CS161
Discussion 4
Constraint Satisfaction Problem

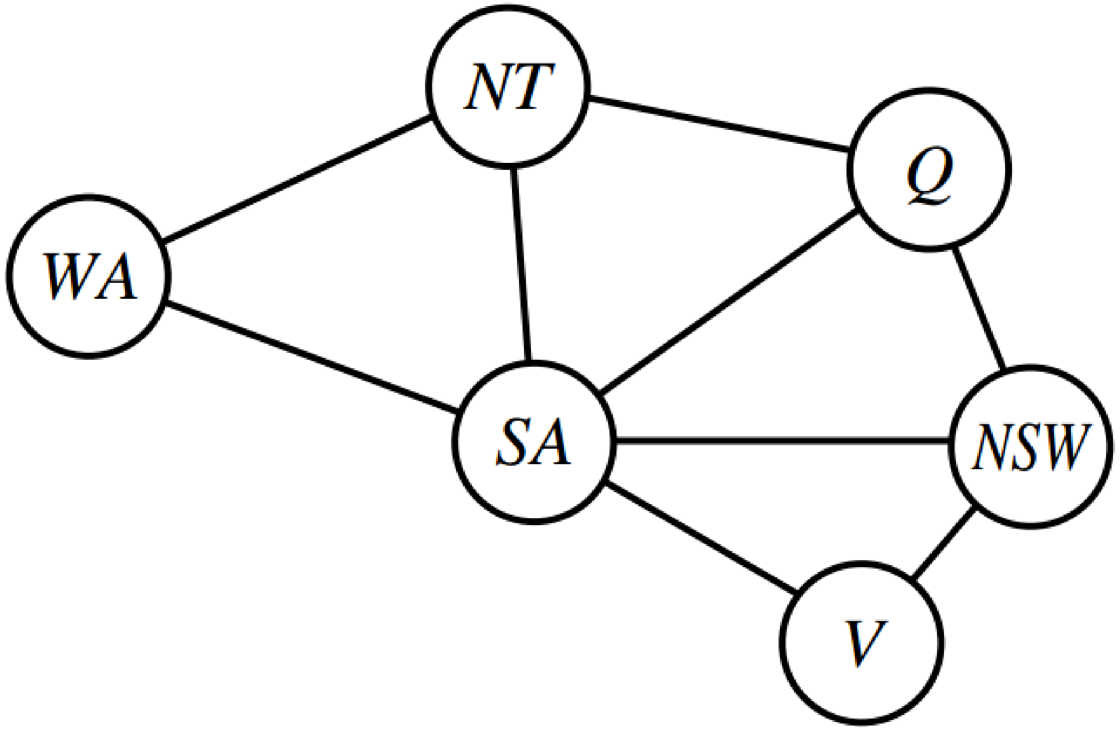
Constraint Satisfaction Problem (CSP)

X is a set of variables, $\{X_1, \ldots, X_n\}$.
D is a set of domains, $\{D_1, \ldots, D_n\}$, one for each variable.
C is a set of constraints that specify allowable combinations of values.

- A **state** in CSP: an assignment of values to some or all variables
  - *Consistent/Legal assignment*: an assignment that does not violate any constraints
  - *Complete assignment*: every variable is assigned (otherwise partial assignment)
- A **solution** in CSP: a consistent, complete assignment

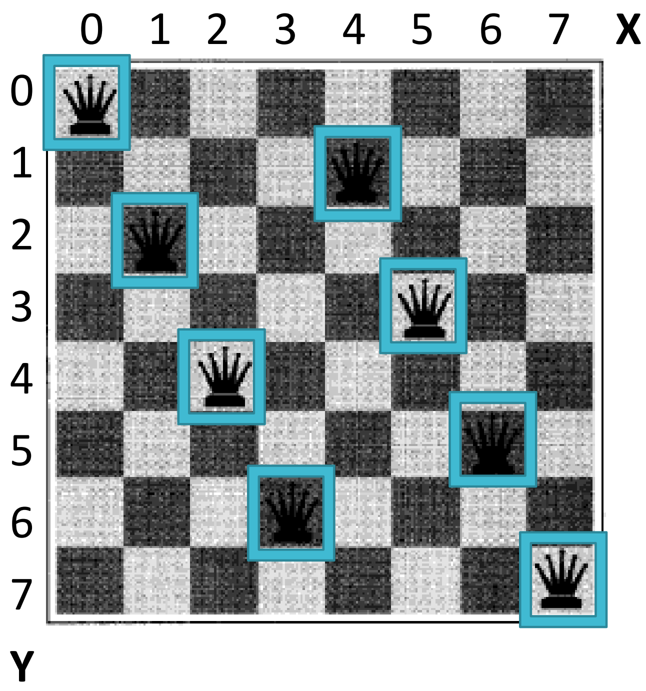
Constraint Graph

- Nodes being variables
- Arcs show constarints



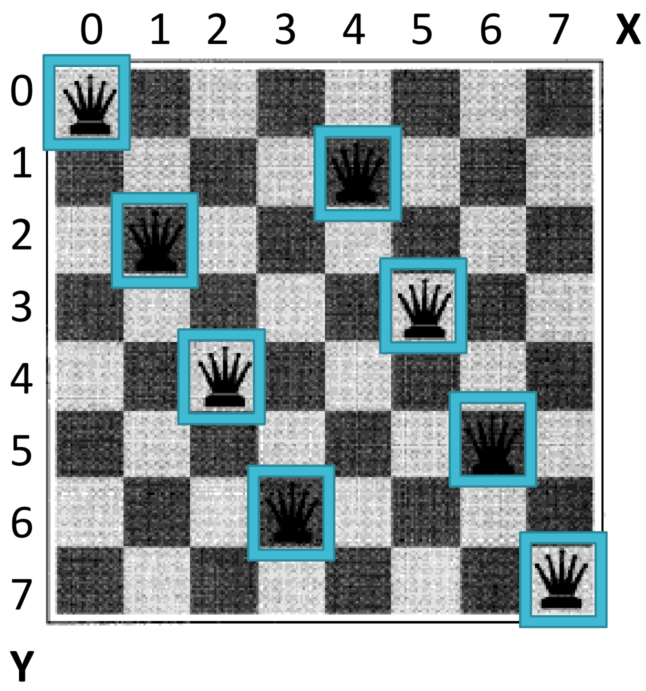
Example – 8 Queens

- **Variables:** ?
- **Domains:** ?
- **Constraints:** ?



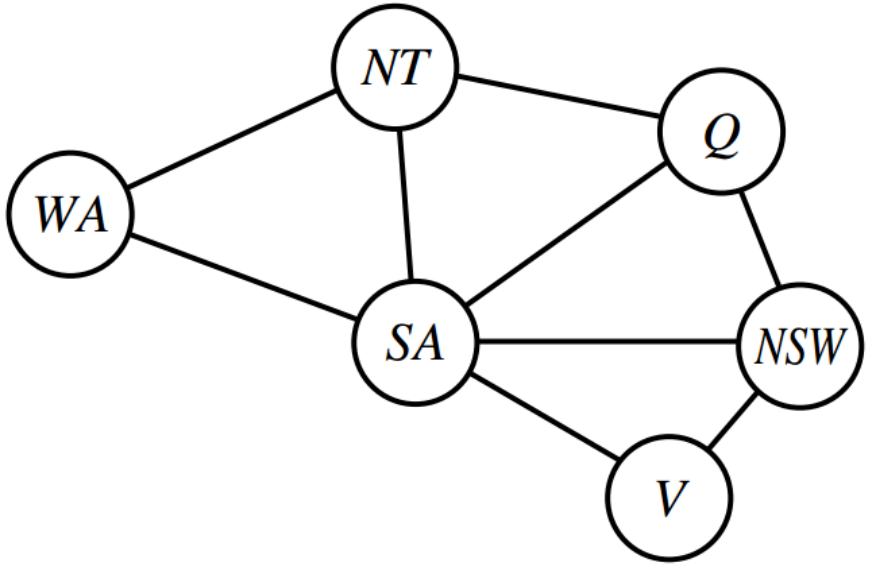
Example – 8 Queens

- **Variables:** coordinate of i-th queen, (x_i, y_i), $i = 1,2,\dots,8$
- **Domains:** {0, 1, 2, 3, 4, 5, 6, 7}
- **Constraints:** no queen attacking another



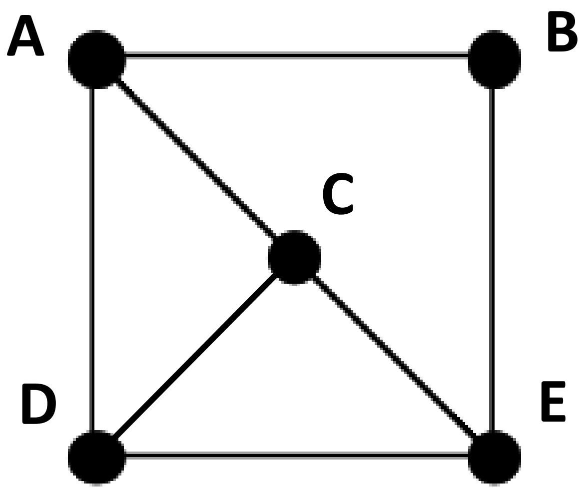
Example – Map Coloring

- **Variables:** WA, NT, Q, NSW, V, SA
- **Domains:** {red, green, blue}
- **Constraints:** adjacent regions must have different colors,
  - e.g., WA ≠ NT
  - or, (WA, NT) in {(red, green), (red, blue), (green, blue) …}



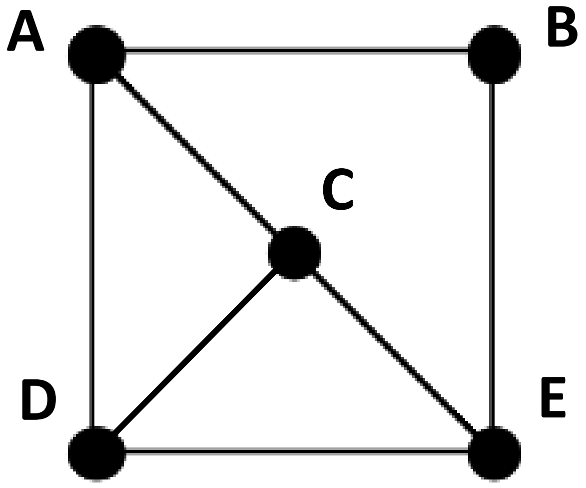
Exercise – CSP Formulations

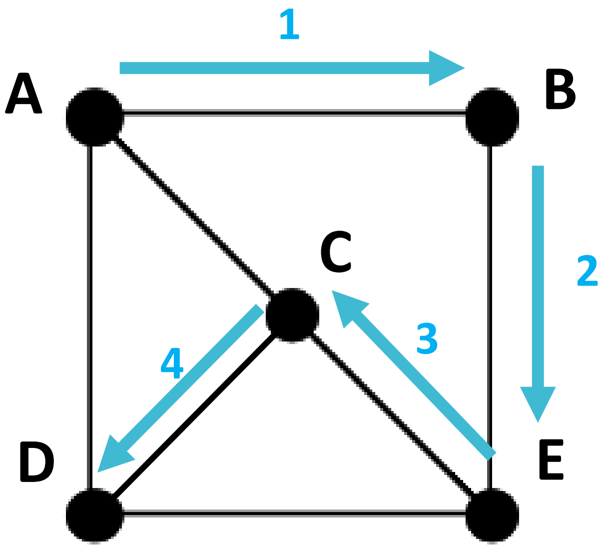
- Hamiltonian tour
  - Given a network of cities connected by roads, choose an order to visit all cities in the map without repeating any.
- How to formulate?
  - **Variables:** ?
  - **Domains:** ?
  - **Constraints:** ?



Exercise – CSP Formulations

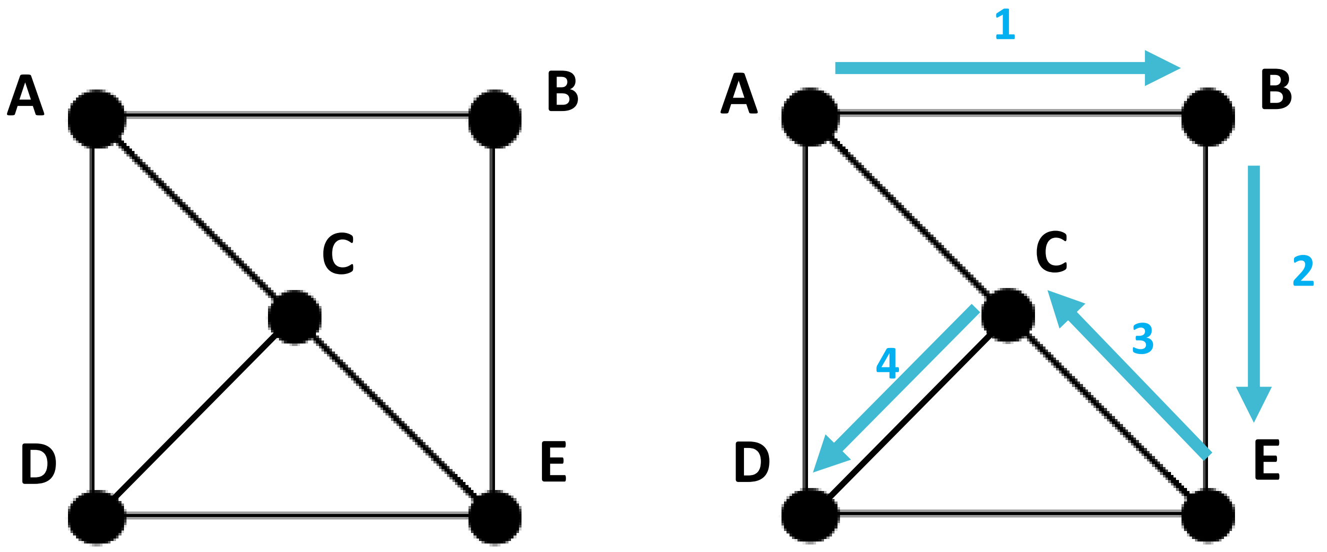
- Hamiltonian tour
  - Given a network of cities connected by roads, choose an order to visit all cities in the map without repeating any.
- How to formulate?
  - **Variables:** ?
  - **Domains:** ?
  - **Constraints:** ?





Exercise – CSP Formulations

- Hamiltonian tour
  - Given a network of cities connected by roads, choose an order to visit all cities in the map without repeating any.
- How to formulate?
  - **Variables:** Stops, X1, X2, …, X5
  - **Domains:** {A, B, C, D, E}
  - **Constraints:** Neighbor(Xi, Xi+1), for i = 1, 2, 3, 4, and Xi ≠ Xj if i ≠ j



- *How to solve CSP?*

Main Algorithm for CSP - Backtracking DFS

```
function BACKTRACKING-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
```

Main Algorithm for CSP - Backtracking DFS

```
function BACKTRACKING-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
```

Main Algorithm for CSP - Backtracking DFS

function BACKTRACKING-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 ← $\boxed{\text{SELECT-UNASSIGNED-VARIABLE}(csp)}$
  for each *value* **in** ORDER-DOMAIN-VALUES(*var*, *assignment*, *csp*) **do**
    if *value* is consistent with *assignment* **then**
      $\boxed{\text{add } \{var = value\} \text{ to } assignment}$
      inferences ← INFERENCE(*csp*, *var*, *value*)
      if *inferences* ≠ *failure* **then**
        add *inferences* to *assignment*
        result ← BACKTRACK(*assignment*, *csp*)
        if *result* ≠ *failure* **then**
          return *result*
    Backtrack
    $\boxed{\text{remove } \{var = value\} \text{ and } inferences \text{ from } assignment}$
  return *failure*

Main Algorithm for CSP - Backtracking DFS

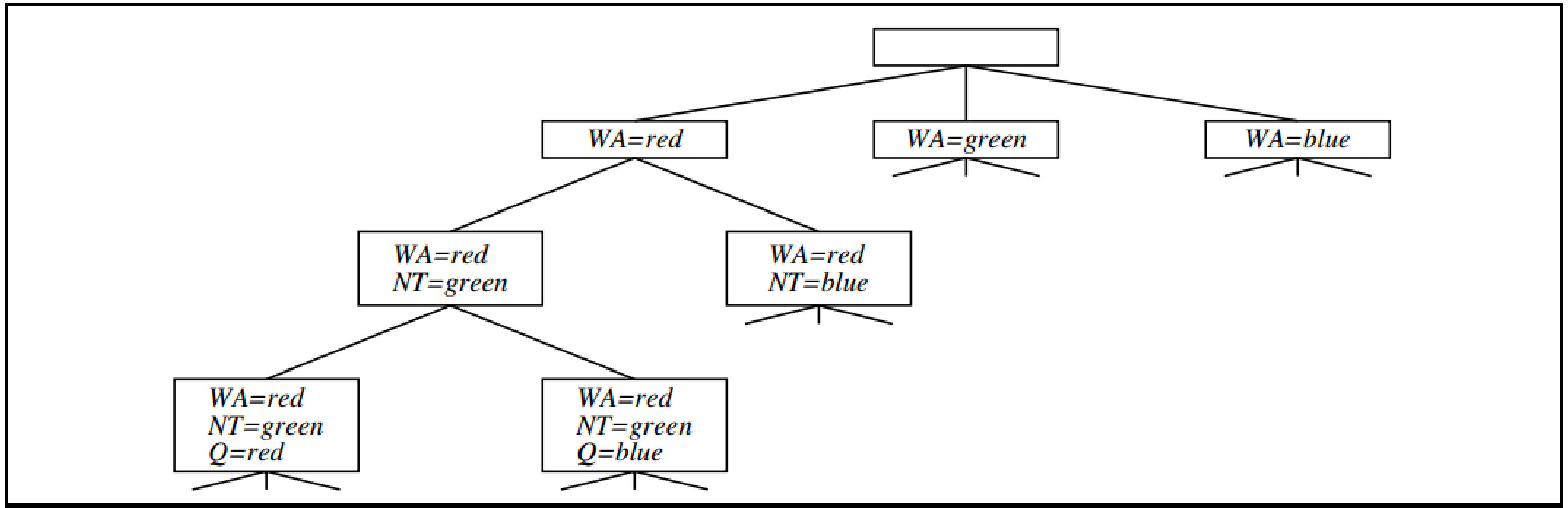
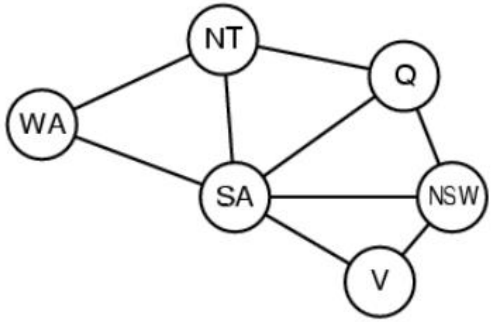


Figure 6.6 Part of the search tree for the map-coloring problem in Figure 6.1.

- When to backtrack?



Main Algorithm for CSP - Backtracking DFS

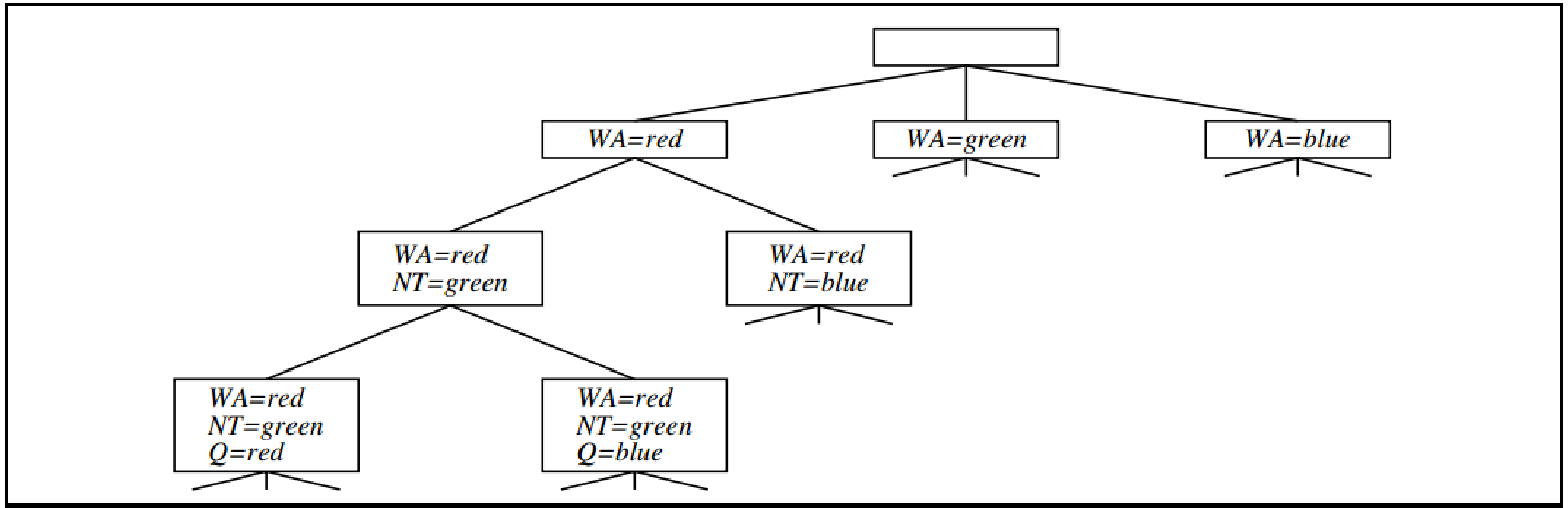
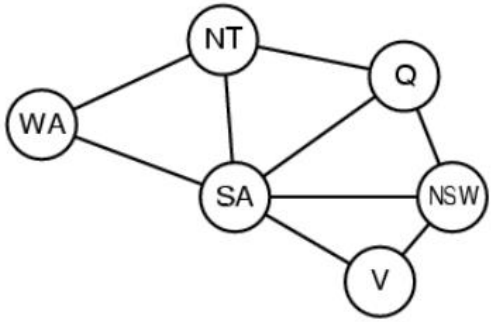


Figure 6.6 Part of the search tree for the map-coloring problem in Figure 6.1.

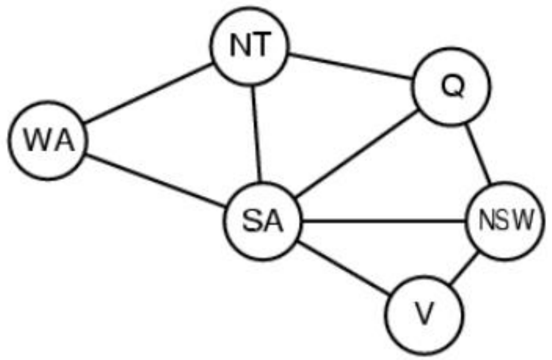
- When to backtrack?
  - When a variable has no legal values left to assign (No possible consistent assignment)



Variable and Value Ordering

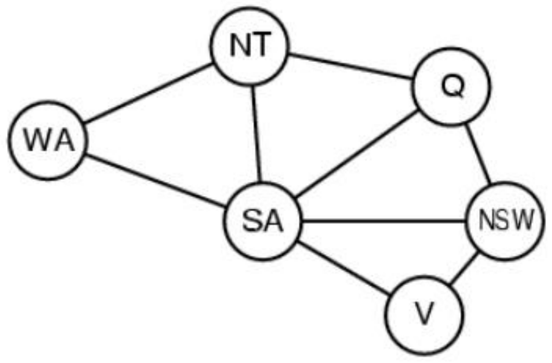
Backtracking DFS:

- Choose a variable and assign a value.
- Backtrack when no legal values left.
- Keep trying until it fails



- **How to select unassigned variable?**
- **In order what should its values be tried?**

Variable and Value Ordering



- **How to select unassigned variable?**
  - Minimum-remaining-values (MRV) heuristic
    - a.k.a. "most constrained variable", or "fail-first"
    - If no legal values left, fail immediately
  - Degree heuristic
    - Attempt to reduce branching factor on future choice
    - Useful as a tie-breaker

- **In what order should its values be tried?**
  - Least-constraining-value
    - Leave the maximum flexibility for subsequent variable assignments

- *To improve even more …*

Arc Consistency

Backtracking DFS:

Choose a variable, try a value in the variable's domain.

Can we eliminate impossible values according to constraints before further search?

- **Arc consistency**
  - Variable is *arc consistent*: Every value in its domain satisfies the variable’s binary constraints
    - X_i is arc-consistent with respect to another variable X_j if for every value in the current domain D_i there is some value in the domain D_j that satisfies the binary constraint on the arc (X_i, X_j)
  - Network is *arc consistent*: every variable is arc consistent with every other variable

Example – Arc Consistency

- The domain of both X and Y is the set of digits (0~9).
- $Y = X^2$

<u>What will the domains of X and Y be after enforcing arc consistency?</u>

Example – Arc Consistency

- The domain of both X and Y is the set of digits (0~9).
- $Y = X^2$

What will the domains of X and Y be after enforcing arc consistency?

X — Y

$X \rightarrow Y$ $\quad$ $Y = X^2$

$\Rightarrow Y \in \{0, 1, 4, 9\}$

$Y \rightarrow X$

$\Rightarrow X \in \{0, 1, 2, 3\}$

Example – Arc Consistency (Solution)

- The initial domain of both X and Y is the set of digits (0~9).
- $Y = X^2$.

- **Consider the arc (X ← Y)**
  - Make X arc-consistent with respect to Y
    - For any value in X's domain, there should be at least one value in Y's domain that satisfied the constraint
  - X: $\{0,1,2,3\}$
- **Consider the arc (Y ← X)**
  - Y: $\{0,1,4,9\}$

Result

X: $\{0,1,2,3\}$

Y: $\{0,1,4,9\}$

- *How to implement Arc consistency?*

Arc Consistency Algorithm: AC-3

- Maintains a queue (set) of arcs
- Pop an arbitrary arc $(X_i \leftarrow X_j)$ and check D_i (the domain of X_i)
  - D_i unchanged
    - Move to next
  - D_i becomes smaller
    - Add to queue all arcs$(X_k \leftarrow X_i)$ where X_k is a neighbor of X_i
  - D_i is empty
    - Fail!

Finally, we get an CSP that is equivalent to the original CSP(with same solutions).

But now variables have smaller domains!

Arc Consistency Algorithm: 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 $queue$; propogate
  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 $revised$

Complexity of AC-3

- n: number of variables
- d: largest domain size
- c: binary constraints
- Each arc (X_k, X_i) can be inserted at most d times
  - X_k has at most *d* values to delete
- Checking consistency of one arc: $O(d^2)$

- $O(cd^3)$

Exercise – Constraints Conversion

- Turn the ternary constraint "A+B=C" into three binary constraints. (Assume finite domains.)
- Turn any ternary constraint into binary constraints.
- Eliminate unary constraints by altering domains of variables.

Conclusion:

Any CSP can be transformed into a CSP with only binary constraints.

Exercise – Constraints Conversion

$\{1, 2, 3\}$. D: var.

- Turn the ternary constraint "A+B=C" into three binary constraints. (Assume finite domains.)

$(A+B=1 \wedge C+D=1 \wedge 2C+D=2)$
$(A+B=2 \wedge C+D=1 \wedge 2C+D=3)$

- Turn any ternary constraint into binary constraints.

sum(vals).

- Eliminate unary constraints by altering domains of variables.

Conclusion:

Any CSP can be transformed into a CSP with only binary constraints.

- *Again, how to improve more?*

Forward Checking and MAC

- We can apply AC-3 *before search starts*
  - $Y = X^2$ => X: {0,1,2,3}, Y: {0,1,4,9}
- Can we do domain reductions according to arc consistency during search (**after assigning a value to a variable)**?
  - And detect inevitable failure early

Forward Checking and MAC

After assigning a value to a variable, we can

- Forward Checking
- Maintaining Arc Consistency (MAC)

Difference between Forward Checking and MAC:

<u>Which arcs to check?</u>

Forward Checking

Which arcs to check after assigning a value to variable X?

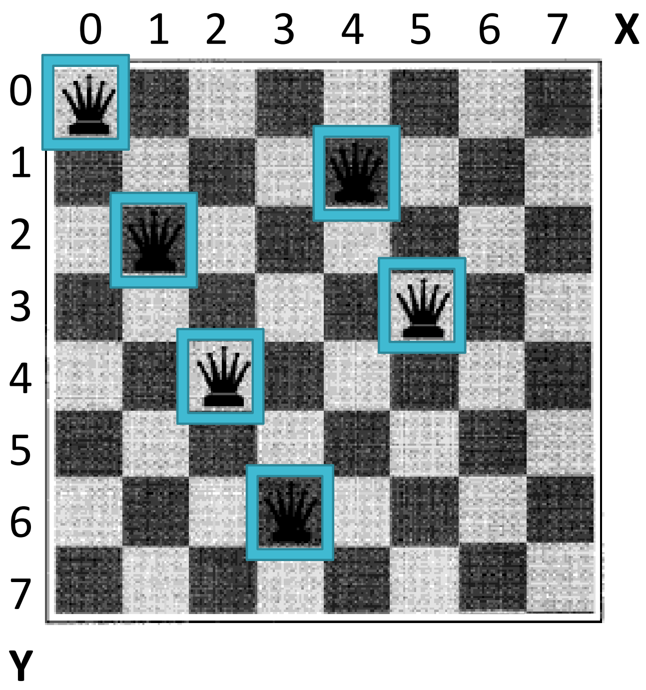
- Only the arcs of X

Forward Checking

- Keep track of remaining legal values for unassigned variables that are connected to current variable. (**Variable-level arc consistency**)
- Terminates when any variable has no legal values
  - Then backtrack!

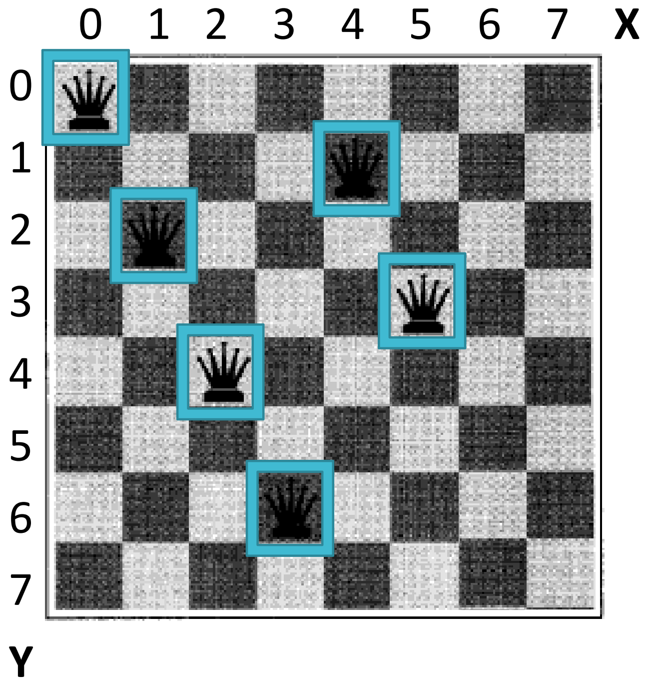
Example – 8 Queens

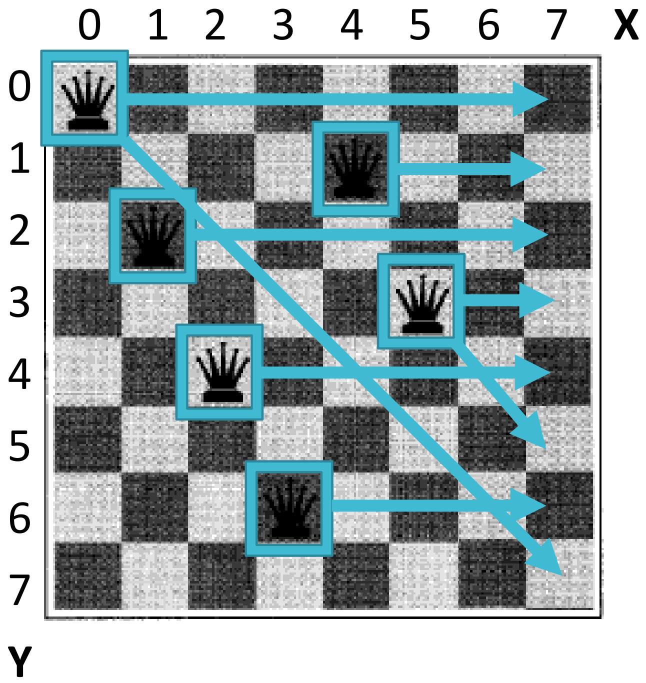
- After the sixth queen …



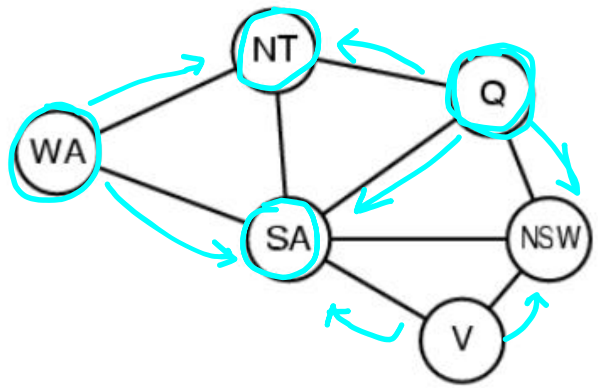
Example – 8 Queens

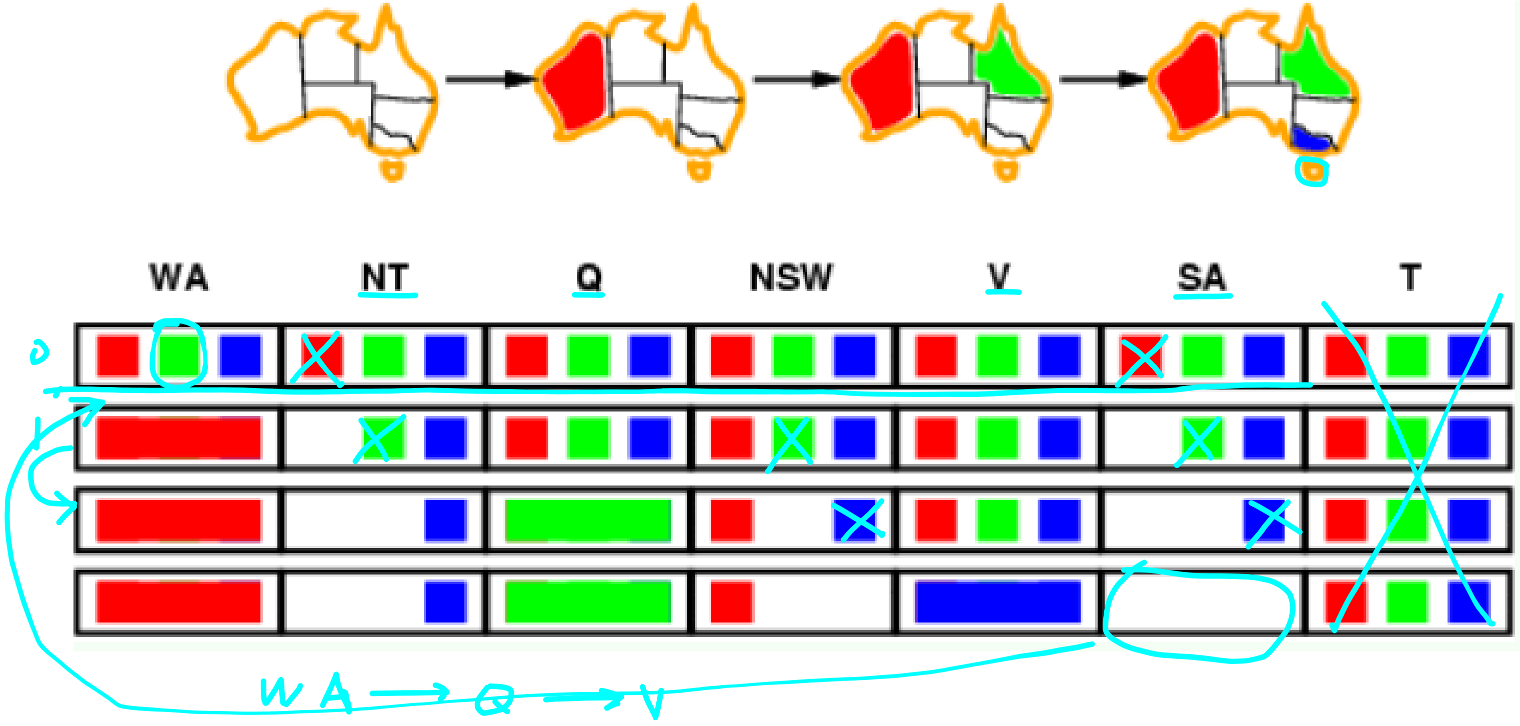
- After the sixth queen …



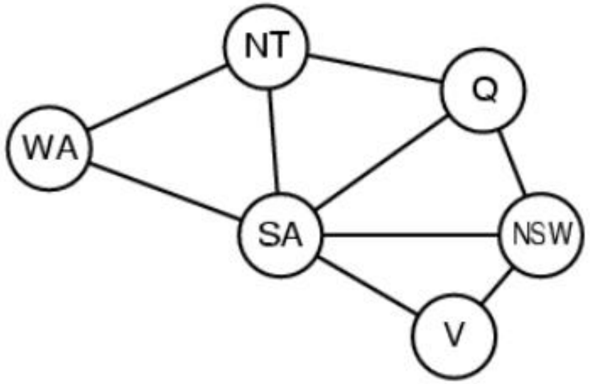


Example – Map Coloring





Example – Map Coloring



Maintaining Arc Consistency (MAC)

- **Forward checking** only makes current variable arc-consistent
- **MAC** maintains global arc consistency

Maintaining Arc Consistency (MAC)

Which arcs to check after assigning a value to variable X?

- First, push all the arcs of X
  - Same as forward checking
- Pop an arc $(Y \leftarrow X)$ and perform domain reduction
  - Y: a neighbor of X, unassigned
  - If domain size of Y reduces, push all the arcs of Y (constraint propagation)
  - If (Z <- Y) reduces the domain reduction of Z, all the arcs of Z are also pushed

$X \rightarrow Y \rightarrow Z$

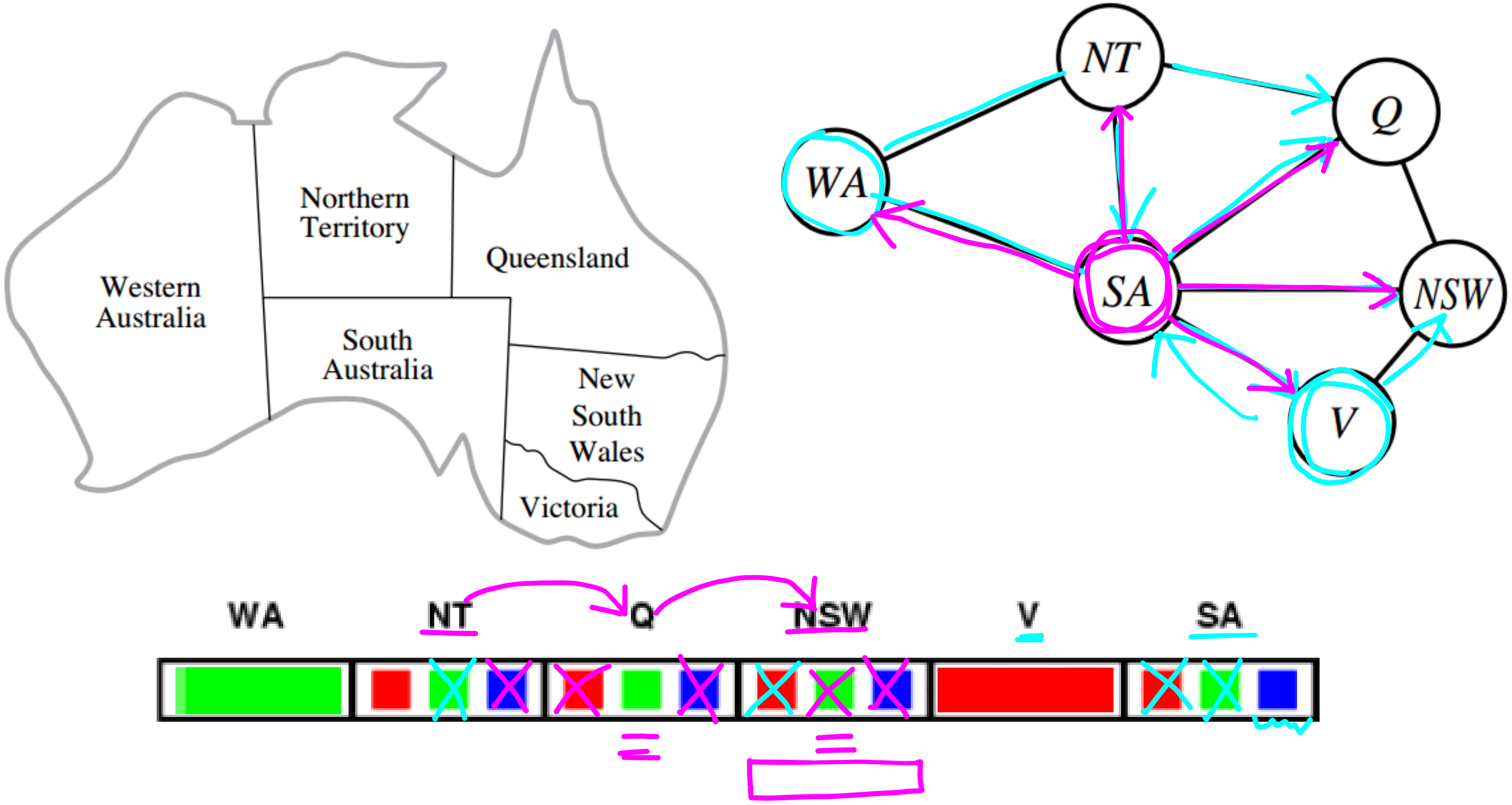
- Keeps popping and pushing until the arc queue is empty

(Similar to AC-3)

MAC is strictly more powerful than forward checking.

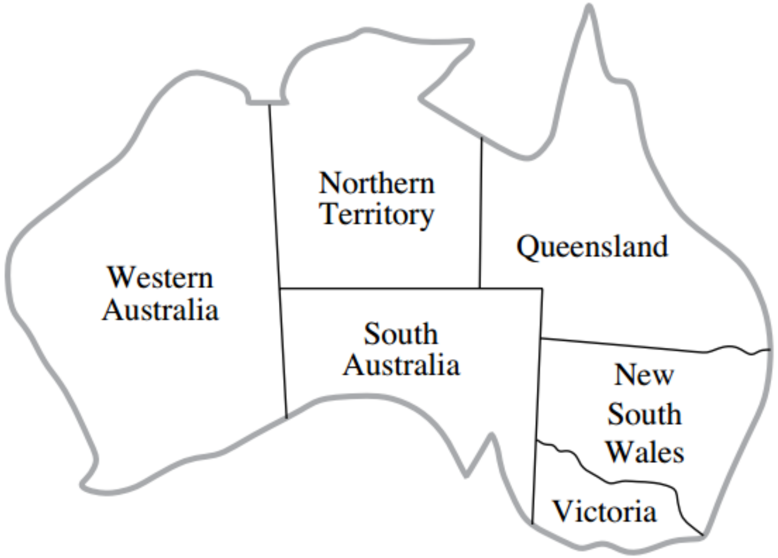
Example

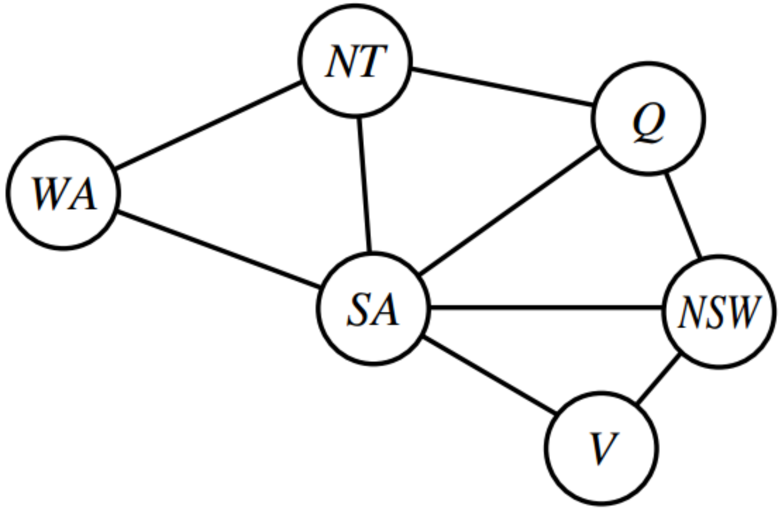
- Use the AC-3 algorithm to show that arc consistency can detect the inconsistency of the partial assignment {WA = green, V = red}

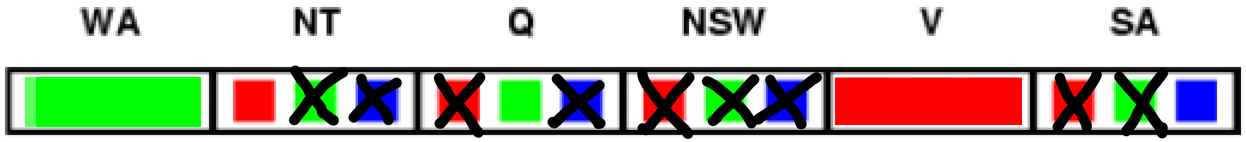


Example

- Use the AC-3 algorithm to show that arc consistency can detect the inconsistency of the partial assignment {WA = green, V = red}







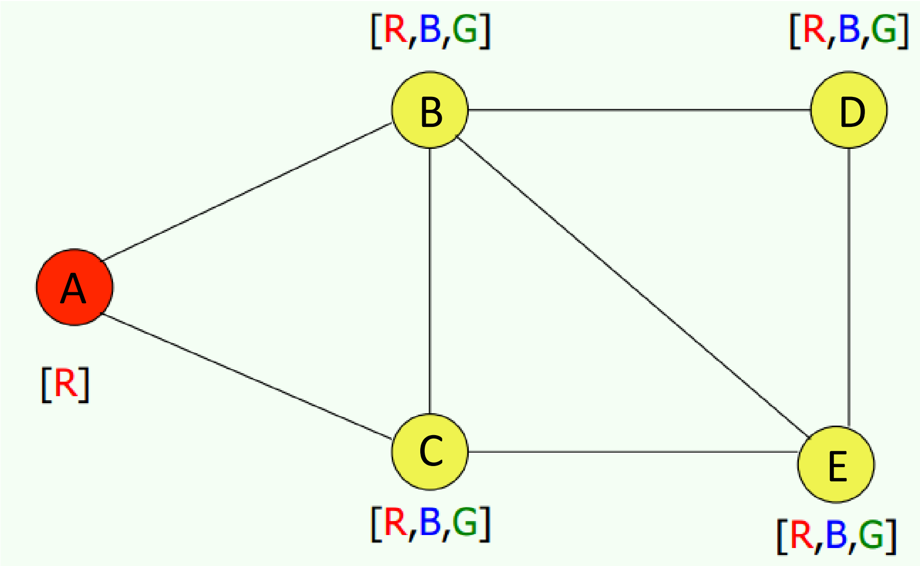
Done in one round!

Exercise – Solve CSP

- Connected variables cannot share color

Solve this CSP and explain each step

- Use all heuristics

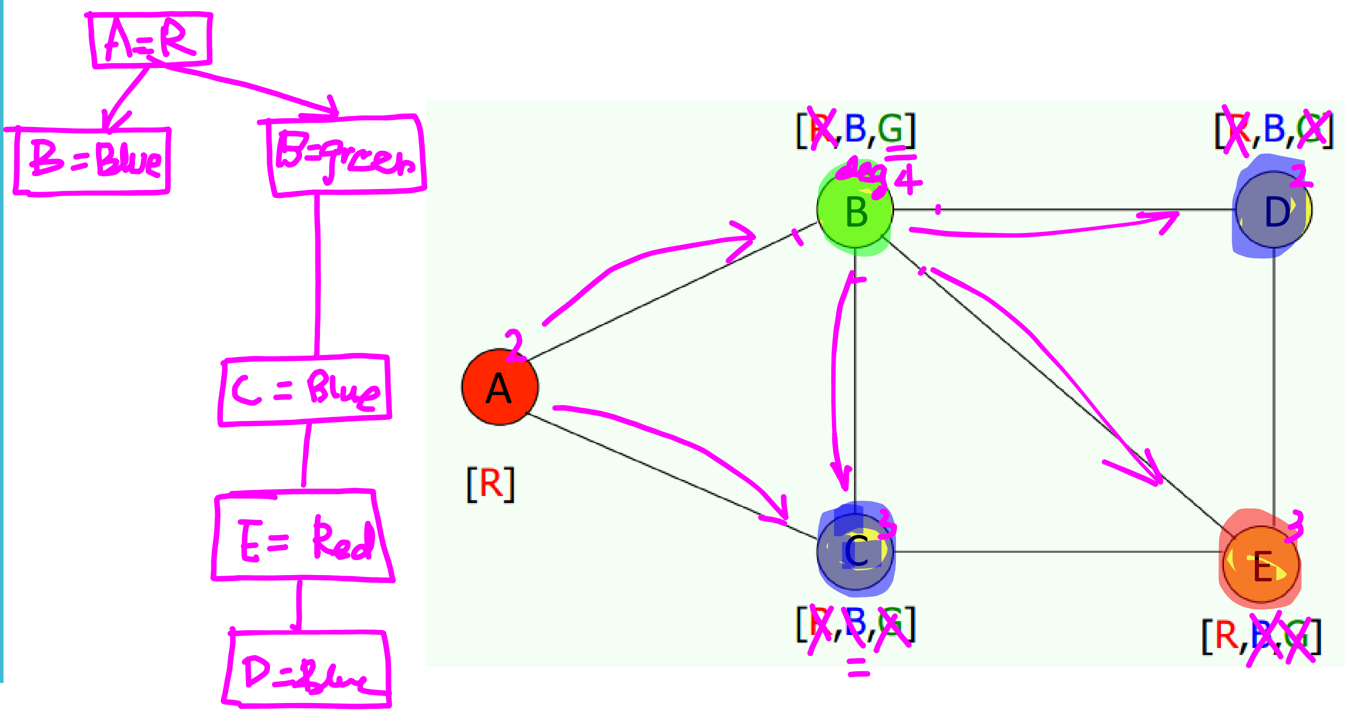


Exercise – Solve CSP

- Connected variables cannot share color

Solve this CSP and explain each step

- Use all heuristics



Demos

- https://inst.eecs.berkeley.edu/~cs188/fa19/assets/demos/csp/csp_demos.html