CMSC 471
Artificial Intelligence

Constraints

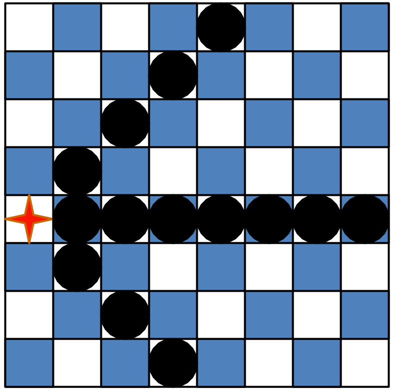
Many slides courtesy Tim Finin

General Methods of Solving CSPs

- Generate-and-Test, aka Brute Force
- Search (backtracking)
- Consistency checking
  - Forward checking
  - Arc consistency
  - Domain splitting
  - Variable Elimination
- Localized search

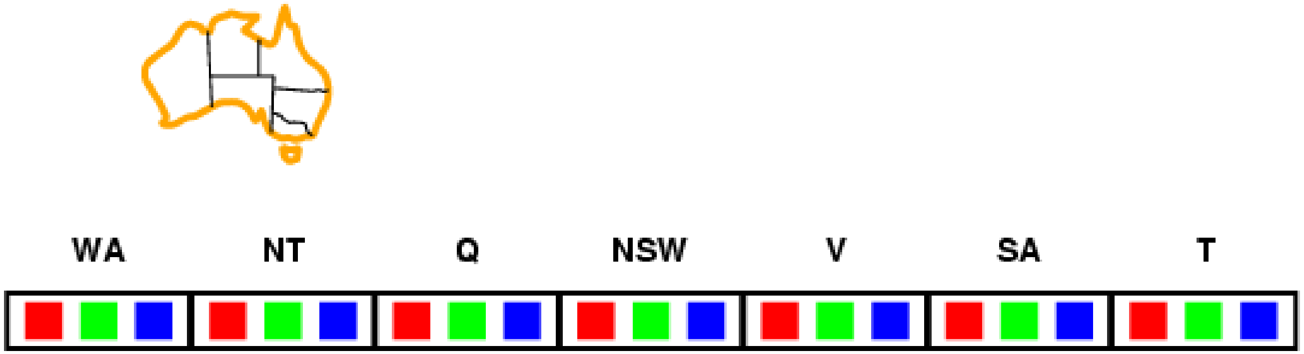
Forward Checking

After variable X is assigned to value v, examine each unassigned variable Y connected to X by a constraint and delete values from Y's domain inconsistent with v



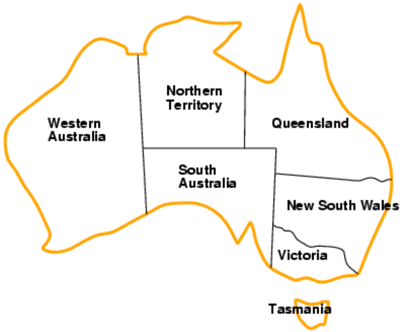
Using forward checking and backward checking roughly doubles the size of N-queens problems that can be practically solved

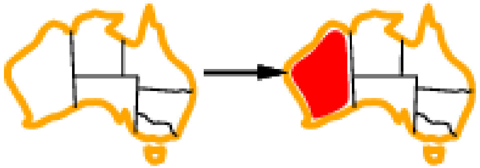
Forward checking

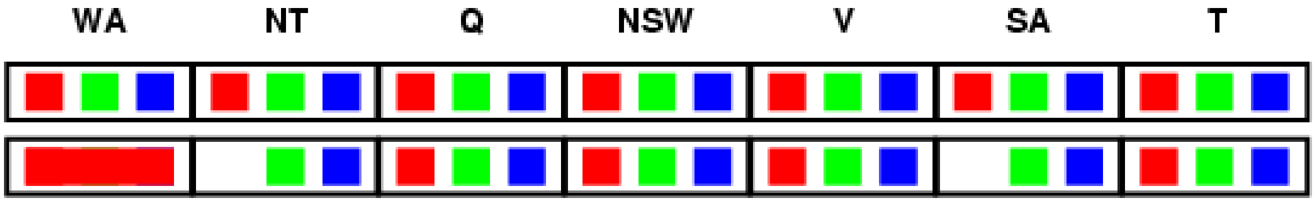


- Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values

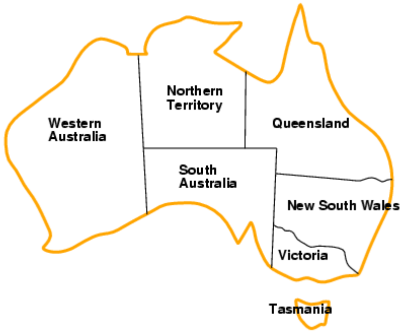
Forward checking

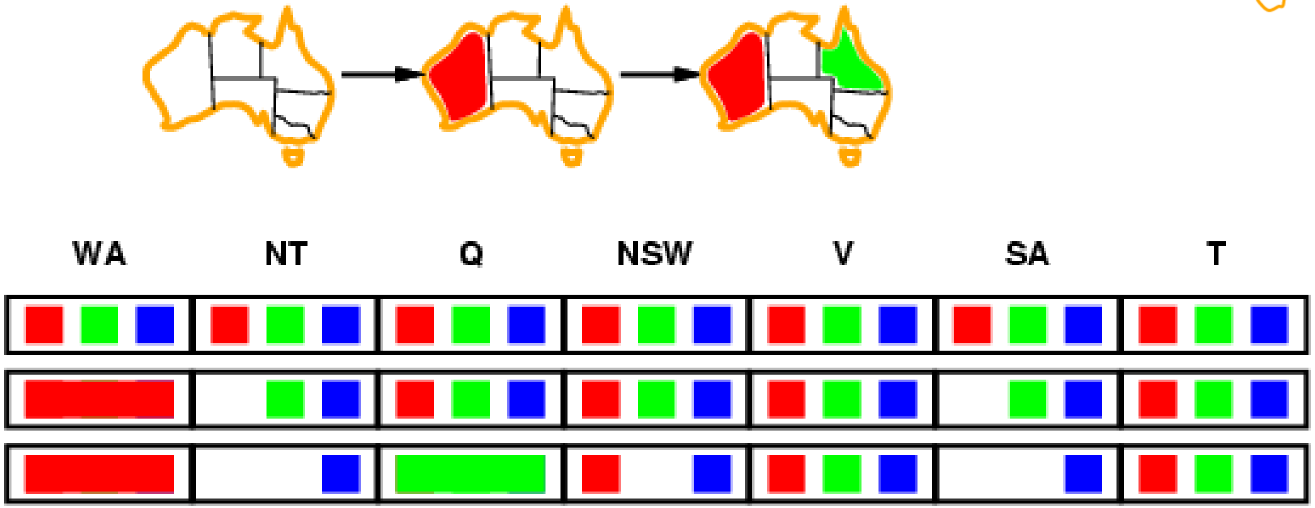




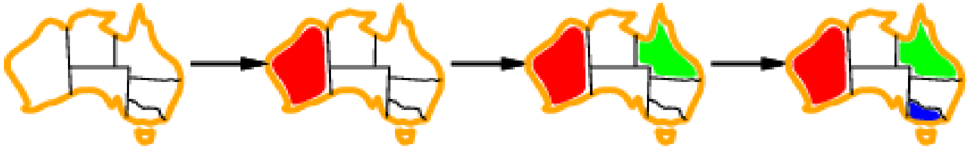


Forward checking



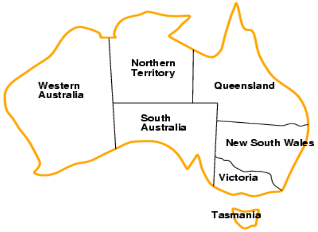


Forward checking

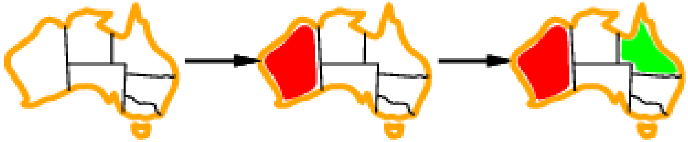


SA (South Australia) domain is empty!

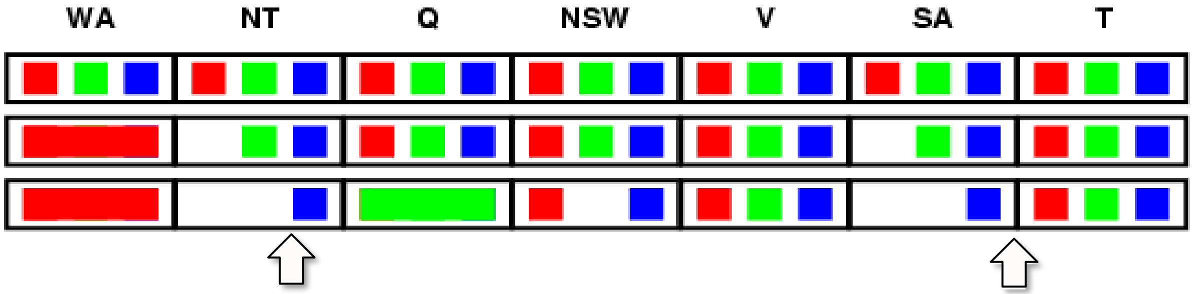
Constraint propagation



- Forward checking propagates info.
  from assigned to unassigned variables, but doesn't provide early detection for all failures
- NT and SA cannot both be blue!



Can we detect problem earlier?



Definition: Arc consistency

A constraint C_xy is arc consistent w.r.t. x

if for each value v of x there is an allowed value of y

Similarly define C_xy as arc consistent w.r.t. y

Binary CSP is arc consistent

iff every constraint C_xy is arc consistent

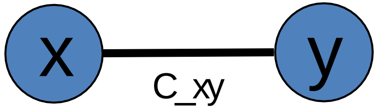
w.r.t. x as well as y

AC3 Algorithm:
Enforcing Arc Consistency

When a CSP is not arc consistent, we can make it arc consistent by using the AC3 algorithm

Arc Consistency Example 1

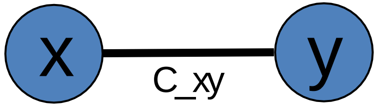
- Domains
  - $D\_x = \{1, 2, 3\}$
  - $D\_y = \{3, 4, 5, 6\}$



- Constraint
  - Note: for finite domains, we can represent a constraint as an set of legal value pairs
  - $C\_xy = \{(1,3), (1,5), (3,3), (3,6)\}$
- C_xy isn't arc consistent w.r.t. x or y. By enforcing arc consistency, we get reduced domains
  - $D'\_x = \{1, 3\}$
  - $D'\_y=\{3, 5, 6\}$

Arc Consistency Example 2

- Domains
  - D_x = {1, 2, 3}
  - D_y = {1, 2, 3}



- Constraint
  - C_xy = `lambda v1,v2: v1 < v2`
- C_xy not arc consistent w.r.t. x or y; enforcing arc consistency, we get reduced domains:
  - D'_x = {1, 2}
  - D'_y = {2, 3}

Aside: Python lambda expressions

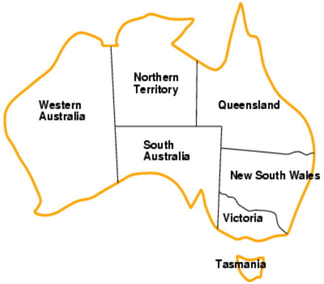
Previous slide expressed constraint between two variables as an *anonymous* Python function of two arguments

```
lambda v1,v2: v1 < v2
```

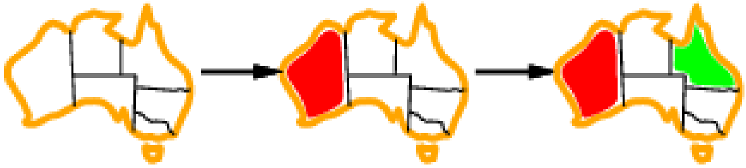
```
>>> f = lambda v1,v2: v1 < v2
>>> f
<function <lambda> at 0x10fcf21e0>
>>> f(100,200)
True
>>> f(200,100)
False
```

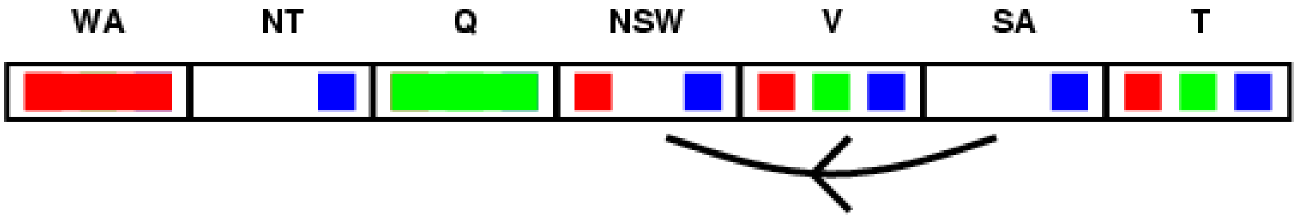
Python uses lambda after [Alonzo Church's lambda calculus from the 1930s]

Arc consistency

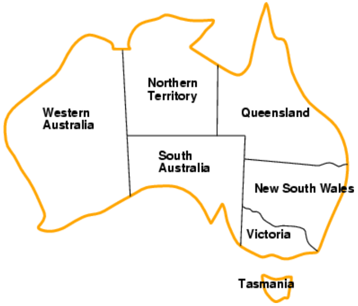


- Simplest form of propagation makes each arc consistent
- X ➔ Y is consistent iff for every value x_i of X, there is some allowed value y_j in Y

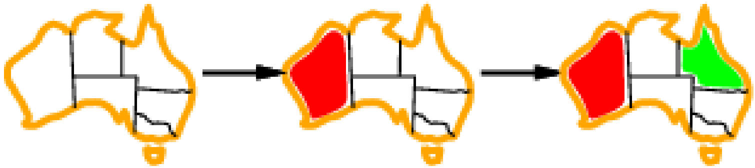


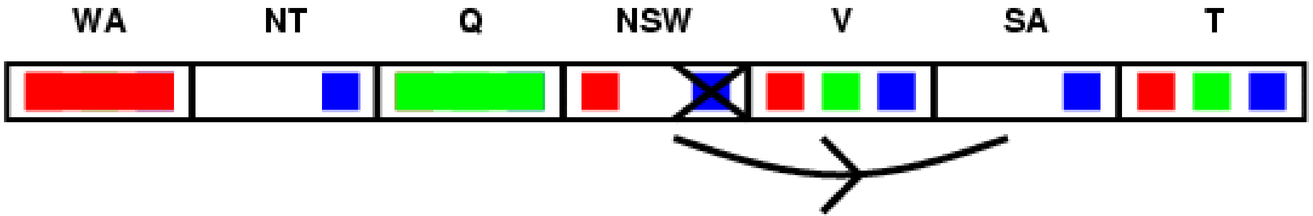


Arc consistency



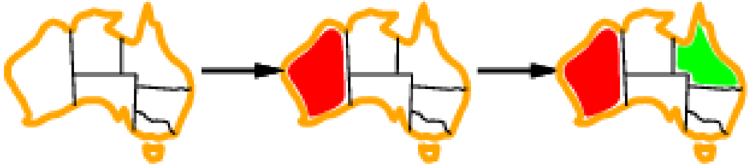
- Simplest form of propagation makes each arc consistent
- X ➔ Y is consistent iff for every value x_i of X there is some allowed value y_j in Y

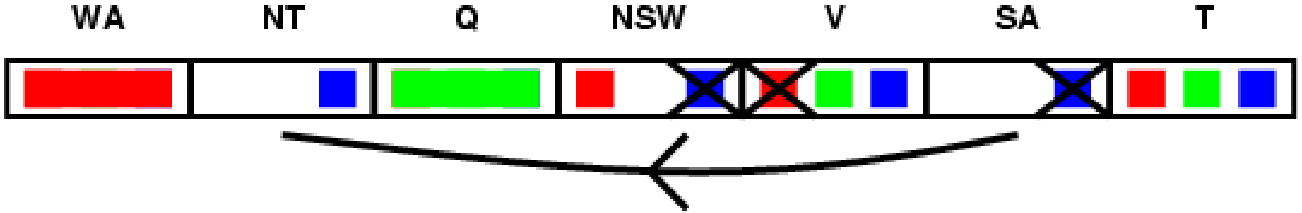




Arc consistency

- Arc consistency detects failure earlier than simple forward checking
- WA=red and Q=green is quickly recognized as a **deadend**, i.e. an impossible partial instantiation
- The arc consistency algorithm can be run as a preprocessor or after each assignment





General CP for Binary Constraints

Algorithm AC3

contradiction ← false

Q ← stack of all variables

General CP for Binary Constraints

Algorithm AC3

contradiction ← false

Q ← stack of all variables

while Q is not empty and not contradiction do

X ← UNSTACK(Q)

General CP for Binary Constraints

Algorithm AC3

contradiction ← false

Q ← stack of all variables

while Q is not empty and not contradiction do

X ← UNSTACK(Q)

For every variable Y adjacent to X do

General CP for Binary Constraints

Algorithm AC3

contradiction ← false

Q ← stack of all variables

while Q is not empty and not contradiction do

  X ← UNSTACK(Q)

  For every variable Y adjacent to X do

    If REMOVE-ARC-INCONSISTENCIES(X,Y)

General CP for Binary Constraints

Algorithm AC3

contradiction ← false

Q ← stack of all variables

while Q is not empty and not contradiction do

X ← UNSTACK(Q)

For every variable Y adjacent to X do

If REMOVE-ARC-INCONSISTENCIES(X,Y)

If domain(Y) is non-empty then STACK(Y,Q)

else return false

General CP for Binary Constraints

Algorithm AC3

contradiction ← false

Q ← stack of all variables

while Q is not empty and not contradiction do

  X ← UNSTACK(Q)

  For every variable Y adjacent to X do

    If REMOVE-ARC-INCONSISTENCIES(X,Y)

      If domain(Y) is non-empty then STACK(Y,Q)

      else return false

Q: What is the time complexity of AC3?
e = # of constraints
d = # of values per variable

General CP for Binary Constraints

- e = number of constraints (edges)
- d = number of values per variable
- Each variable is inserted into stack up to d times
- REMOVE-ARC-INCONSISTENCY takes $O(d^2)$ time
- CP takes $O(ed^3)$ time

A Poole & Mackworth Example (Fig 4.4)

Setup: 5 variables (A, B, C, D, E) each with domain {1,2,3,4}

Constraints:

$A \neq B$

$A = D$

$A \geq E$

$D \neq B$

$C \neq B$

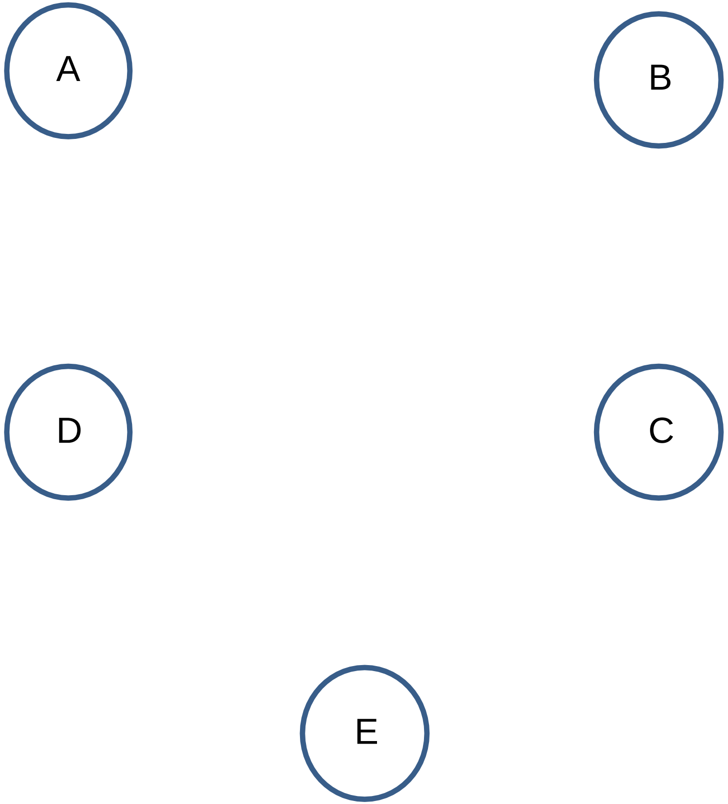
$E < B$

$C < D$

$E < C$

$E < D$

A Poole & Mackworth Example (Fig 4.4)



Setup: 5 variables (A, B, C, D, E) each with domain {1,2,3,4}

Constraints:

$A \neq B$

$A = D$

$A \geq E$

$D \neq B$

$C \neq B$

$E < B$

$C < D$

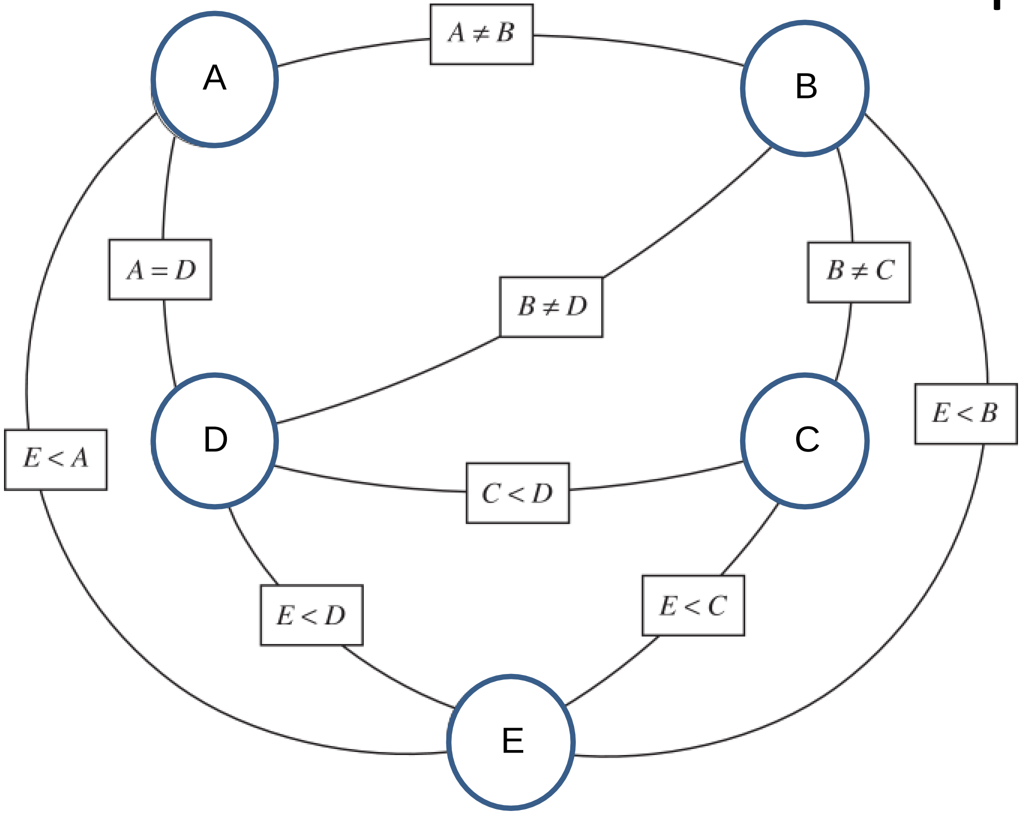
$E < C$

$E < D$

$B \neq 3$

$C \neq 2$

A Poole & Mackworth Example (Fig 4.4)

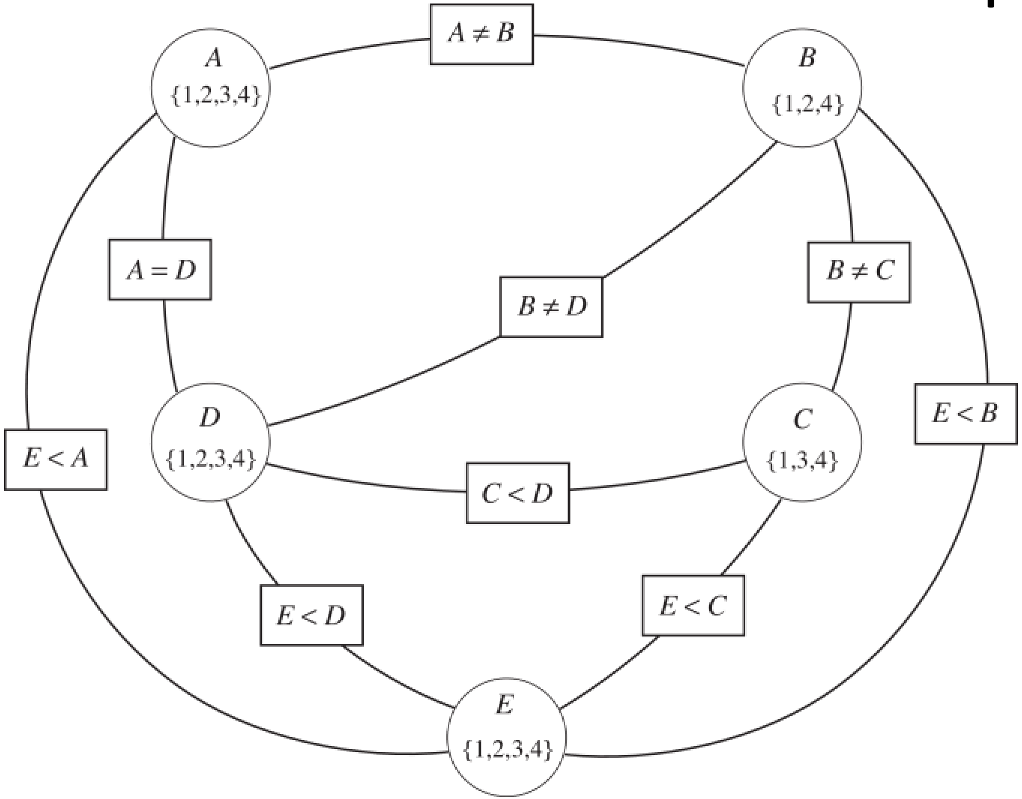


Setup: 5 variables (A, B, C, D, E) each with domain {1,2,3,4}

Constraints:

$A \neq B$
$A = D$
$A \geq E$
$D \neq B$
$C \neq B$
$E < B$
$C < D$
$E < C$
$E < D$
$B \neq 3$
$C \neq 2$

A Poole & Mackworth Example (Fig 4.4)



Setup: 5 variables (A, B, C, D, E) each with domain {1,2,3,4}

Constraints:

$A \neq B$
$A = D$
$A \geq E$
$D \neq B$
$C \neq B$
$E < B$
$C < D$
$E < C$
$E < D$
$B \neq 3$
$C \neq 2$

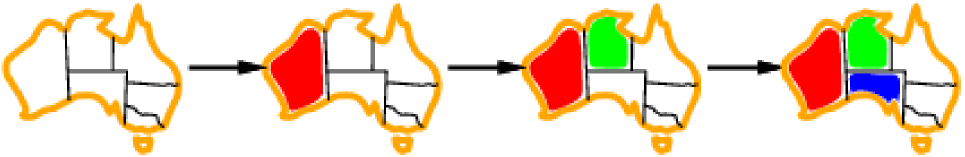
Q: Is this arc consistent?

Improving backtracking efficiency

- Some standard techniques to improve the efficiency of backtracking
  - Can we detect inevitable failure early?
  - Which variable should be assigned next?
  - In what order should its values be tried?
- Combining constraint propagation with these heuristics makes 1000-queen puzzles feasible

Most constrained variable

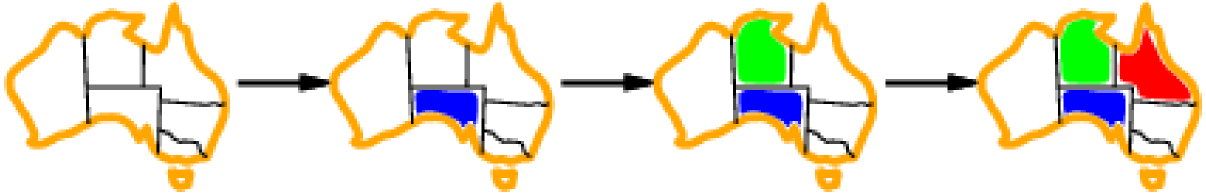
- Most constrained variable:
  choose the variable with the fewest legal values



- a.k.a. minimum remaining values (MRV) heuristic
- After assigning value to WA, both NT and SA have only two values in their domains
  – choose one of them rather than Q, NSW, V or T

Most constraining variable

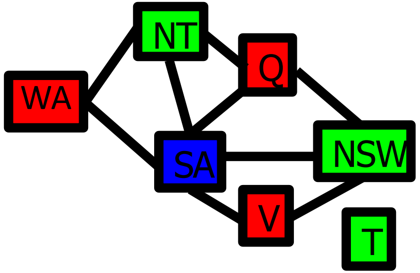
- Tie-breaker among most constrained variables
- Choose variable involved in largest # of constraints on remaining variables

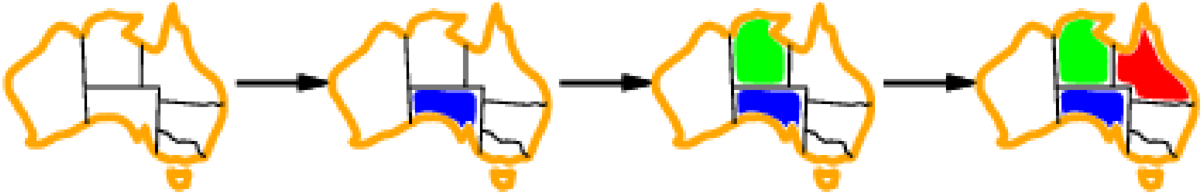


- After assigning SA to be blue, WA, NT, Q, NSW and V all have just two values left.
- WA and V have only one constraint on remaining variables and T none, so choose one of NT, Q & NSW

Most constraining variable

- Tie-breaker among most constrained variables
- Choose variable involved in largest # of constraints on remaining variables

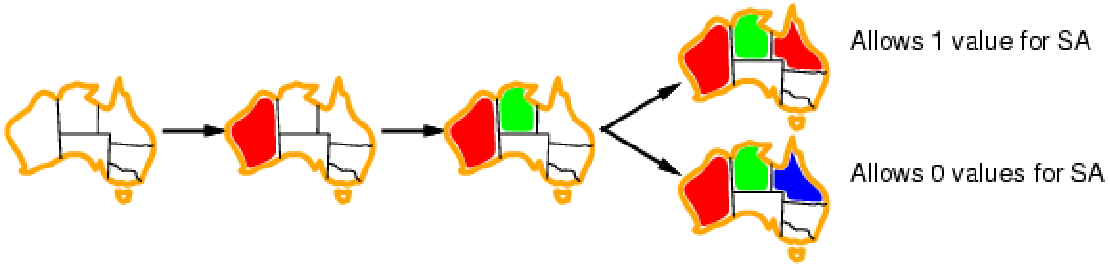




- After assigning SA to be blue, WA, NT, Q, NSW and V all have just two values left.
- WA and V have only one constraint on remaining variables and T none, so choose one of NT, Q & NSW

Least constraining value

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



- Combining these heuristics makes 1000 queens feasible
- What's an intuitive explanation for this?

Domain Splitting

Also called “case analysis”

Split a variable’s domain into disjoint subsets, and consider them each separately

Domain Splitting

Also called "case analysis"

Split a variable's domain into disjoint subsets, and consider them each separately

- If $\mathrm{dom}(X_i) = \{a_1, \ldots, a_M\}$, then for each possible setting of $X_i = a_m$, find an assignment to all other variables that satisfy the constraints
- This is solved a **reduced** problem

Domain Splitting

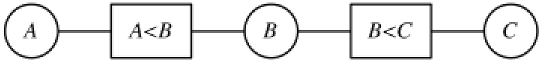
Also called "case analysis"

Split a variable's domain into disjoint subsets, and consider them each separately

- If $\text{dom}(X_i) = \{a_1, \ldots, a_M\}$, then for each possible setting of $X_i = a_m$, find an assignment to all other variables that satisfy the constraints
- This is solved a **reduced** problem

Q: how does this relate to search?

Domain Splitting Example



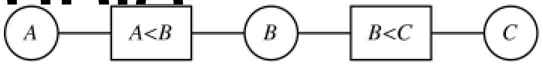
Setup: 3 variables (A, B, C) each with domain {1,2,3,4}

Constraints:
$A < B$
$B < C$

	A	B	C
Original Domains:	{1,2,3,4}	{1,2,3,4}	{1,2,3,4}
After arc consistency:	{1,2}	{2,3}	{3,4}
Domain Splitting:			
B=2	{1,~~2~~}	{2}	{3,4}
B=3	{1,2}	{3}	{~~3~~, 4}

Domain Splitting

Example



Setup: 3 variables (A, B, C) each with domain {1,2,3,4}

Constraints:
$A < B$
$B < C$

	A	B	C	
Original Domains:	{1,2,3,4}	{1,2,3,4}	{1,2,3,4}	
After arc consistency:	{1,2}	{2,3}	{3,4}	
Domain Splitting:				
B=2, C=3	{1,~~2~~}	{2}	{3,~~4~~}	✓
B=2, C=4	{1,~~2~~}	{2}	{~~3~~,4}	✓
B=3, A=1	{1,~~2~~}	{3}	{~~3~~,4}	✓
B=3, A=2	{~~1~~,2}	{3}	{~~3~~,4}	✓

Variable Elimination

- Simplify the network by incrementally removing variables
  - Remove a variable, and create a new constraint on the remaining variables to account for its removal

Variable Elimination Algorithm

```
procedure VE_CSP(Vs, Cs)
    Inputs
        Vs: a set of variables
        Cs: a set of constraints on Vs
    Output
        a relation containing all of the consistent variable assignments
    if Vs contains just one element then
        return the join of all the relations in Cs
    else
```

Variable Elimination Algorithm

```
procedure VE_CSP(Vs, Cs)
    Inputs
        Vs: a set of variables
        Cs: a set of constraints on Vs
    Output
        a relation containing all of the consistent variable assignments
    if Vs contains just one element then
        return the join of all the relations in Cs
    else
        select variable Xs to eliminate
```

Variable Elimination Algorithm

```
procedure VE_CSP(Vs, Cs)
    Inputs
        Vs: a set of variables
        Cs: a set of constraints on Vs
    Output
        a relation containing all of the consistent variable assignments
    if Vs contains just one element then
        return the join of all the relations in Cs
    else
        select variable X to eliminate
        Vs' := Vs \ {X}
```

Remove X from the set of variables

```
        S := VE_CSP(Vs', (Cs \ Cs_X) ∪ {R'})
        return R ⋈ S
```

Variable Elimination Algorithm

procedure *VE_CSP*(Vs, Cs)

  Inputs

    Vs: a set of variables

    Cs: a set of constraints on Vs

  Output

    a relation containing all of the consistent variable assignments

  if Vs contains just one element **then**

    return the join of all the relations in Cs

  else

    select variable Xs to eliminate

    $Vs' := Vs \setminus \{X\}$

    $Cs_X := \{T \in Cs : T \text{ involves } X\}$

Identify the constraints involving X that need to be reformulated/accounted for

Variable Elimination Algorithm

procedure *VE_CSP*(Vs, Cs)
    Inputs
        Vs: a set of variables
        Cs: a set of constraints on Vs
    Output
        a relation containing all of the consistent variable assignments
    if Vs contains just one element **then**
        return the join of all the relations in Cs
    else
        select variable $X \in Vs$ to eliminate
        $Vs' := Vs \setminus \{X\}$
        $Cs_X := \{T \in Cs : T \text{ involves } X\}$
        let R be the join of all of the constraints in Cs_X
        let R' be R projected onto the variables other than X

Based on individual assignments to X, identify the set of allowed assignments to other variables in those constraints' scopes

Variable Elimination Algorithm

procedure *VE_CSP*(Vs, Cs)
 Inputs
  Vs: a set of variables
  Cs: a set of constraints on Vs
 Output
  a relation containing all of the consistent variable assignments
 if Vs contains just one element **then**
  return the join of all the relations in Cs
 else
   select variable Xs to eliminate
   $\mathrm{Vs}' := \mathrm{Vs} \setminus \{X\}$
   $\mathrm{Cs}_X := \{T \in \mathrm{Cs} : T \text{ involves } X\}$
   let R be the join of all of the constraints in Cs_X
   let R' be R projected onto the variables other than X
   $S := \mathrm{VE\_CSP}\,(\mathrm{Vs}', (\mathrm{Cs} \setminus \mathrm{Cs}_X) \cup \{R'\})$
   return $R \bowtie S$

Example

Setup: 3 variables (A, B, C) each with domain {1,2,3,4}

Constraints:
$A < B$
$B < C$

A	B
1	2
1	3
1	4
2	3
2	4
3	4

B	C
1	2
1	3
1	4
2	3
2	4
3	4

Eliminate B

Example

Setup: 3 variables (A, B, C) each with domain {1,2,3,4}

Initial Constraints:

$A < B$

$B < C$

A	B
1	2
1	3
1	4
2	3
2	4
3	4

B	C
1	2
1	3
1	4
2	3
2	4
3	4

Identify possible, legal combinations. Red rows are not feasible.

Example

Setup: 3 variables (A, B, C) each with domain {1,2,3,4}

Initial Constraints:
$A < B$
$B < C$

A	B
1	2
1	3
1	4
2	3
2	4
3	4

B	C
1	2
1	3
1	4
2	3
2	4
3	4

A	B	C
1	2	3
1	2	4
1	3	4
2	3	4

Reformulate constraints/constraint table…

Example

Setup: 3 variables (A, B, C) each with domain {1,2,3,4}

Initial Constraints:
$A < B$
$B < C$

A	C
1	3
1	4
2	4

Reformulate constraints/constraint table… into one that doesn't involve B, and solve the simpler problem

Characteristics of Variable Elimination

- Depends entirely on the tree-width
- Finding a good elimination order is NP-hard (!!!)
  - Heuristic 1: min-factor: select the variable that results in the smallest relation
  - Heuristic 2: minimum fill: select the variable that adds the fewest arcs to the resulting graph (don't make the graph more complicated)