Constraint Satisfaction, Part 2

Lecture 12

Chapter 6, Sections 6.1-6.3

Jim Rehg

College of Computing

Georgia Tech

February 8, 2016

Ways to Improve Backtracking Search

What variable to assign next?

Choose variable with fewest values left → fail first

What order to use for values of a variable?

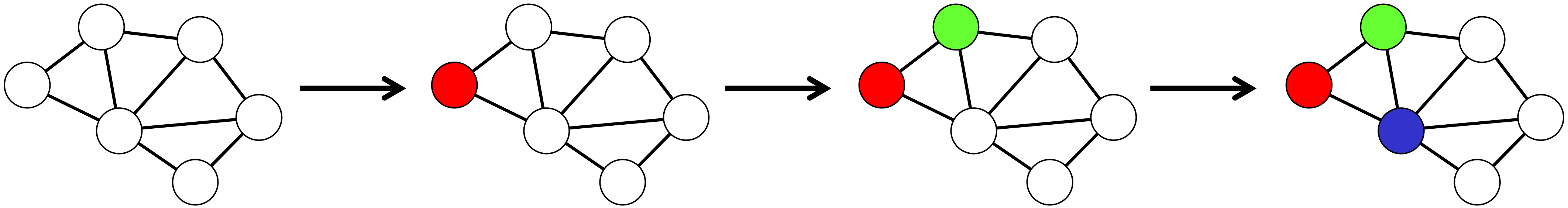
Choose val that leaves most options for remaining vars → fail last

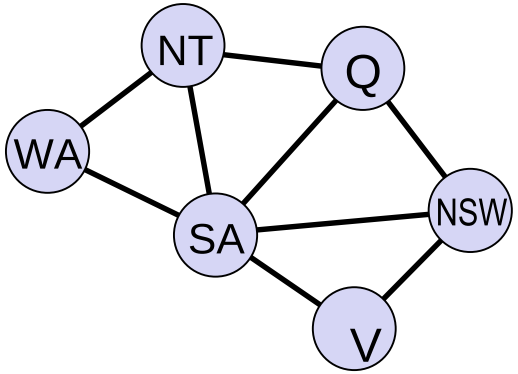
What inferences can be made from an assignment?

How can we take advantage of problem structure?

Minimum Remaining Values (MRV)

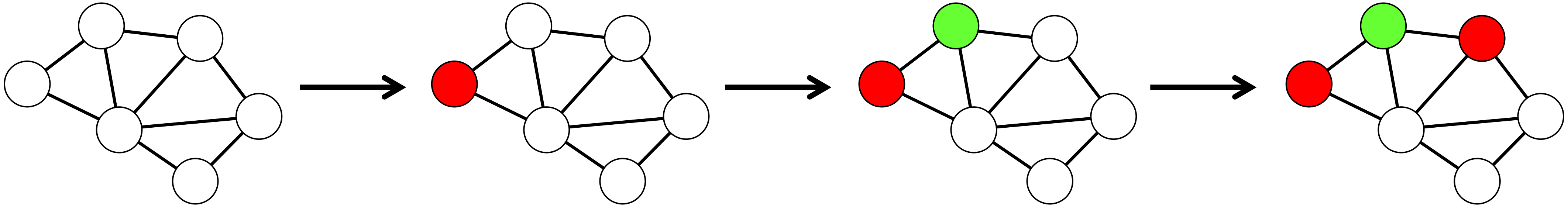
Choose the variable with the fewest remaining legal values

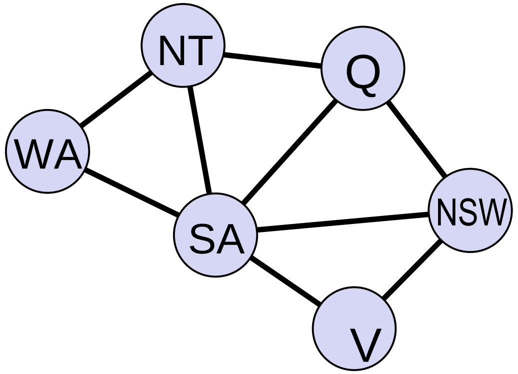




Least Constraining Value

Choose the value which removes the least number of potential values from the remaining unassigned variables





Choosing Q=red leaves
1 value for SA,
while Q=blue would leave
0 values

Ways to Improve Backtracking Search

What variable to assign next?

Choose variable with fewest values left → *fail first*

What order to use for values of a variable?

Choose val that leaves most options for remaining vars → fail last

What inferences can be made from an assignment?

Constraint propagation to reduce the legal values for variables

How can we take advantage of problem structure?

Checking Consistency

Node consistency:

Ensure values in domain of variable X satisfy unary constraints

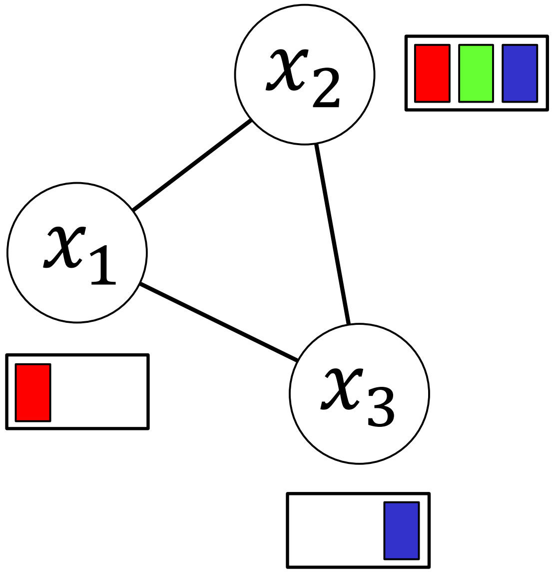
Arc Consistency (AC):

Ensure each value in domain D_i of X_i satisfies binary constraints

Binary constraint $C_{i,j}$ for vars $X_i \leftrightarrow X_j$

If X_i is AC with X_j then:

for each $x \in D_i$ there exists $y \in D_j$ such that $(x, y) \in C_{i,j}$



x_1 is AC with x_2 and x_3

x_3 is AC with x_1 and x_2

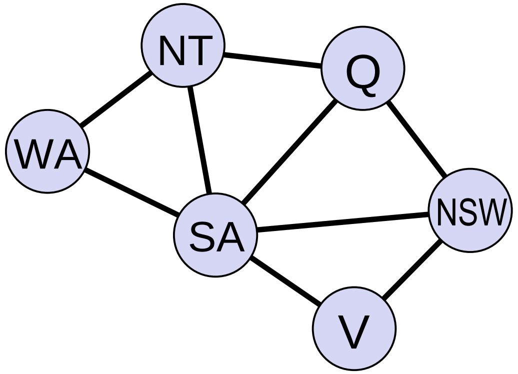
x_2 is **not** AC with x_1 and x_3

Forward Checking

Whenever a variable X is assigned, enforce arc consistency for all links $Y \rightarrow X$ for unassigned variables Y

But we could have identified it here!

No valid assignment!



Recursively Enforcing Arc Consistency

Alan Mackworth
Univ. of British Columbia

function AC-3(*csp*)
    while *queue* is not empty **do**
        $(X_i, X_j) \leftarrow$ *queue*.POP()
        revised ← *false*
        for each x **in** D_i **do**
            if there is no value $y \in D_j$ satisfying $C_{i,j}$ **then**
                remove x from D_i
                revised ← *true*
        if *revised* **then**
            if size(D_i) = 0 **then** return *false*
            for each X_k neighbor of X_i (excluding X_j) **do** queue.PUSH((X_k, X_i))
    return *true*

Goes beyond forward checking by propagating constraints

AC-3 Map Coloring Example

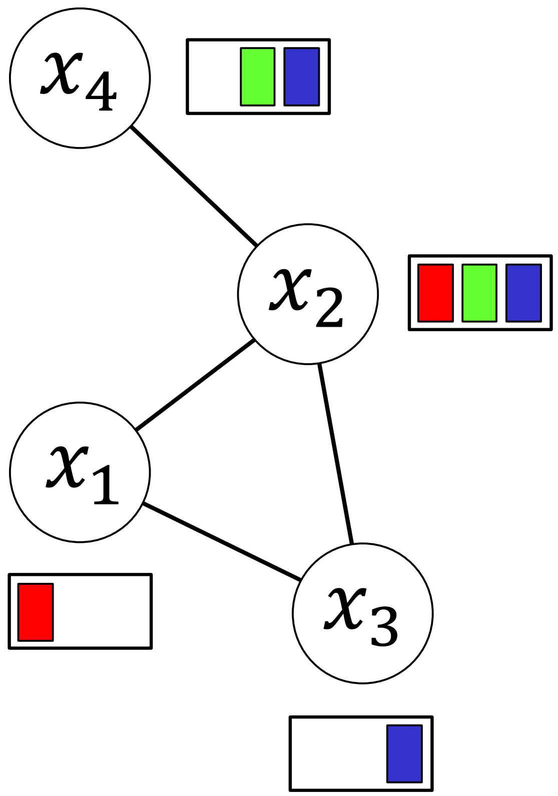
Queue

$X_1 \rightarrow X_2$

$X_2 \rightarrow X_1$

$X_2 \rightarrow X_3$

⋮



AC-3 Map Coloring Example

Queue

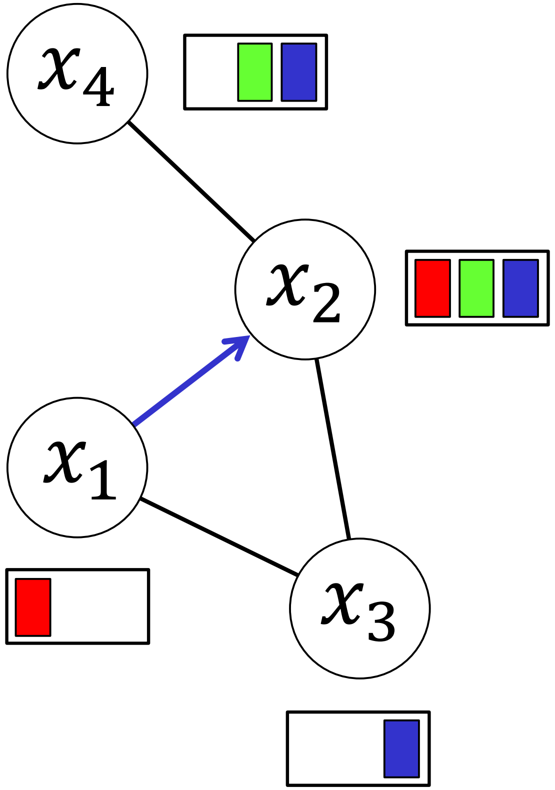
$X_1 \rightarrow X_2$

$X_2 \rightarrow X_1$

$X_2 \rightarrow X_3$

$\vdots$

When you set $x_1 = red$ you can choose $x_2 = blue$ to satisfy constraint



AC-3 Map Coloring Example

<u>Queue</u>

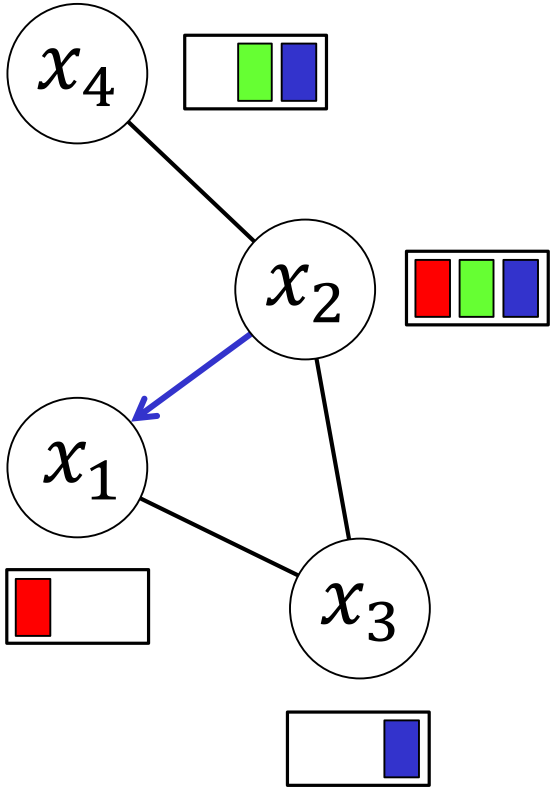
~~$X_1 \to X_2$~~

$X_2 \to X_1$

$X_2 \to X_3$

$\vdots$

When you set $x_2 = red$
there is no choice for x_1
that satisfies the constraint



AC-3 Map Coloring Example

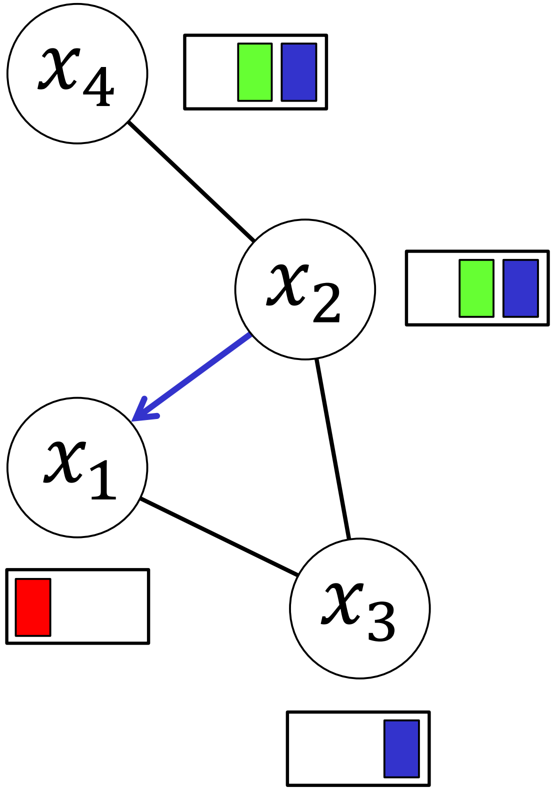
<u>Queue</u>

$\cancel{X_1 \rightarrow X_2}$

$X_2 \rightarrow X_1$ remove

$X_2 \rightarrow X_3$

⋮



AC-3 Map Coloring Example

Queue

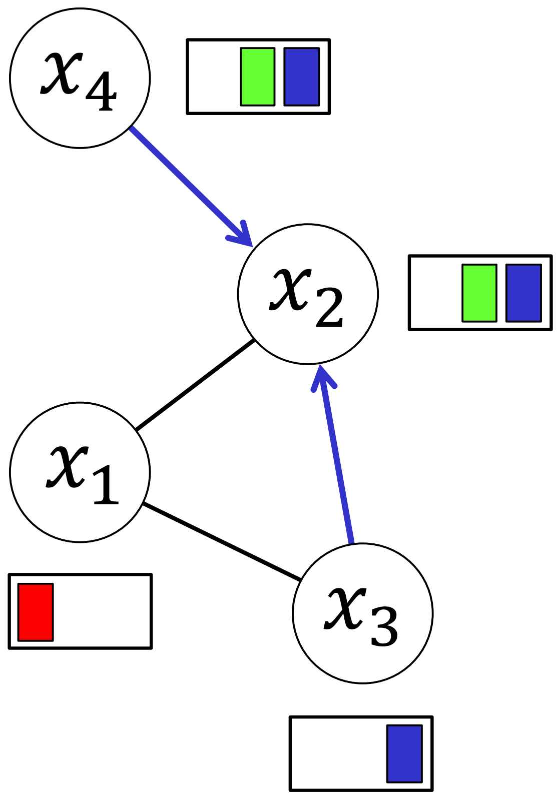
$X_3 \to X_2$

$X_4 \to X_2$

D_2 changed, check neighbors

$X_2 \to X_3$

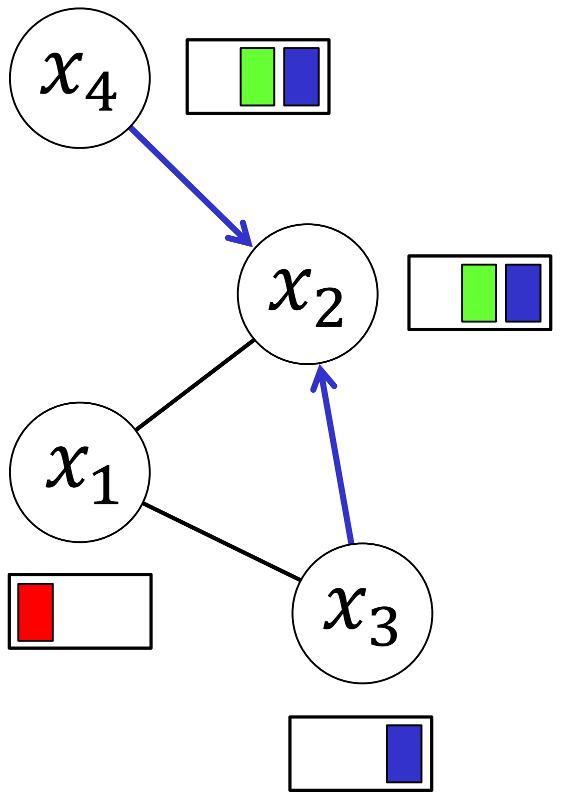
$\vdots$



AC-3 Map Coloring Example

Queue

$\cancel{X_3 \rightarrow X_2}$
$\cancel{X_4 \rightarrow X_2}$ } D_2 changed, check neighbors
$X_2 \rightarrow X_3$
⋮

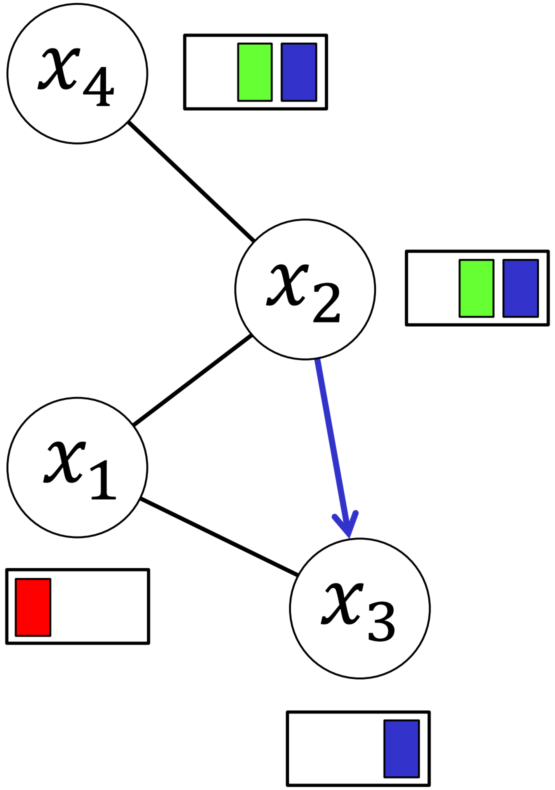


AC-3 Map Coloring Example

<u>Queue</u>

$X_2 \rightarrow X_3$

⋮

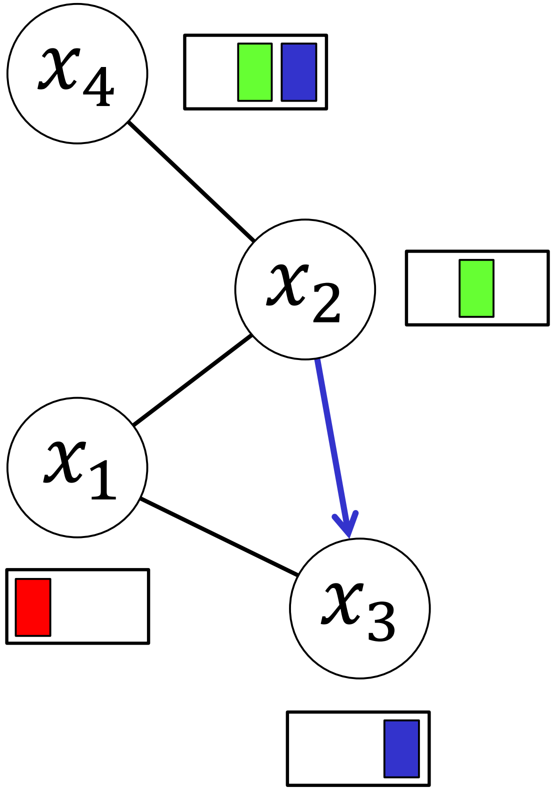


AC-3 Map Coloring Example

<u>Queue</u>

$X_2 \rightarrow X_3$ remove

⋮



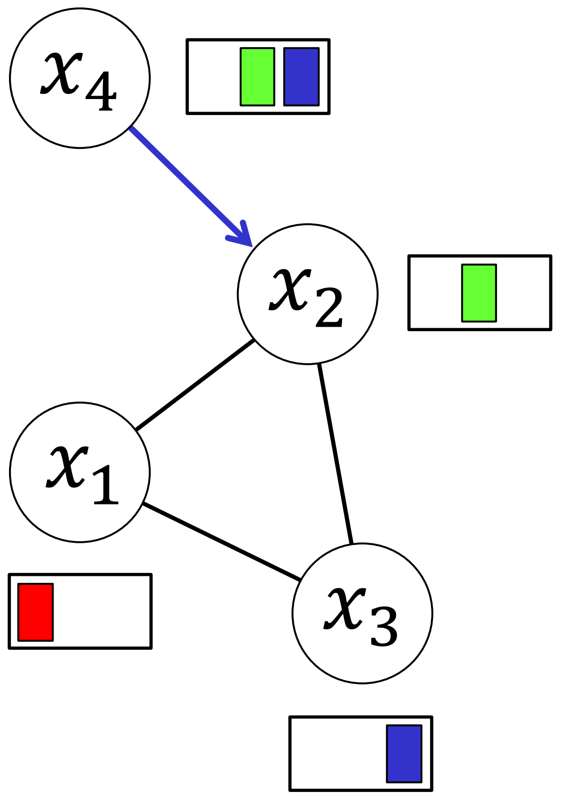
AC-3 Map Coloring Example

Queue

$X_4 \rightarrow X_2$

$X_1 \rightarrow X_2$

$\vdots$



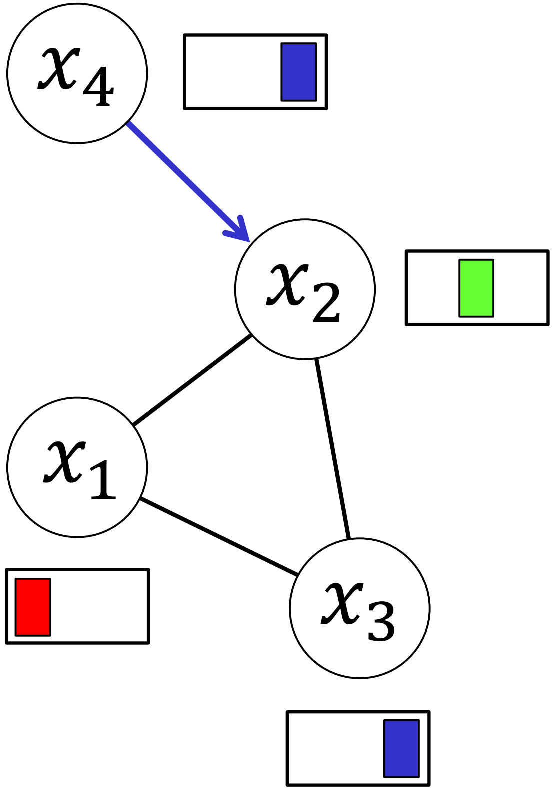
AC-3 Map Coloring Example

Queue

$X_4 \rightarrow X_2$ remove

$X_1 \rightarrow X_2$

$\vdots$



AC-3 Sudoku Example

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

Domain: {1,2,3,4,5,6,7,8,9}

Propagate box constraint

Domain: {1,2,3,4,5, 7,8,9}

Propagate row constraint

Domain: { 2,3, 7,8 }

Propagate column constraint

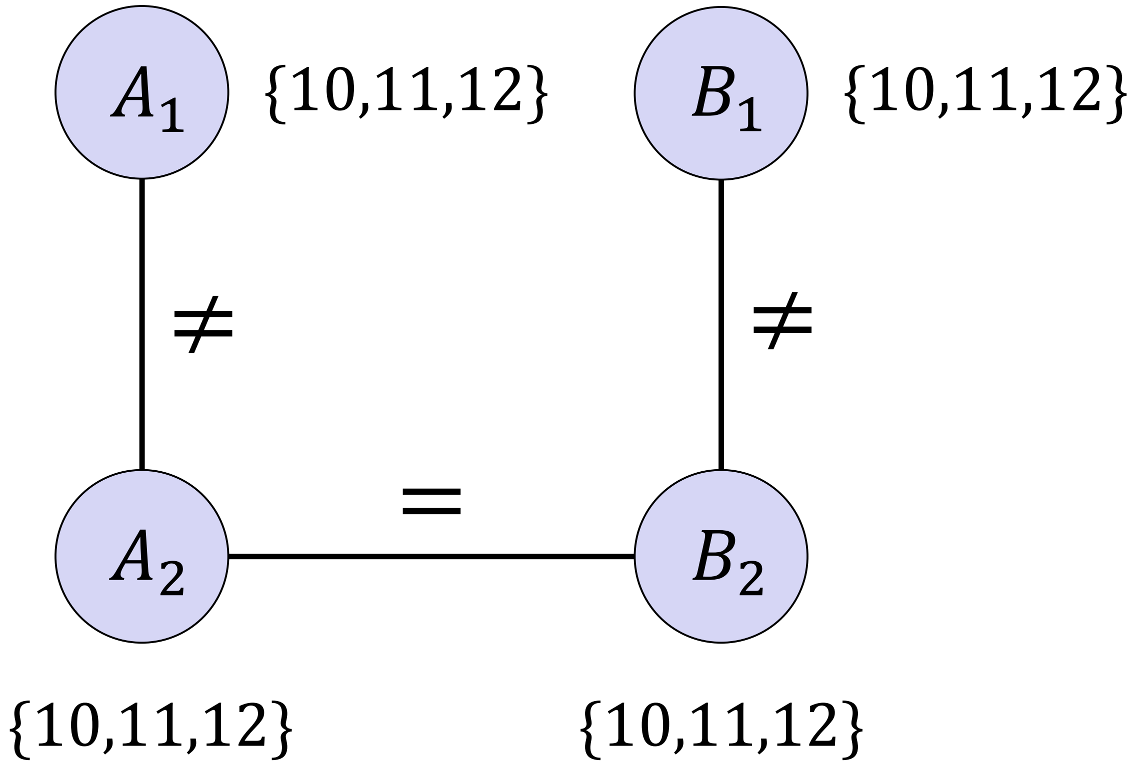
Domain: { 2,3, 7 }

Constraint Propagation Example

Schedule 2 classes for 2 profs into 3 possible time slots

Interleaving search and AC-3:

- Expand A_1 and select value
- Call AC-3 on $A_2 \rightarrow A_1$
- Expand B_2 and select value
- Call AC-3 on $A_2 \rightarrow B_2$ and $B_1 \rightarrow B_2$

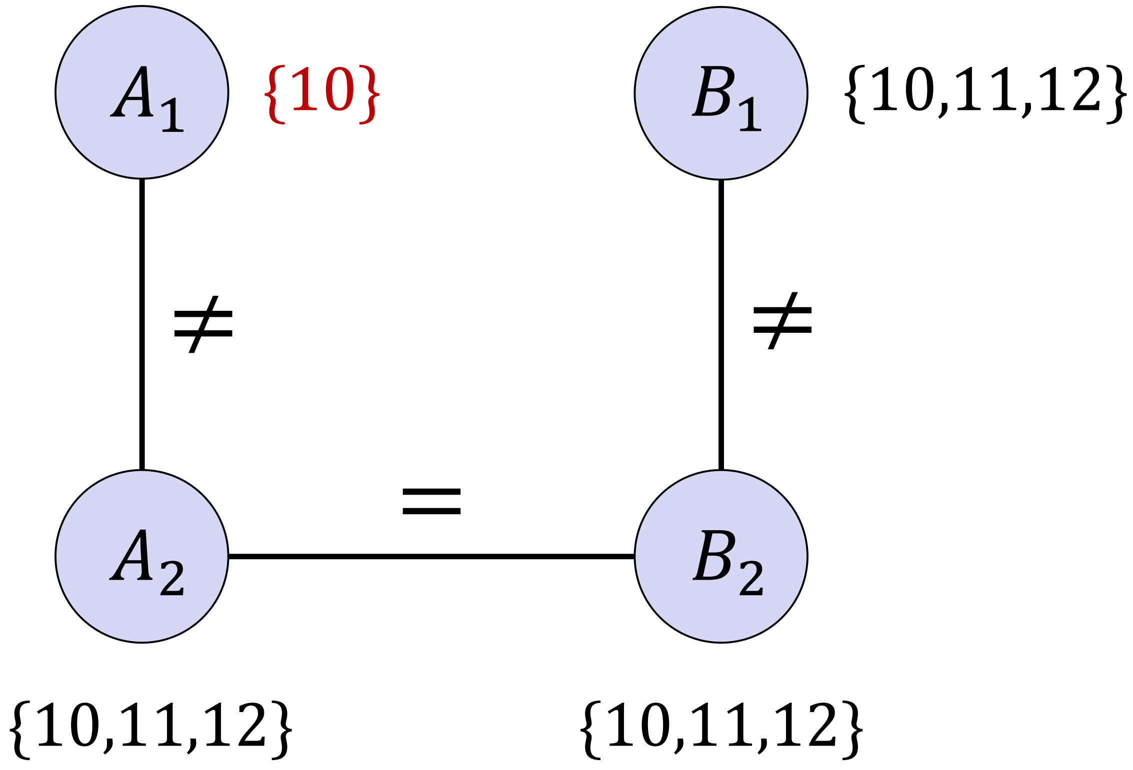


Constraint Propagation Example

Schedule 2 classes for 2 profs into 3 possible time slots

Interleaving search and AC-3:

- Expand A_1 and select value
- Call AC-3 on $A_2 \rightarrow A_1$
- Expand B_2 and select value
- Call AC-3 on $A_2 \rightarrow B_2$ and $B_1 \rightarrow B_2$



Constraint Propagation Example

Schedule 2 classes for 2 profs into 3 possible time slots

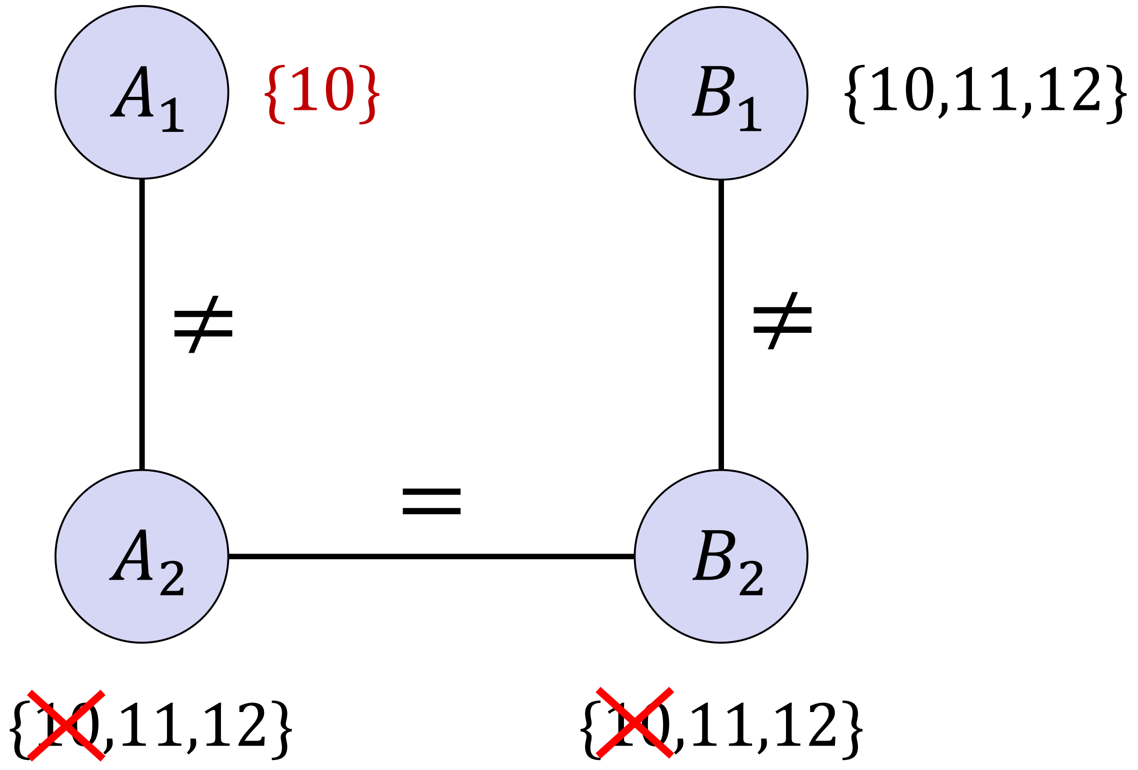
Interleaving search and AC-3:

Expand A_1 and select value

Call AC-3 on $A_2 \rightarrow A_1$

Expand B_2 and select value

Call AC-3 on $A_2 \rightarrow B_2$ and $B_1 \rightarrow B_2$

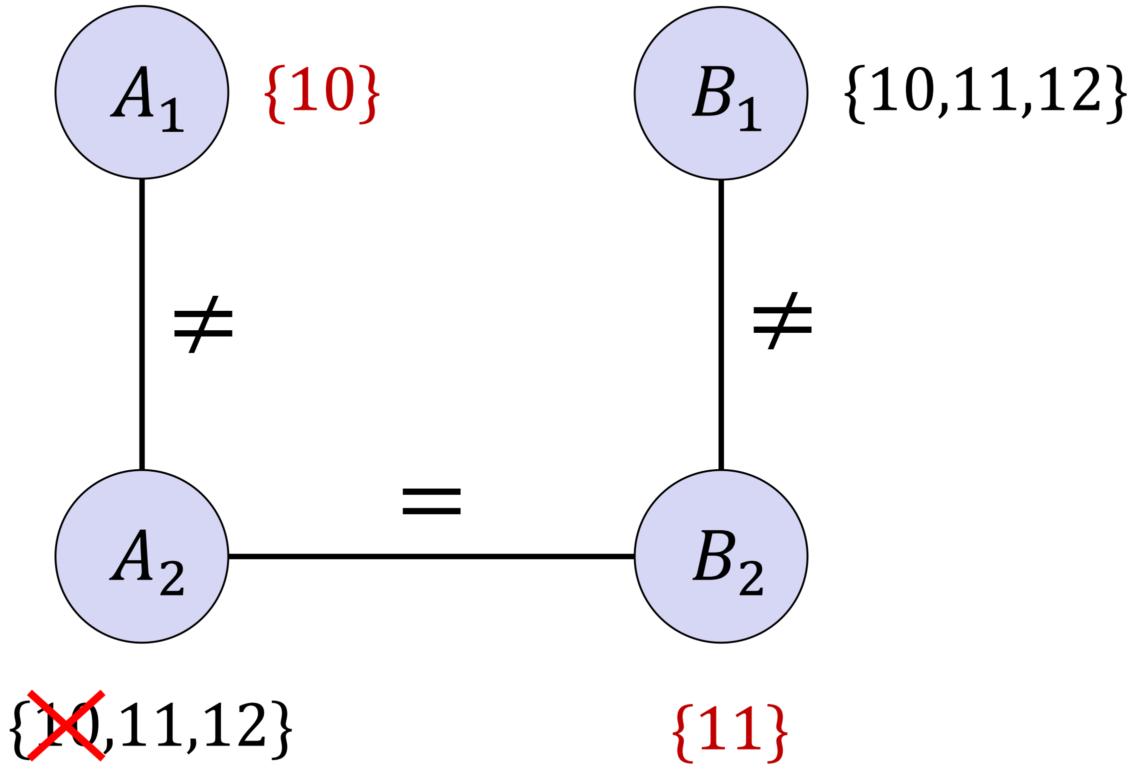


Constraint Propagation Example

Schedule 2 classes for 2 profs into 3 possible time slots

Interleaving search and AC-3:

- Expand A_1 and select value
- Call AC-3 on $A_2 \rightarrow A_1$
- Expand B_2 and select value
- Call AC-3 on $A_2 \rightarrow B_2$ and $B_1 \rightarrow B_2$



Constraint Propagation Example

Schedule 2 classes for 2 profs into 3 possible time slots

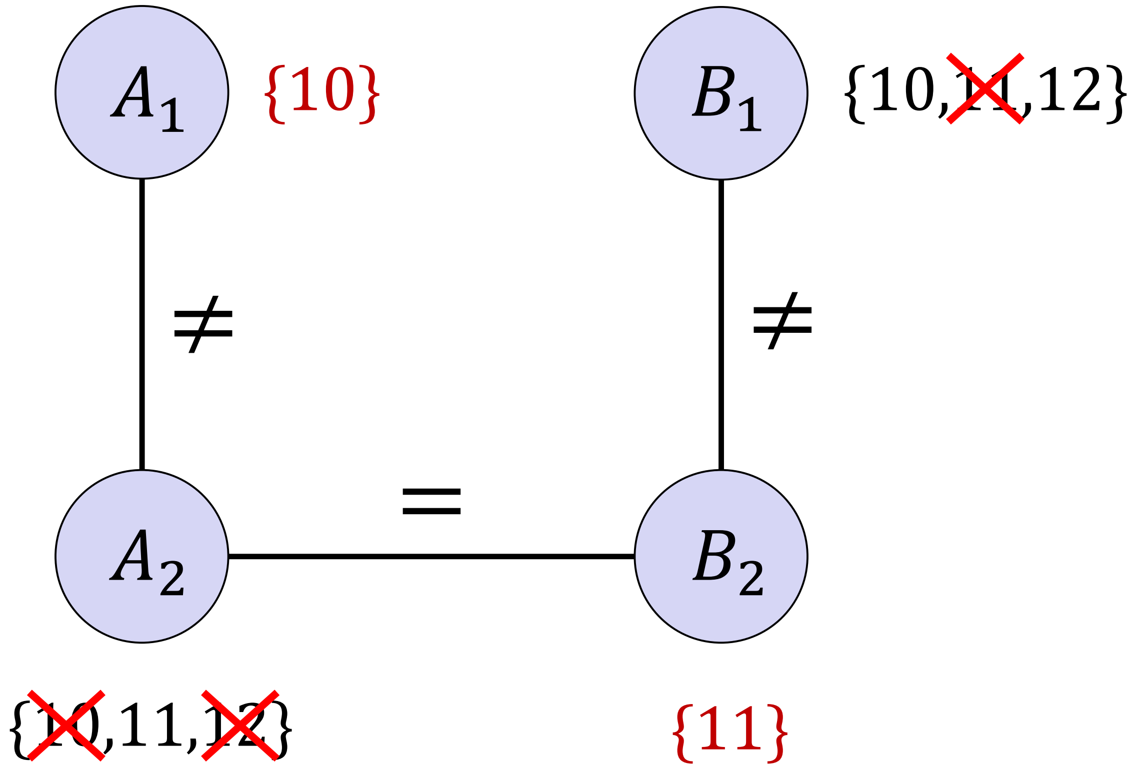
Interleaving search and AC-3:

Expand A_1 and select value

Call AC-3 on $A_2 \rightarrow A_1$

Expand B_2 and select value

Call AC-3 on $A_2 \rightarrow B_2$ and $B_1 \rightarrow B_2$



Constraint Propagation Example

Schedule 2 classes for 2 profs into 3 possible time slots

Interleaving search and AC-3:

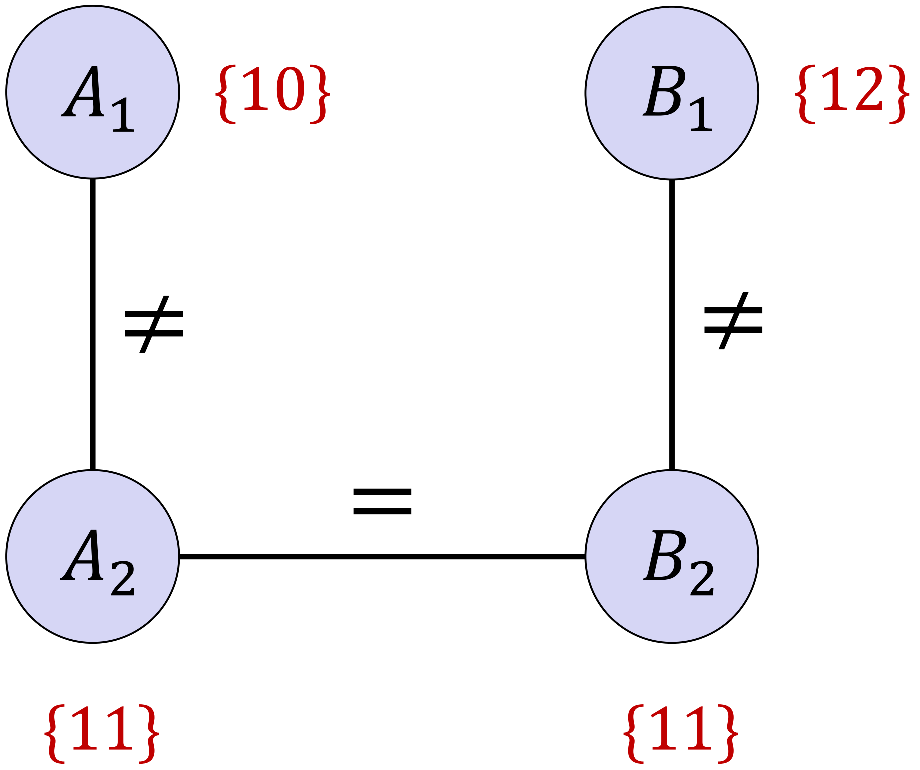
Expand A_1 and select value

Call AC-3 on $A_2 \rightarrow A_1$

Expand B_2 and select value

Call AC-3 on $A_2 \rightarrow B_2$ and $B_1 \rightarrow B_2$

Expand A_2 and B_1



Ways to Improve Backtracking Search

What variable to assign next?

Choose variable with fewest values left → fail first

What order to use for values of a variable?

Choose val that leaves most options for remaining vars → fail last

What inferences can be made from an assignment?

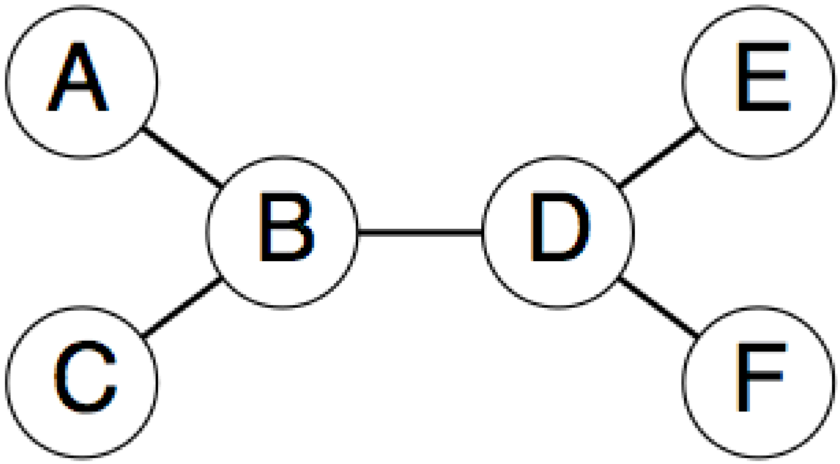
Constraint propagation to reduce the legal values for variables

How can we take advantage of problem structure?

Cutset conditioning and tree-structured CSPs

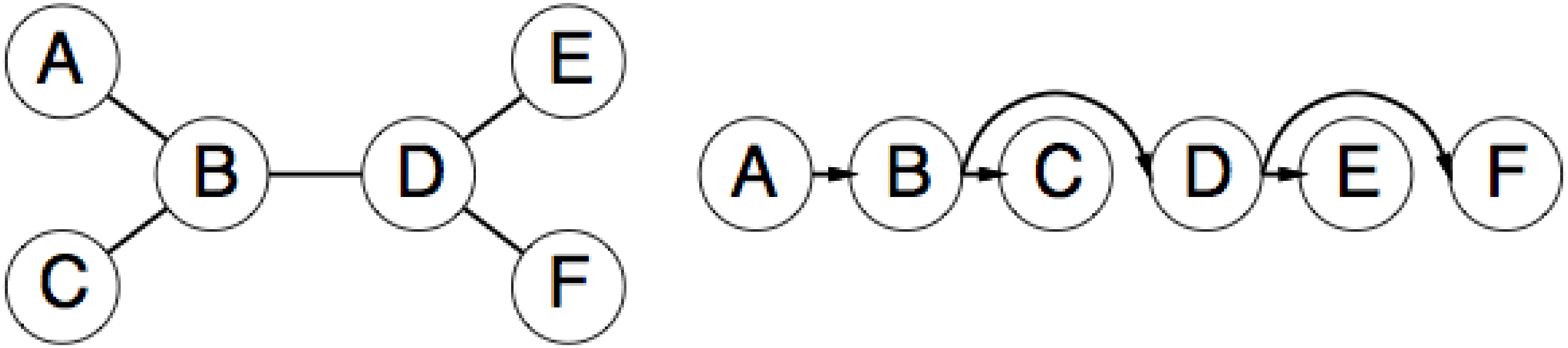
Tree-Structured CSPs

If the constraint graph has no loops (also called cycles), then the CSP can be solved in $O(nd^2)$ time (vs $O(d^n)$ in general)



Solving Tree-Structured CSPs

1. Choose a variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering



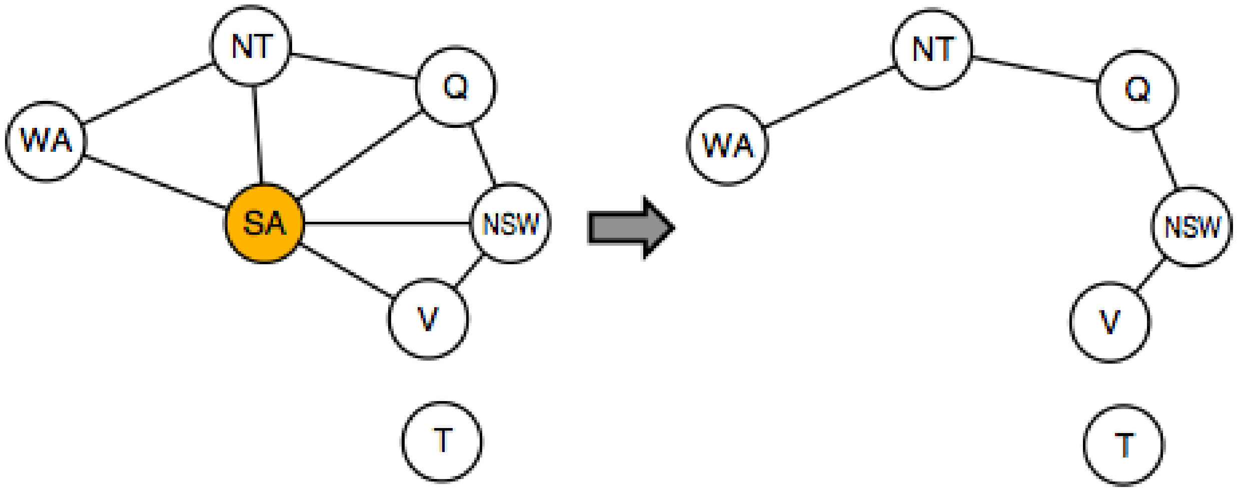
2. For j from n down to 2, apply $\text{RemoveInconsistent}(Parent(X_j), X_j)$

3. For j from 1 to n, assign X_j consistently with $Parent(X_j)$

Cutset Conditioning

Rina Dechter
Univ. of California, Irvine

Conditioning: instantiate a variable, prune its neighbors' domains



Cutset conditioning: instantiate (in all ways) a set of variables such that the remaining constraint graph is a tree

Cutset size $c \Rightarrow$ runtime $O(d^c \cdot (n-c)d^2)$, very fast for small c

Summary

Node and arc consistency refer to whether the domains for a set of variables satisfy the constraints

Forward checking enforces arc consistency for all neighbors of a variable that has been assigned during search

Constraint propagation goes further and enforces arc consistency recursively until all domains are consistent (AC-3 algorithm)

The Maintaining Arc Consistency version of backtracking search calls AC-3 after each variable assignment

Tree-structured graphs can be solved orders of magnitude faster than general graphs. Cutset conditioning can be used with graphs that are “close” to trees

Questions?