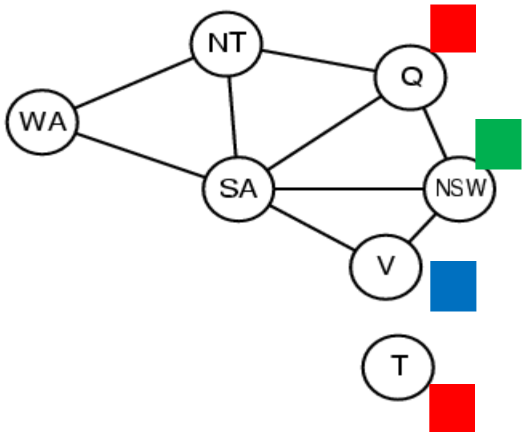
CONSTRAINT SATISFACTION PROBLEMS – PART IV

Chapter 6

Outline

- Constraint Satisfaction Problems (CSP)
- Backtracking search
- Backjumping
- No-good
- Forward checking
- Constraint propagation
- Local search for CSPs

Chronological backtracking



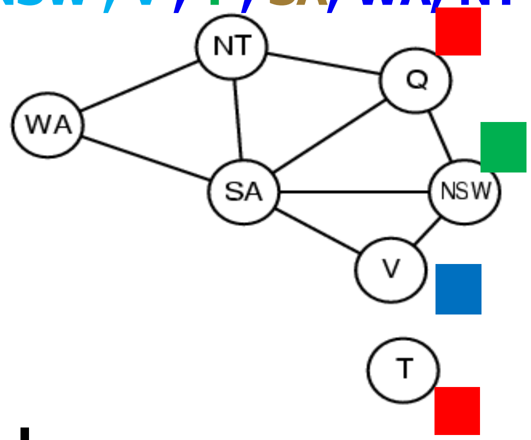
Chronological backtracking

- Backtrack to the <u>previous variable</u> and try another value

- **Example:** Assume a **fixed variable ordering** Q, NSW , V , **T** , **SA**, WA, NT
  - Suppose the **partial assignment** {Q=red, NSW =green, V =blue, **T** =red}
  - The next variable is **SA**, but **every value violates** a constraint
  - We **back up to T** and try a new color for Tasmania. This is <u>**not useful**</u> **!** recoloring Tasmania <u>cannot</u> possibly <u>resolve</u> the problem with **SA**

Assume variable ordering Q, NSW , V , T , SA, WA, NT

Backjumping

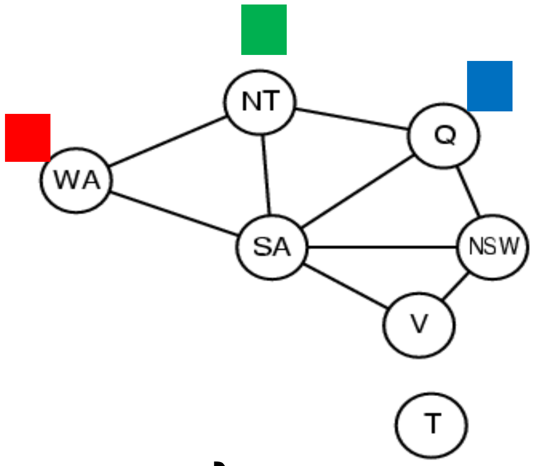


- Backtrack to **a variable** that might **fix the problem**
- The **set of these variables** is called the **conflict set**
  - **Example:** The conflict set for **SA** is {Q, NSW, V}
- Backjumping: Backtrack to the ***most recent*** **variable** in the **conflict set**
  - **Example:** We jump over **T** and try a new value for V

No-good

- To avoid redundant work
  - **To avoid** running into the **same problem again**
  - **Constraint learning**: finding a minimum set of variables from the conflict set that cause the problem. This set of variables with their corresponding values is called **no-good**
  - **Record** the **no-good** by adding a **new constraint** to the CSP

No-good

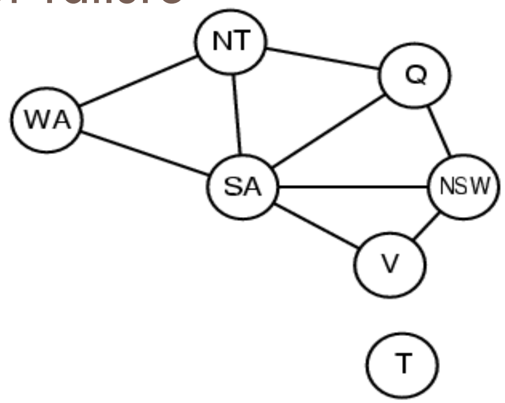


Example

- Consider the state {WA = red, NT = green, Q = blue}
- This state is a no-good, because there is no valid assignment to SA
- If the **search tree** starts by assigning values for WA, NT, Q → **recording this no-good would not help** because once we prune this branch from the search tree, we will **never encounter this combination** again
- If the **search tree** starts by assigning values for V, T → **Useful to record** {WA = red, NT = green, Q = blue} as **a no-good** because we will run into the **same problem** again **for each possible set of assignments** to V and T

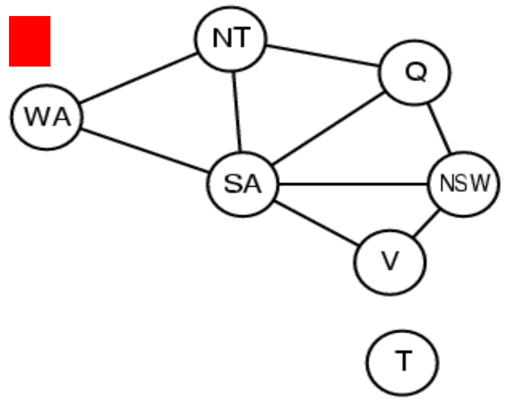
Early detection of failure

Forward checking

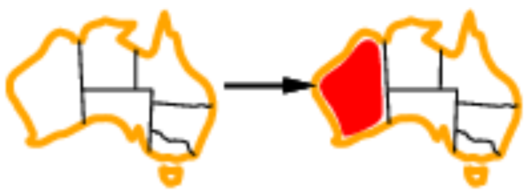


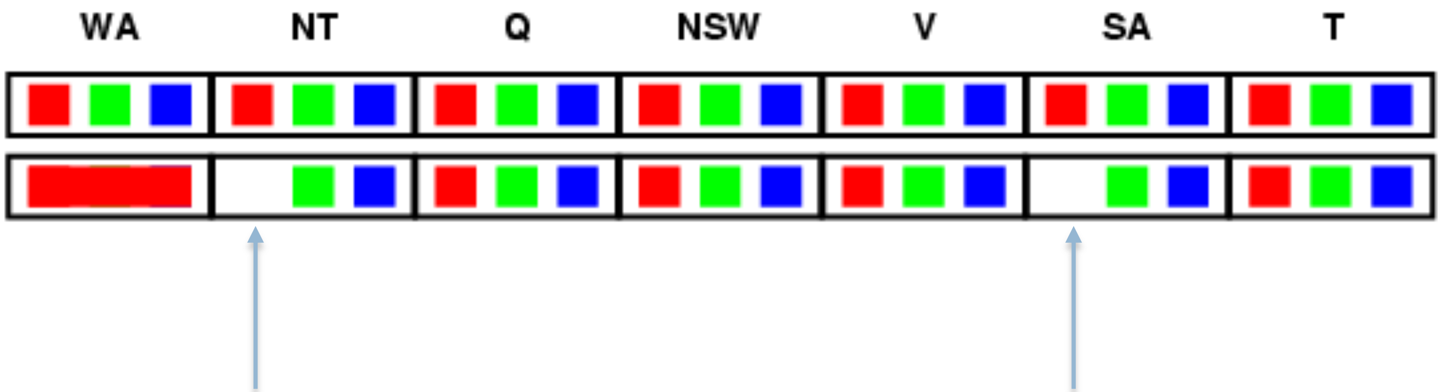
- Idea:
  - **Keep track** of **remaining legal values** for **unassigned variables**
  - **Terminate** search when any **variable** has **no legal values**

Forward checking

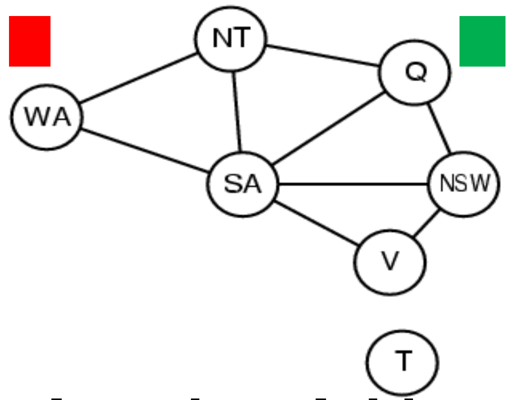


- Idea:
  - **Keep track** of remaining **legal values** for **unassigned variables**
  - **Terminate** search when any **variable** has **no legal values**

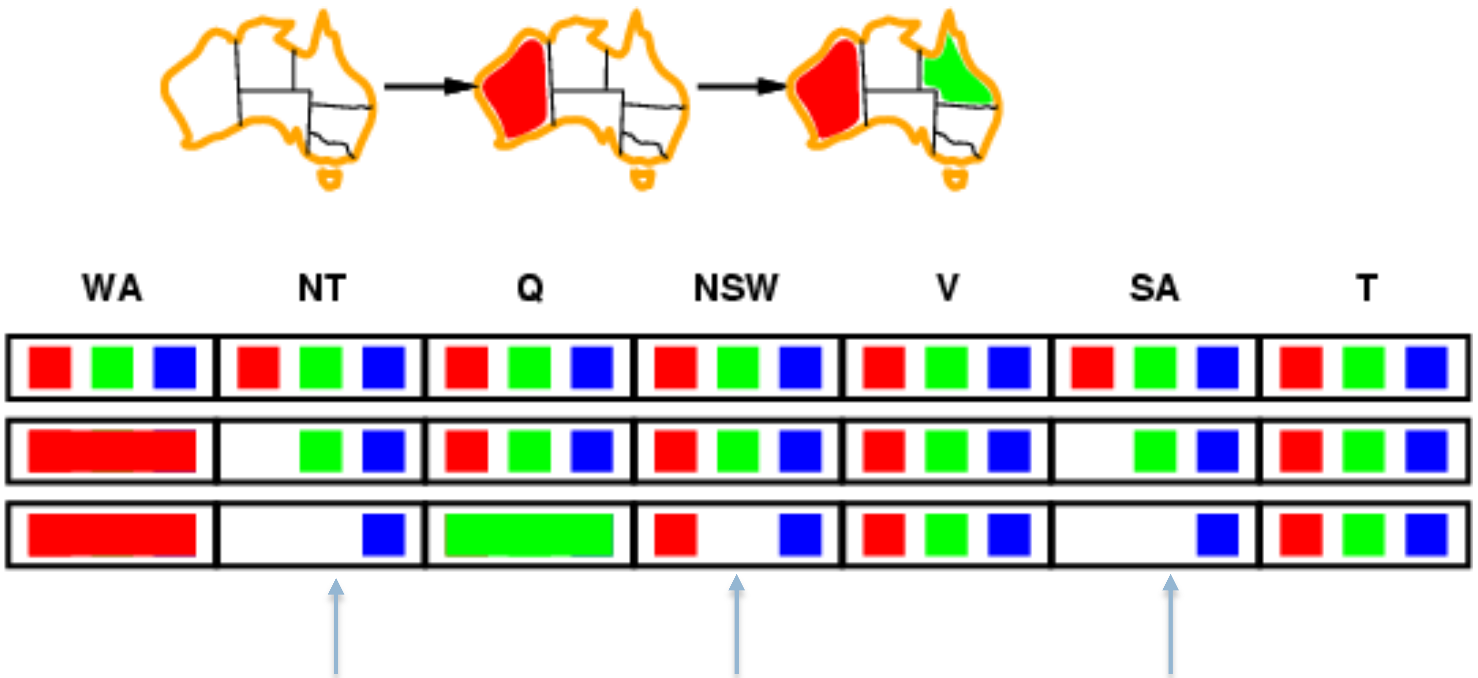




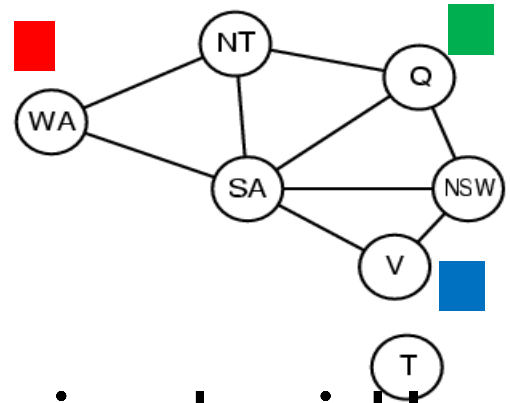
Forward checking



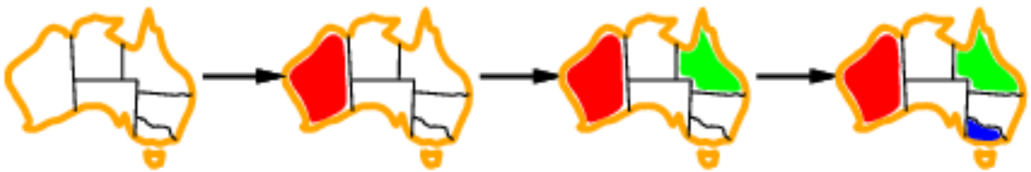
- Idea:
  - **Keep track** of remaining **legal values** for **unassigned variables**
  - **Terminate** search when any **variable** has **no legal values**



Forward checking

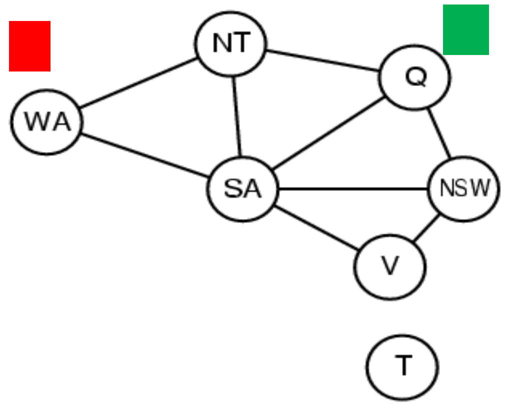


- Idea:
  - **Keep track** of remaining **legal values** for **unassigned variables**
  - **Terminate** search when any **variable** has **no legal values**



SA has no legal values

Constraint propagation



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

NT and SA cannot both be blue!

- **Constraint propagation** repeatedly enforces **constraints locally**

CONSTRAINT SATISFACTION PROBLEMS – PART V

Chapter 6

Outline

- Constraint Satisfaction Problems (CSP)
- Backtracking search
- Forward checking
- Constraint propagation
- Local search for CSPs
- Structure of the problem

Constraint propagation

- **Constraint propagation**
  - **Using constraints** **to reduce** **the number of legal values** **for a variable** → this can **reduce** the legal values **for another variable** …
  - May be **interleaved with search**
  - May be done **as a preprocessing step**, before search starts
  - **Sometimes** it **can solve the whole problem**, so **no search is required**
  - The key idea is local consistency
    - By **enforcing local consistency** in each part of the constraint graph → **inconsistent values** are **eliminated** in graph
    - There are **different types** of local consistency

Node consistency

- A **variable** (corresponding to a node in the CSP network) is **node-consistent** if **all the values** in the **variable's domain satisfy** the variable's **unary constraints**

- **Example** (a variant of map coloring)
  - Assume South Australians **dislike green**
  - Variable SA starts with domain {red,green,blue}
  - We can **make SA node consistent** by **eliminating green**, leaving SA with the reduced domain {red,blue}

- A **CSP network** is **node-consistent** if **every variable** in the network is **node-consistent**

CSPs with binary constraints

- **Unary constraints** can be eliminated by running **node consistency**
- **All n-ary constraints** can be transformed **into binary ones**

We will consider **CSPs** with only **binary constraints**

Arc consistency

- **Simplest form** of propagation **makes** each arc consistent

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

Formally:

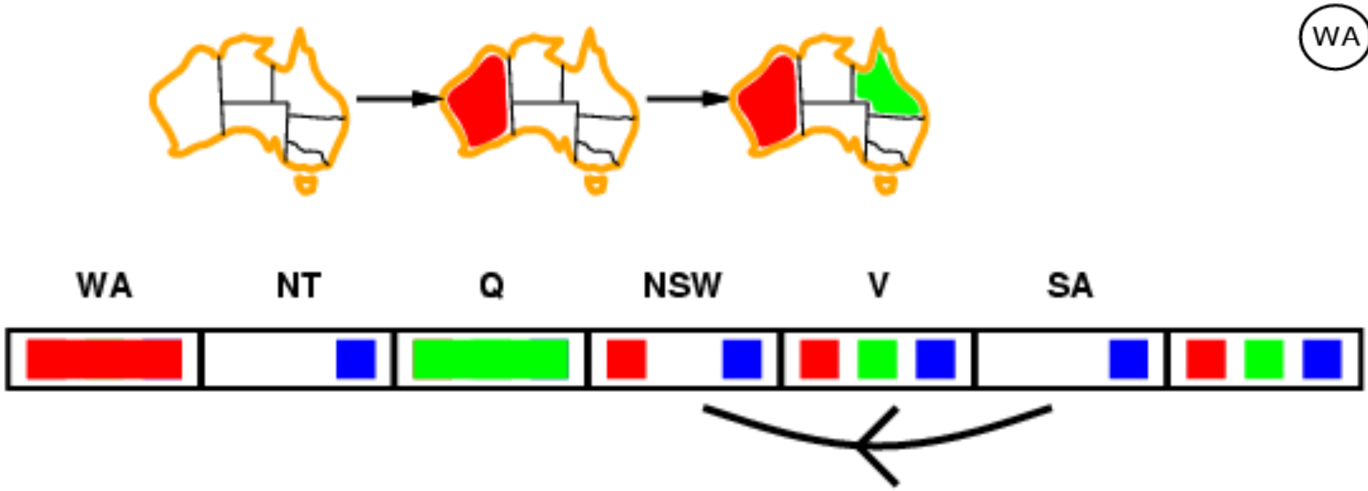
Assume there is a binary **constraint** between X_i and X_j,

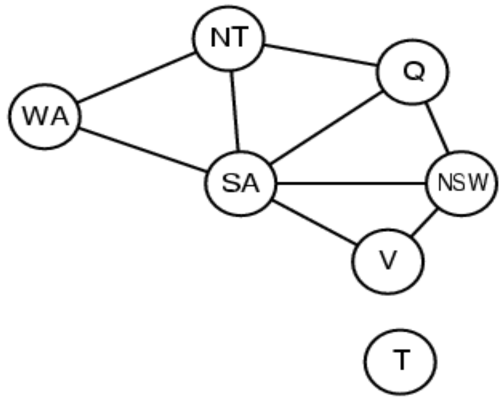
X_i is **arc consistent** with respect to X_j iff

for every value x for X_i, there is **some allowed** y *for* X_j that **satisfies** the binary constraint between X_i and X_j

Arc consistency

- X_i is **arc consistent** with respect to X_j iff
  for every value x for X_i, there is some allowed y for X_j

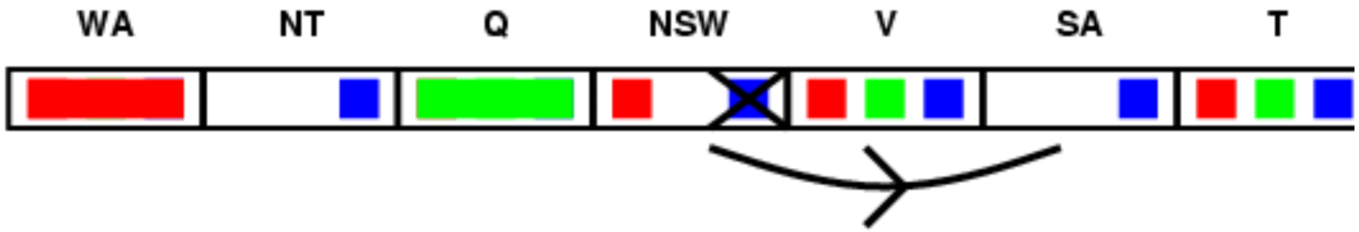


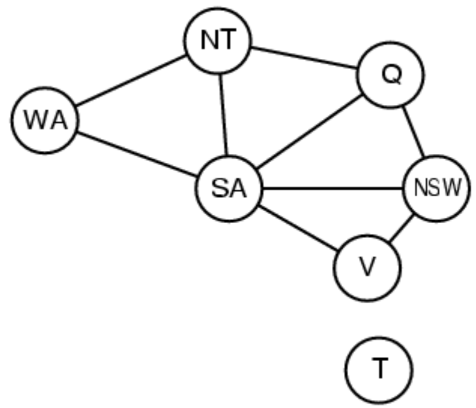


SA is **arc-consistent**
with respect to **NSW**

Arc consistency

- X_i is **arc consistent** with respect to X_j iff
  for every value x for X_i, there is some allowed y for X_j

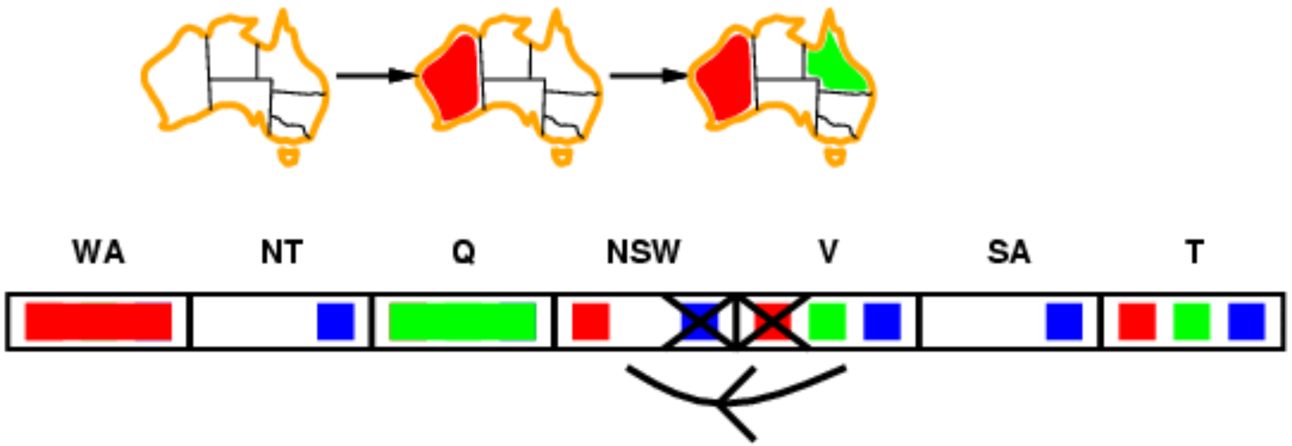


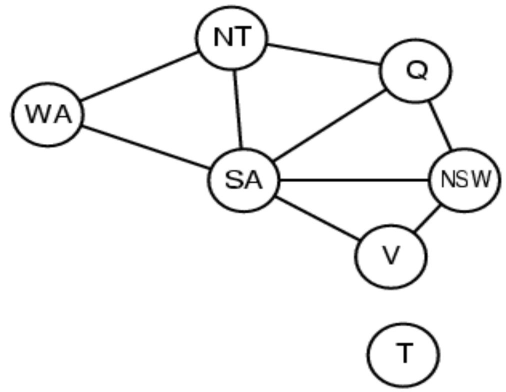


- **NSW** is **not** arc-consistent with respect to **SA**
- **To make NSW arc-consistent** with **SA**, it is sufficient to **remove** the value ***blue*** from the domain of **NSW**

Arc consistency

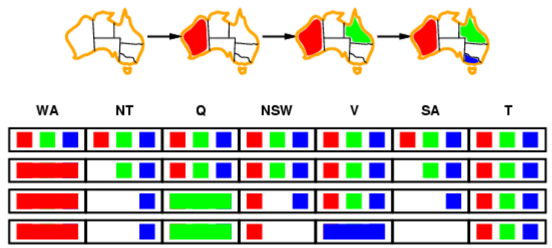
- X_i is **arc consistent** with respect to X_j iff
  for every value x for X_i, there is some allowed y for X_j



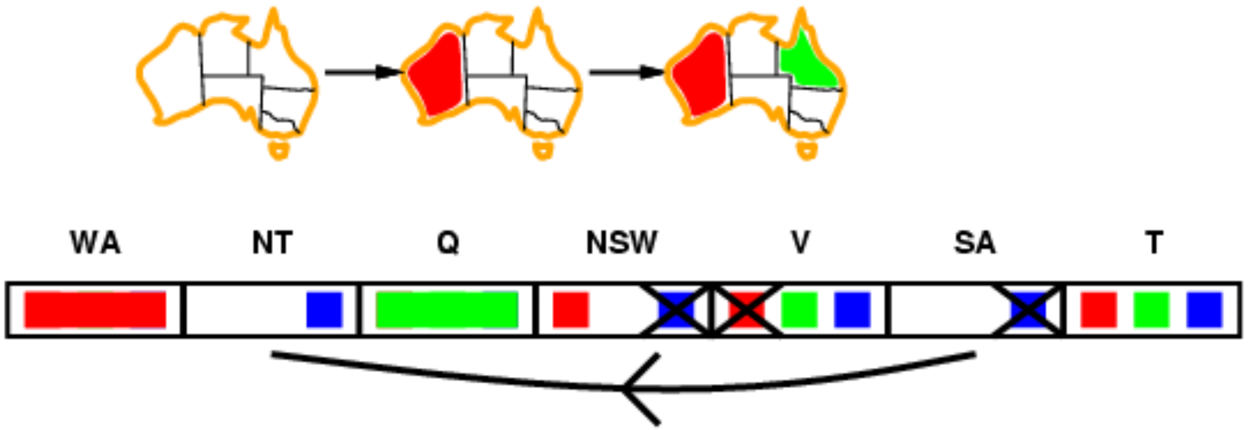


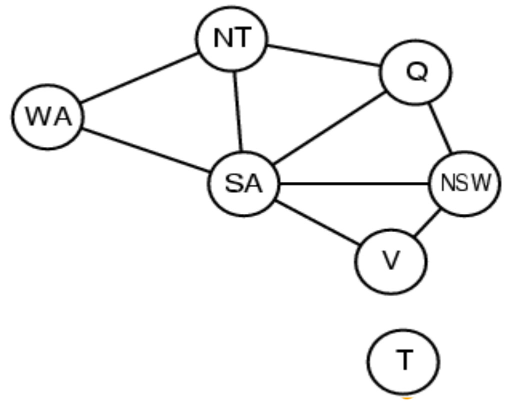
If a **variable *X*** **loses a value**,
neighbors of *X* need to be **rechecked**

Arc consistency



- X_i is **arc consistent** with respect to X_j iff
  for every value x for X_i, there is some allowed y for X_j





- **Arc consistency detects failure earlier** than **forward checking**
- Can be run as a <u>preprocessor</u> or <u>after each assignment</u>

Arc consistency algorithm AC-3 (Mackworth 1977)

```
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
```

- If **Di** unchanged → the algorithm just moves on to the next arc
- If **Di** reduced → we add to the queue all **arcs (Xk,Xi)** where Xk is a neighbor of Xi

```
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
```

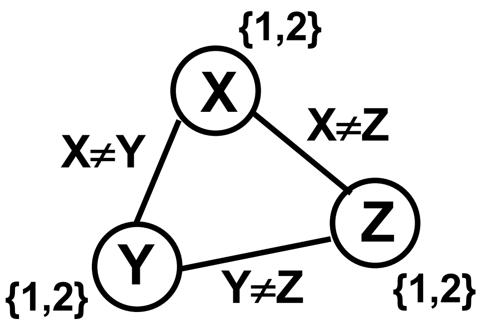
Complexity of AC-3

- For binary CSPs
  - n: number of variables
  - c: number of binary constraints (arcs)
  - d: domain size
- Time: $\boxed{O(cd^3)}$
  - **Each arc** $(X_k, \mathbf{X_i})$ **inserted** in the queue **only d times** because $\mathbf{X_i}$ has **at most d values** to delete
  - **Checking consistency of an arc** can be done in $O(d^2)$ time
  - Thus, $O(cd^3)$ total worst-case time

Is arc consistency enough?

- By **using AC** we can **<u>remove</u> many incompatible values**
  - Do we **get a solution**?
  - Do we know **if there exists a solution?**
- Unfortunately, the answer to **both** above questions is NO!

- ***Example:***



CSP is arc consistent
but there is <u>no solution</u>

Is arc consistency enough?

- **So what are the benefits of AC?**
  - **Sometimes** we have a **solution/failure after AC**
    - **a domain is empty → no solution exists**
    - **all the domains are singleton → we have a solution**
  - **In general, AC prunes the search space → equivalent easier problem**

Path consistency (PC)

- **<u>How to strengthen</u>** the consistency level?
- Require **consistency** over **<u>more than one</u> constraint**

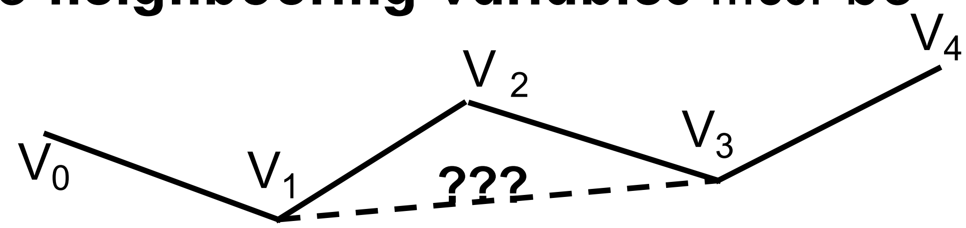
- A **two-variable set {Xi, Xj}** is **path-consistent** with respect to a third **variable Xm** if

  ∀assignment **{Xi = a, Xj = b} consistent** with the constraints on **{Xi , Xj}**
  ∃assignment to **Xm** that satisfies
  - the **constraints on {Xi , Xm}** and
  - the **constraints on{Xm , Xj }**

- This is called <u>path consistency</u> since it is like to consider
  a **path** from **Xi** to **Xj** with **Xm** in the middle

Path consistency (PC)

- Path consistency
  - does **<u>not guarantee</u>** that **<u>all</u>** <u>the</u> **<u>constraints</u>** among the **variables on the path** are <u>satisfied</u>
  - **only the constraints** between the **neighbouring variables** must be satisfied

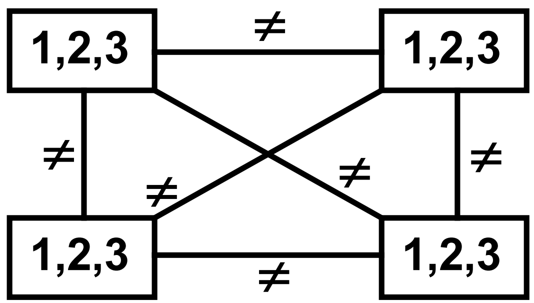


- PC is still **not a complete** technique

A,B,C,D in {1,2,3}

A≠B, A≠C, A≠D, B≠C, B≠D, C≠D

is PC but has not solution



Other stronger consistency notions…

Review: Constraint propagation

- **Constraint propagation**
  - May be **interleaved with search**
  - May be done **as a preprocessing step**, before search starts
  - **Sometimes** it **can solve the whole problem**, so **no search is required**
  - By **enforcing** **local consistency** (arc consistent, or path consistency,…) in each part of the constraint graph → **inconsistent** values are **eliminated** in graph