O(d^c n/c)$, which is linear in n.
  - Without the decomposition, the total work is $O(d^n)$.

Tree-structured CSP

- A constraint graph is a tree when any two variables are connected by only one path.
- *Any tree-structured CSP can be solved in time linear in the number of variables*
- **Directed arc consistency** (DAC): A CSP is directed arc-consistent under an ordering of variables $X_1, X_2, \ldots, X_n$ iff every X_i is arc-consistent with each X_j for $j > i$.

Tree-structured CSP

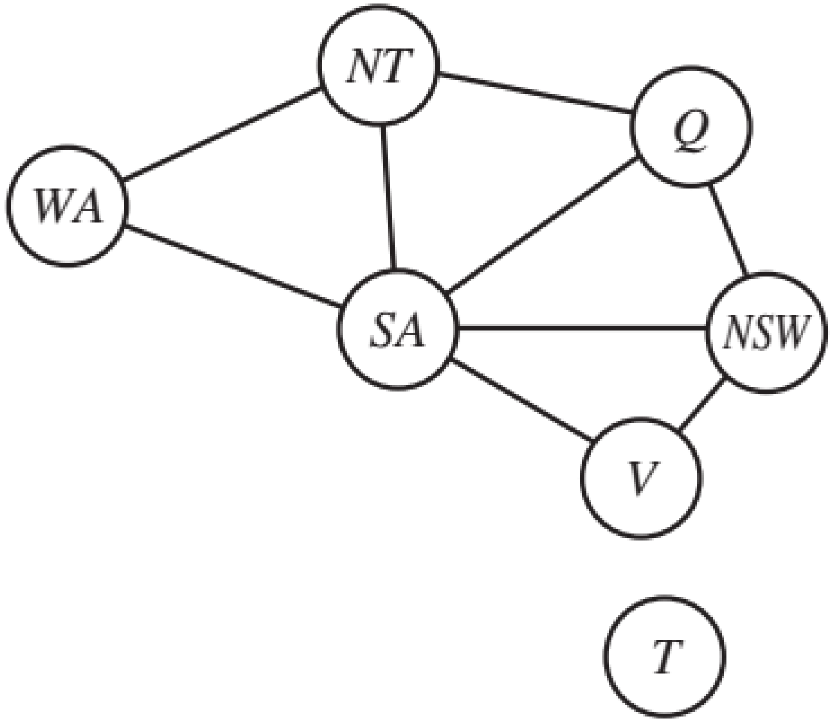
- **Topological sort:** first pick any variable to be the root of the tree and choose an ordering of the variables such that each variable appears after its parent in the tree.

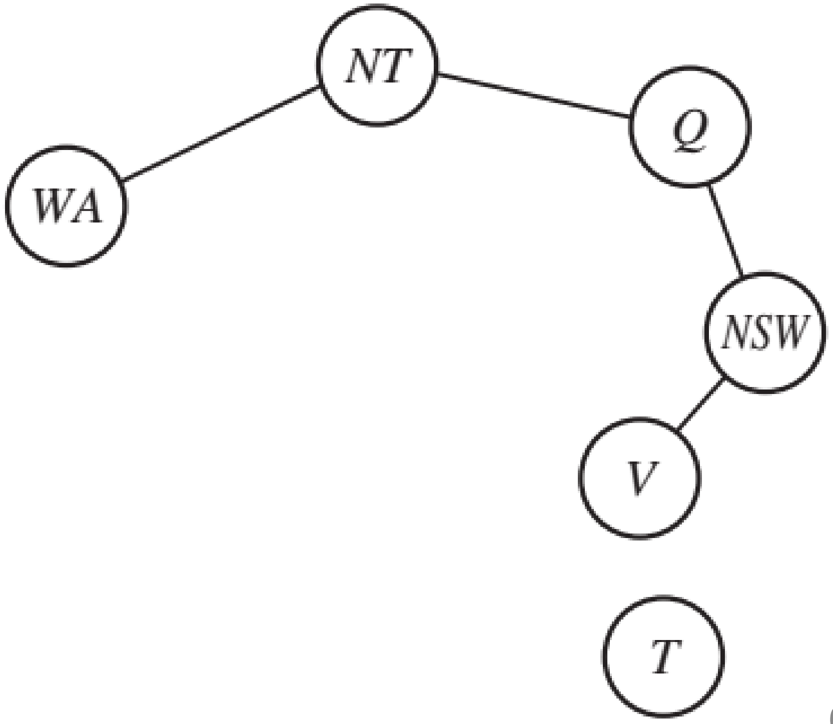


(a) The constraint graph of a tree-structured CSP.
(b) A linear ordering of the variables consistent with the tree with A as the root.

Reducing graphs to trees

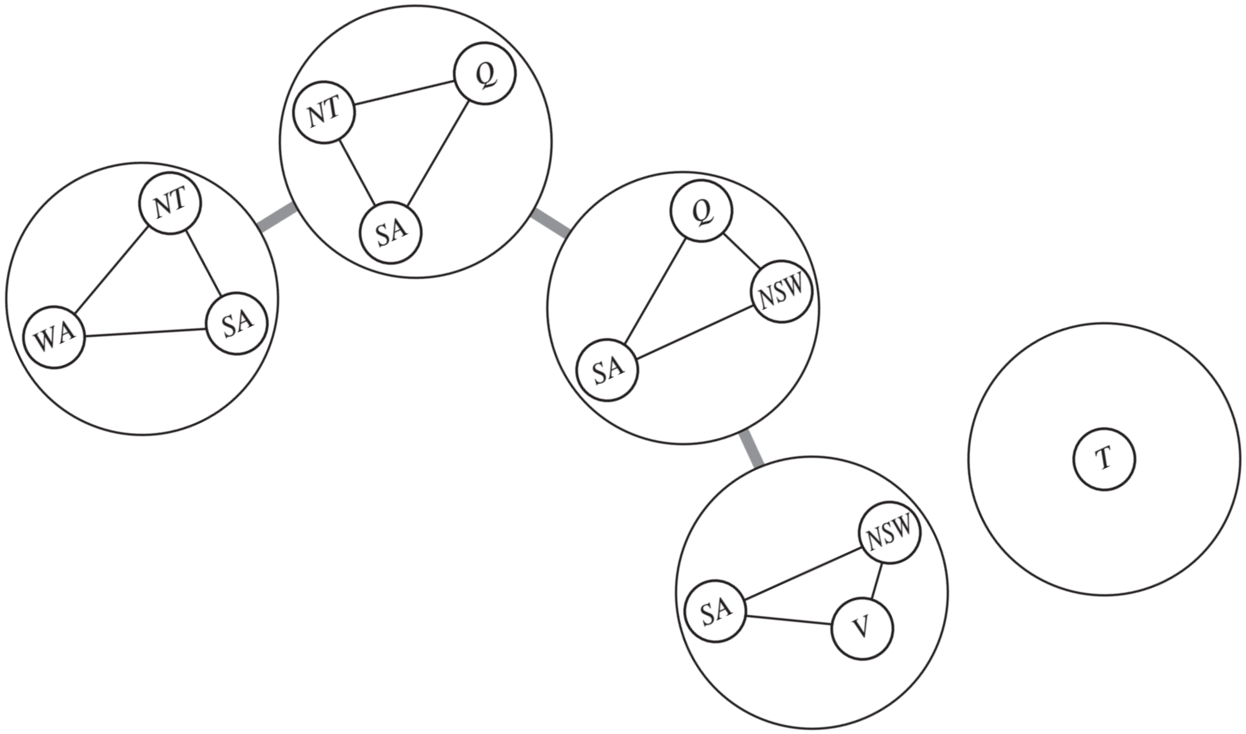
- Assign values to some variables so that the remaining variables form a tree
  - E.g., fix a value for SA and delete from other variables' domains any values that are inconsistent with the value chosen for SA





Reducing graphs to trees

- Construct a tree decomposition of the constraint graph into a set of connected subproblems.
- Each subproblem is solved independently and the resulting solutions are then combined.



The structure of values

- Consider the map-coloring problem with n colors.
- For every consistent solution, there is a set of $n!$ solutions formed by permuting the color names.
  - E.g., WA, NT, and SA must all have different colors, but there are 3! ways to assign the three colors to these three regions.
- Symmetry-breaking constraint: Impose an arbitrary ordering constraint that requires the values to be in alphabetical order
  - E.g., $NT < SA < WA \rightarrow$ only one solution possible: $\{NT = blue, SA = green, WA = red\}$

...the end.