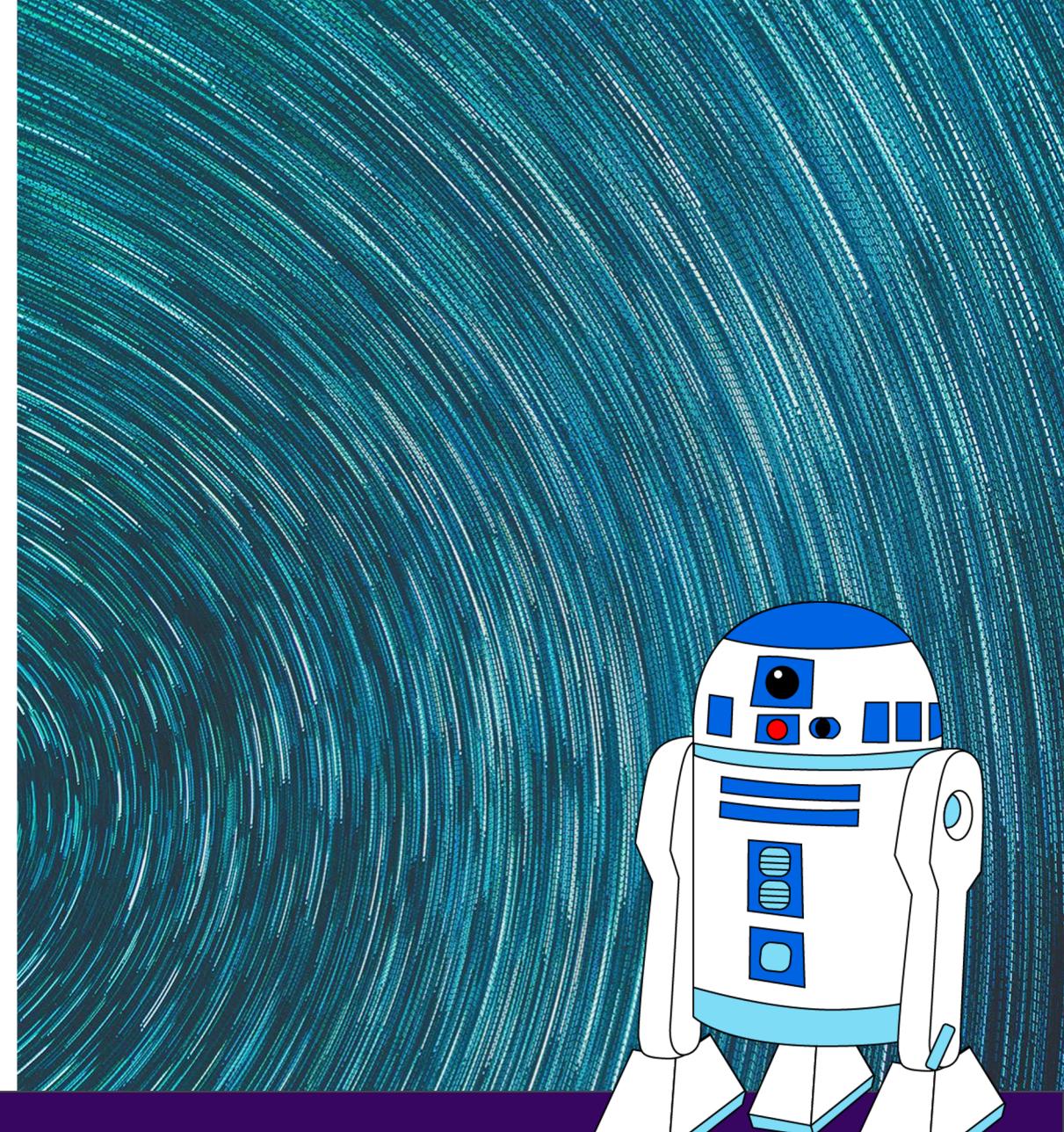


CIS 521:
ARTIFICIAL INTELLIGENCE

Constraint Satisfaction Problems

Professor Chris Callison-Burch



What is Search For?

- **Assumptions about the world: a single agent, deterministic actions, fully observed state, discrete state space**

- **Planning: sequences of actions**

The path to the goal is the important thing

Paths have various costs, depths

Heuristics give problem-specific guidance

7	2	4
5		6
8	3	1

	1	2
3	4	5
6	7	8

- **Identification: assignments to variables**

The goal itself is important, not the path

All paths at the same depth (for some formulations)

CSPs are specialized for identification problems

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		7	9	

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

Big idea

- Represent the *constraints* that solutions must satisfy in a uniform *declarative* language
- Find solutions by *GENERAL PURPOSE* search algorithms with no changes from problem to problem
 - No hand built transition functions
 - No hand built heuristics
- Just specify the problem in a formal declarative language, and a general purpose algorithm does everything else!

Constraint Satisfaction Problems

A CSP consists of:

Finite set of variables X_1, X_2, \dots, X_n

Nonempty domain of possible values for each variable

D_1, D_2, \dots, D_n where $D_i = \{v_1, \dots, v_k\}$

Finite set of constraints C_1, C_2, \dots, C_m

- Each *constraint* C_i limits the values that variables can take, e.g., $X_1 \neq X_2$. A *state* is defined as an *assignment* of values to some or all variables.

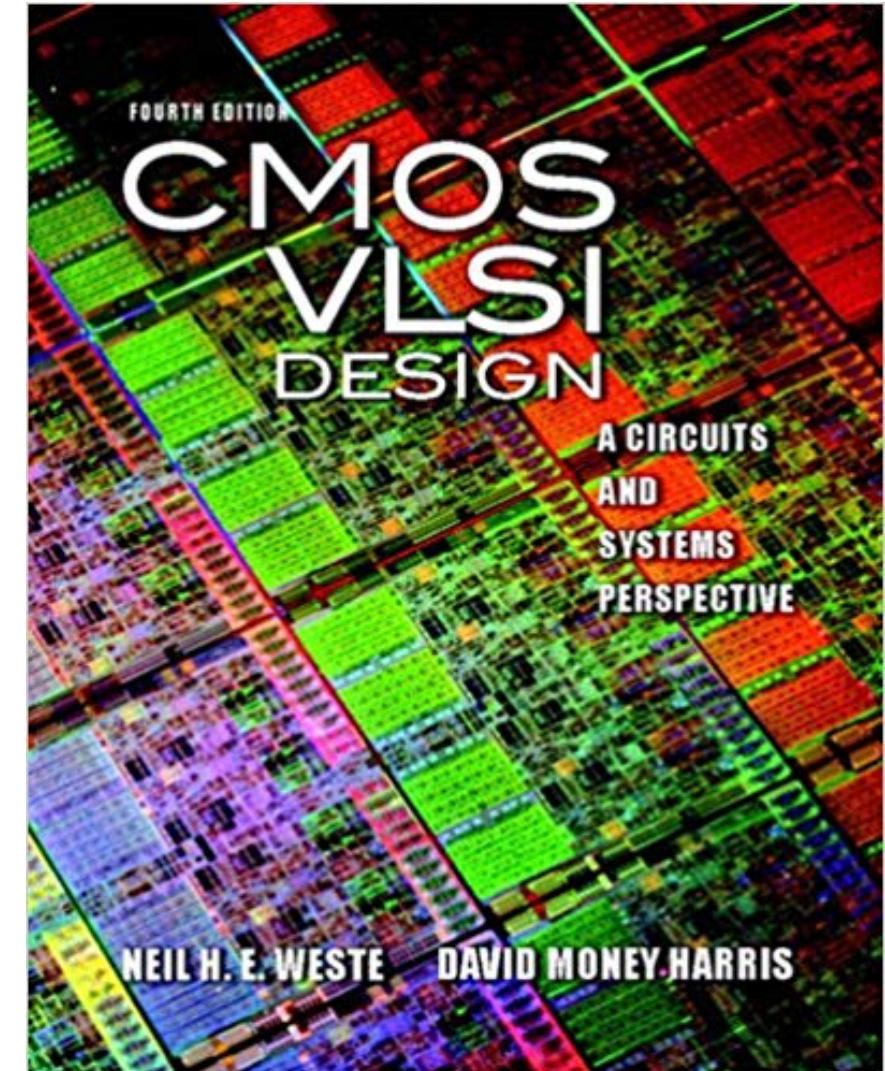
- A *consistent assignment* does not violate the constraints.
- Example problem: Sudoku

Constraint satisfaction problems

- An assignment is *complete* when every variable is assigned a value.
- A *solution* to a CSP is a *complete, consistent* assignment.
- Solutions to CSPs can be found by a completely *general purpose* algorithm, given only the formal specification of the CSP.
- Beyond our scope: CSPs that require a solution that maximizes an *objective function*.

Applications

- **Map coloring**
- **Scheduling problems**
 - Job shop scheduling
 - Scheduling the Hubble Space Telescope
- **Floor planning for VLSI**
- **Sudoku**
- ...

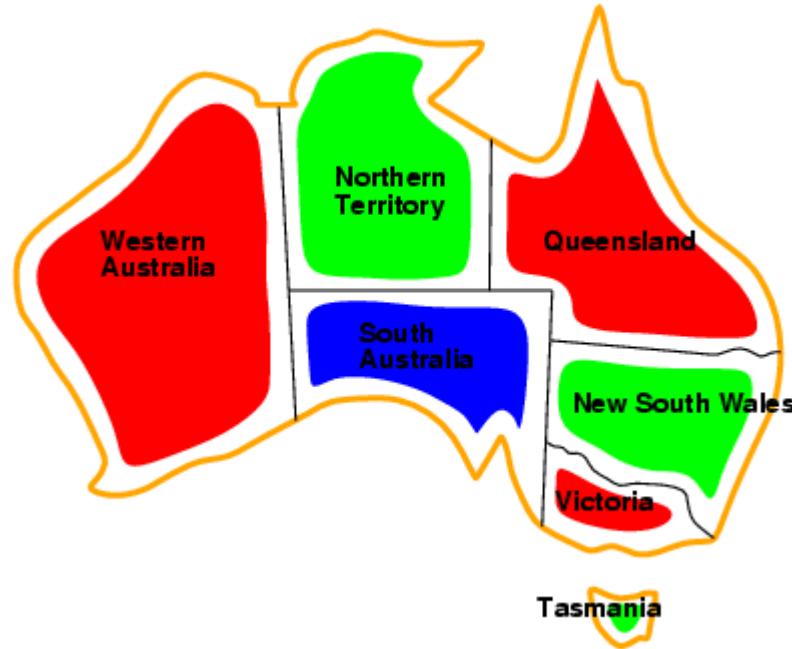


Example: Map-coloring



- **Variables:** WA, NT, Q, NSW, V, SA, T
- **Domains:** $D_i = \{\text{red}, \text{green}, \text{blue}\}$
- **Constraints:** adjacent regions must have different colors
 - e.g., $WA \neq NT$
 - So (WA, NT) must be in $\{(\text{red}, \text{green}), (\text{red}, \text{blue}), (\text{green}, \text{red}), \dots\}$

Example: Map-coloring



Solutions: complete and consistent assignments

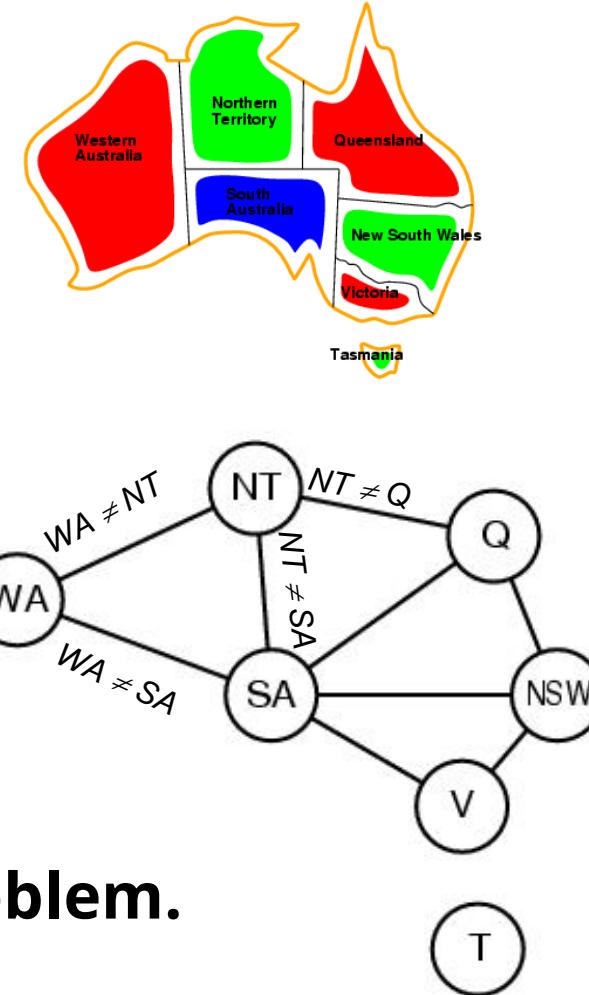
e.g., WA = red, NT = green, Q = red, NSW = green,
V = red, SA = blue, T = green

Benefits of CSP

- **Clean specification of many problems, generic goal, successor function & heuristics**
Just represent problem as a CSP & solve with general package
- **CSP “knows” which variables violate a constraint**
And hence where to focus the search
- **CSPs: Automatically prune off all branches that violate constraints**
(State space search could do this only by *hand-building constraints into the successor function*)

CSP Representations

- **Constraint graph:**
 - nodes** are variables
 - arcs** are (binary) constraints
- **Standard representation pattern:**
 - variables with values
- **Constraint graph** simplifies search.
 - e.g. Tasmania is an independent subproblem.
- **This problem: A binary CSP:**
 - each constraint relates two variables



Varieties of CSPs

- ***Discrete variables***

finite domains:

- n variables, domain size $d \rightarrow O(d^n)$ complete assignments
- e.g., Boolean CSPs, includes Boolean satisfiability (NP-complete)

infinite domains:

- integers, strings, etc.
- e.g., job scheduling, variables are start/end days for each job
- need a constraint language, e.g., $StartJob_1 + 5 \leq StartJob_3$

- ***Continuous variables***

e.g., start/end times for Hubble Space Telescope observations

linear constraints solvable in polynomial time by linear programming

Varieties of constraints

- **Unary** constraints involve a single variable,
e.g., SA \neq green
- **Binary** constraints involve pairs of variables,
e.g., SA \neq WA
- **Higher-order** constraints involve 3 or more variables
e.g., crypt-arithmetic column constraints
- **Preference** (soft constraints) e.g. *red is better than green* can be represented by a cost for each variable assignment
Constrained optimization problems.

Idea 1: CSP as a search problem

- **A CSP can easily be expressed as a search problem**

Initial State: the empty assignment {}.

Successor function: Assign value to any unassigned variable *provided that there is not a constraint conflict.*

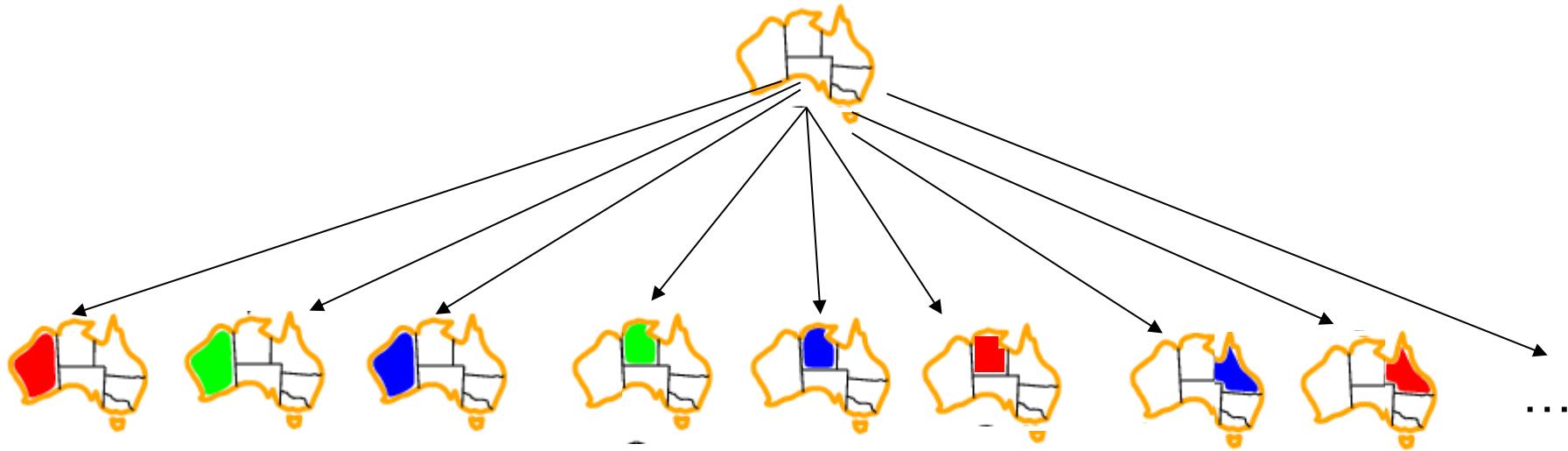
Goal test: the current assignment is complete.

Path cost: a constant cost for every step.

- **Solution is always found at depth n , for n variables**

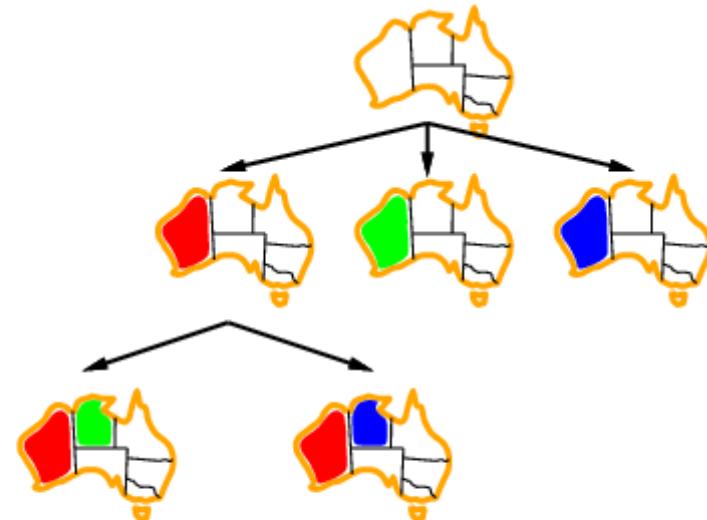
Hence Depth First Search can be used

Search and branching factor



- **n variables of domain size d**
- Branching factor at the root is **$n*d$**
- Branching factor at next level is **$(n-1)*d$**
- Tree has **$n!*d^n$ leaves**

Search and branching factor

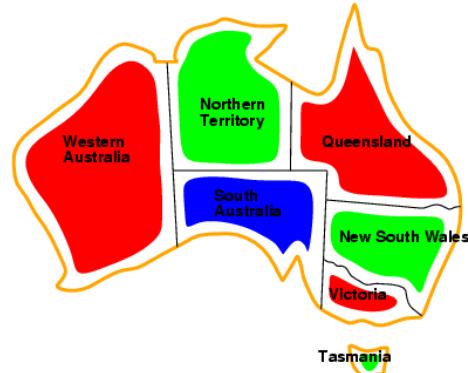


- The variable assignments are *commutative*
Eg [step 1: **WA = red**; step 2: **NT = green**]
equivalent to [step 1: **NT = green**; step 2: **WA = red**]
Therefore, a *tree search*, not a *graph search*
- Only need to consider assignments to a single variable at each node
 $b = d$ and there are d^n leaves (*n variables, domain size d*)

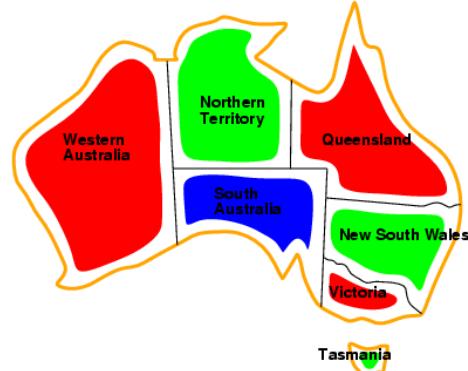
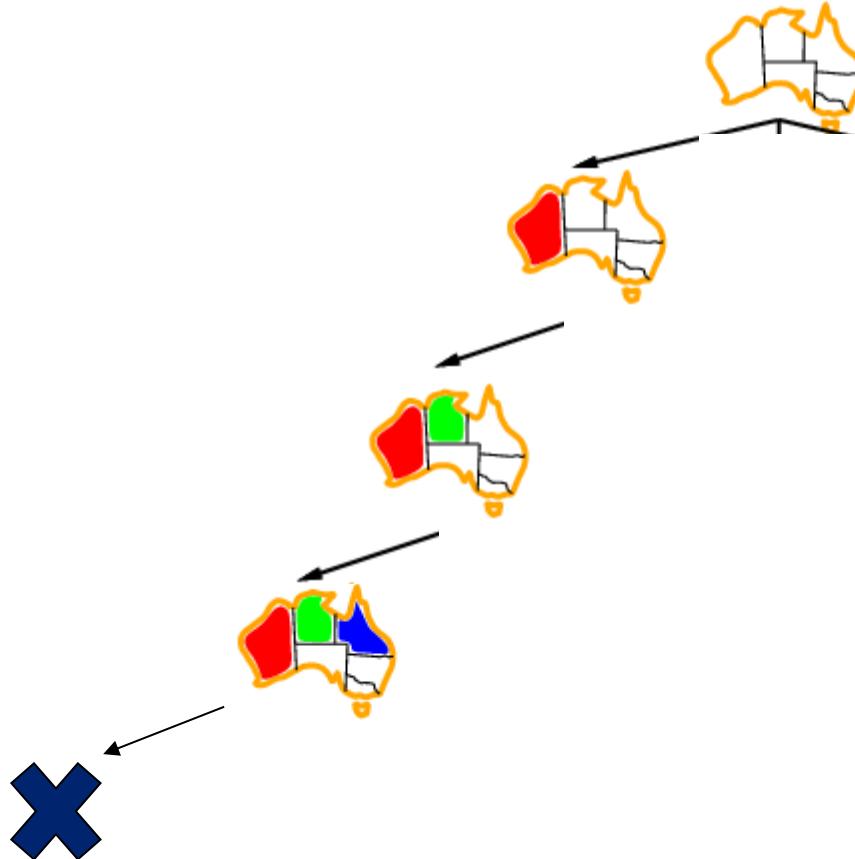
Search and *Backtracking*

- Depth-first search for CSPs with single-variable assignments is called *backtracking* search
- The term backtracking search is used for a depth-first search that chooses values for one variable at a time and backtracks when a variable has no legal values left to assign.
- Backtracking search is the basic *uninformed* algorithm for CSPs

Backtracking example



Backtracking example

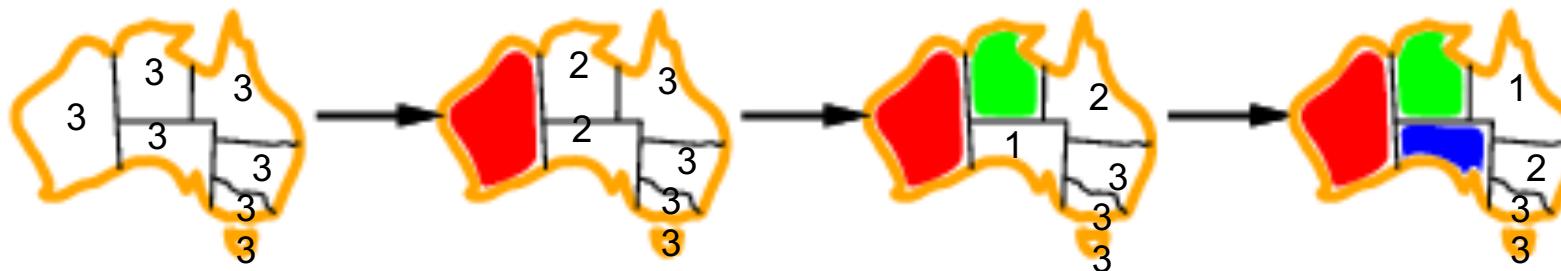


Idea 2: Improving backtracking efficiency

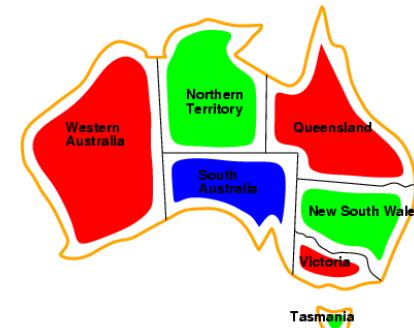
- ***General-purpose* methods & *general-purpose* heuristics can give huge gains in speed, *on average***
- **Heuristics:**
 - Q: Which variable should be assigned next?
 1. **Most constrained** variable
 2. (if ties:) **Most constraining** variable
 - Q: In what order should that variable's values be tried?
 3. **Least constraining** value
 - Q: Can we detect inevitable failure early?
 4. **Forward checking**

Heuristic 1: Most constrained *ed* variable

- Choose a variable with the *fewest legal values*



- a.k.a. *minimum remaining values (MRV)* heuristic



Heuristic 2: Most constraining variable

- Tie-breaker among most constrained variables
- Choose the variable with the *most constraints on remaining variables*



These two heuristics together lead to immediate solution of our example problem

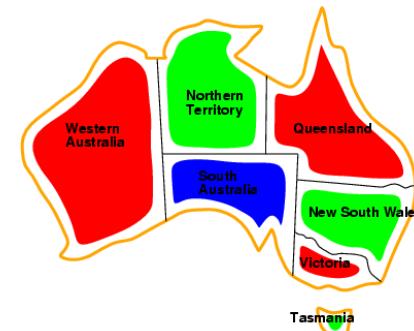


Heuristic 3: Least constraining *value*

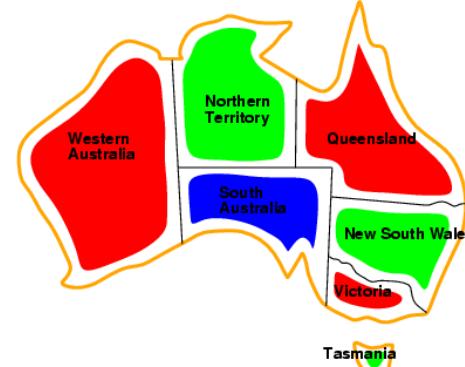
- Given a variable, *choose the least constraining value*:
the one that rules out the fewest values in the remaining variables



Note: demonstrated here independent of the other heuristics



Heuristic 4: Forward checking



- **Idea:**

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



New data structure

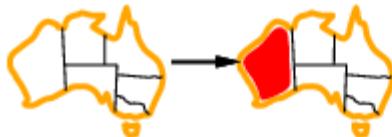
(A first step towards Arc Consistency & AC-3)

Forward checking



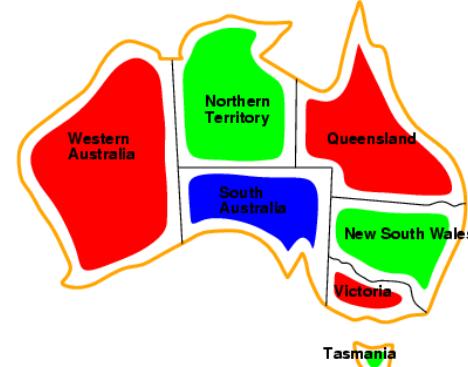
- **Idea:**

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



WA	NT	Q	NSW	V	SA	T
■ Red ■ Green ■ Blue						
■ Red	■ Yellow	■ Red ■ Green ■ Blue	■ Red ■ Green ■ Blue	■ Red ■ Green ■ Blue	■ Yellow	■ Red ■ Green ■ Blue

Forward checking



- **Idea:**

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



WA	NT	Q	NSW	V	SA	T
Red, Green, Blue						
Red	Green, Blue	Red, Green, Blue	Red, Green, Blue	Red, Green, Blue	Green, Blue	Red, Green, Blue
Red	Yellow	Green	Yellow	Red, Green, Blue	Yellow	Red, Green, Blue

Forward checking



- **Idea:**

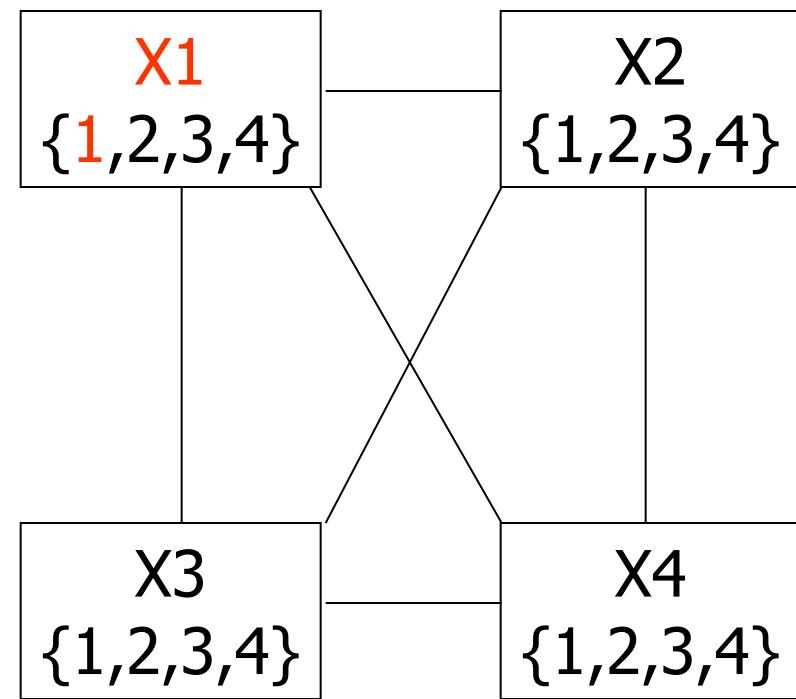
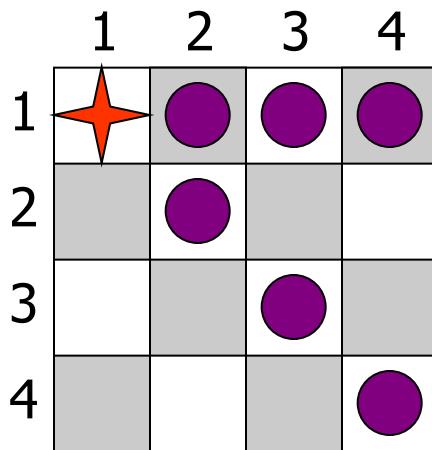
Keep track of remaining legal values for unassigned variables
Terminate search when any unassigned variable has no remaining legal values



WA	NT	Q	NSW	V	SA	T
■ Red	■ Green	■ Blue				
■ Red		■ Green	■ Blue		■ Green	■ Blue
■ Red			■ Blue	■ Green	■ Blue	■ Green
■ Red		■ Green	■ Yellow	■ Blue	■ Yellow	■ Green

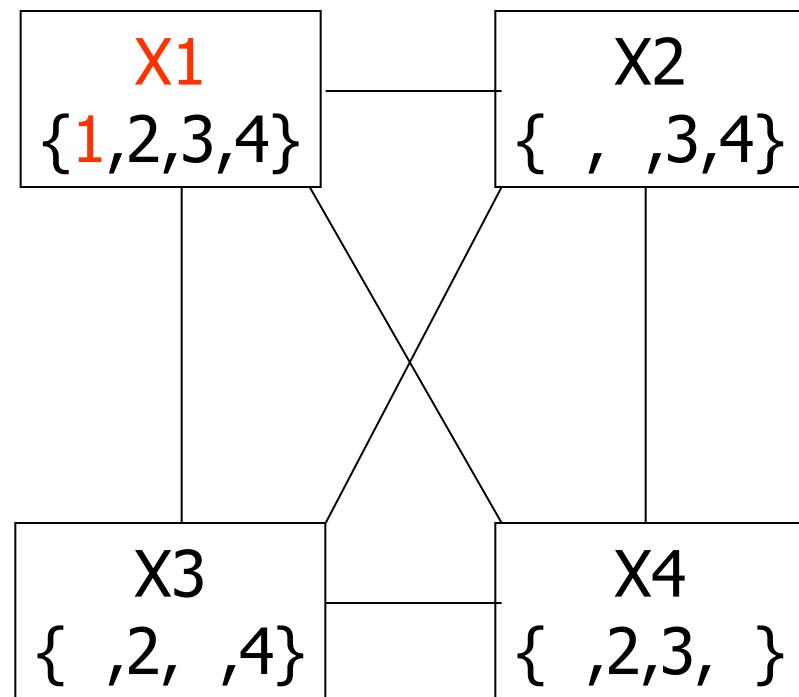
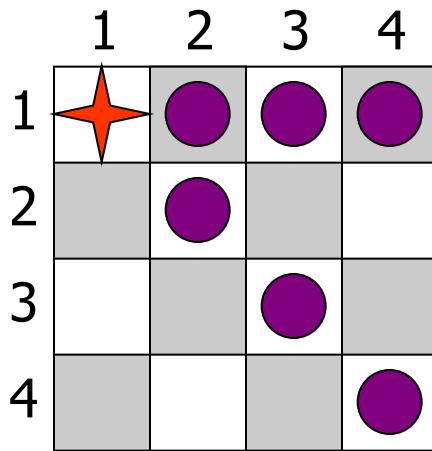
Terminate! No possible value for SA

Example: 4-Queens Problem



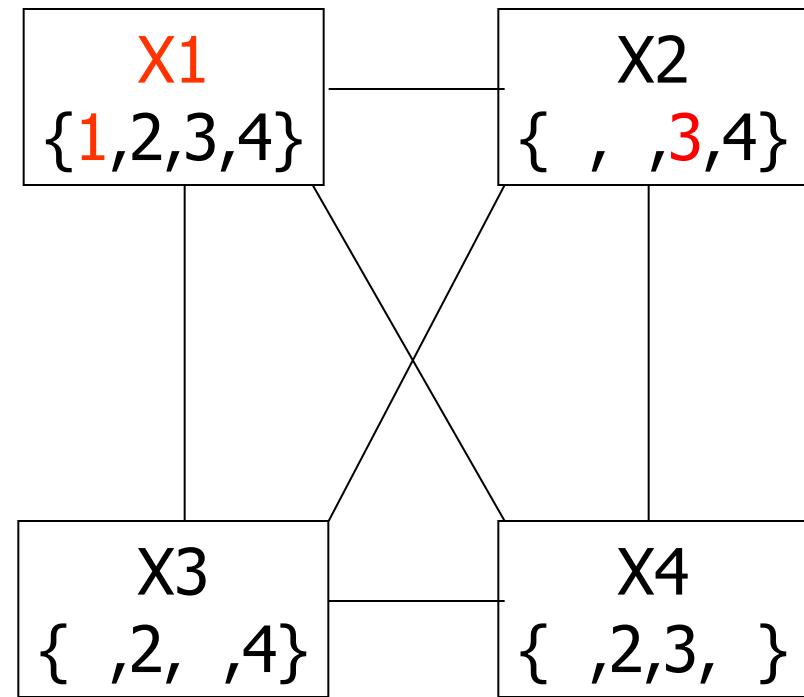
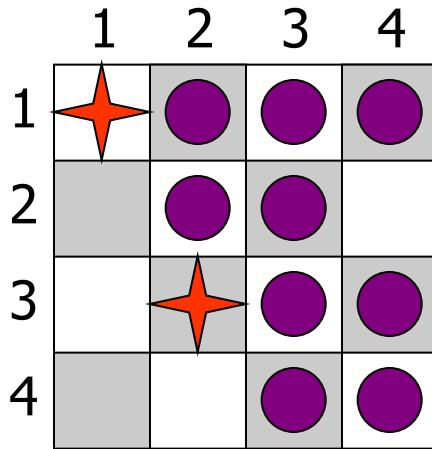
Assign value to unassigned variable

Example: 4-Queens Problem



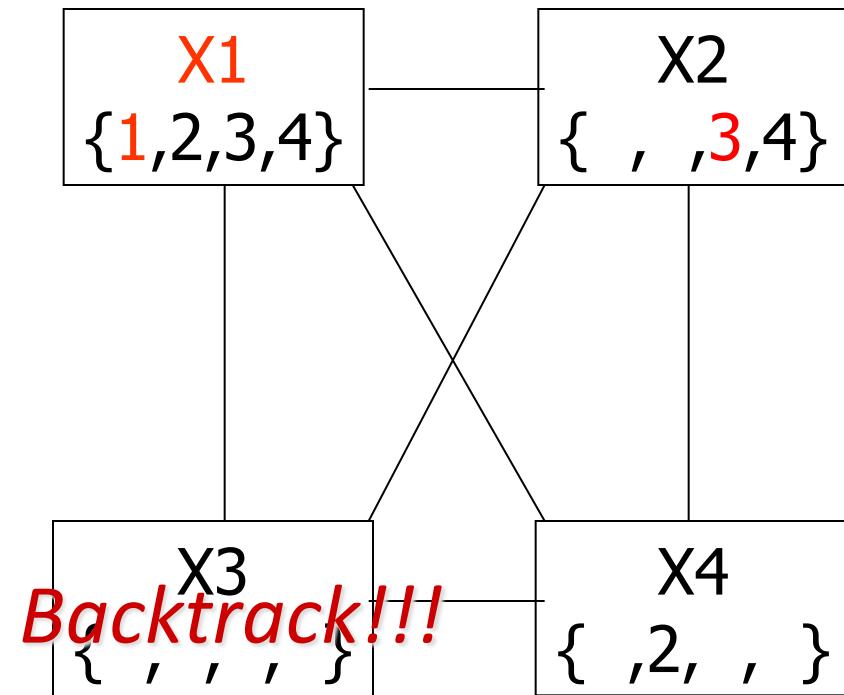
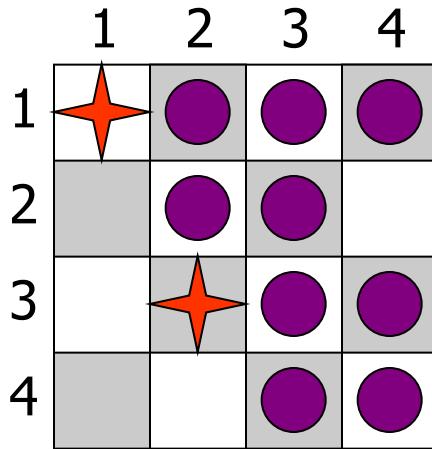
Forward check!

Example: 4-Queens Problem



Assign value to unassigned variable

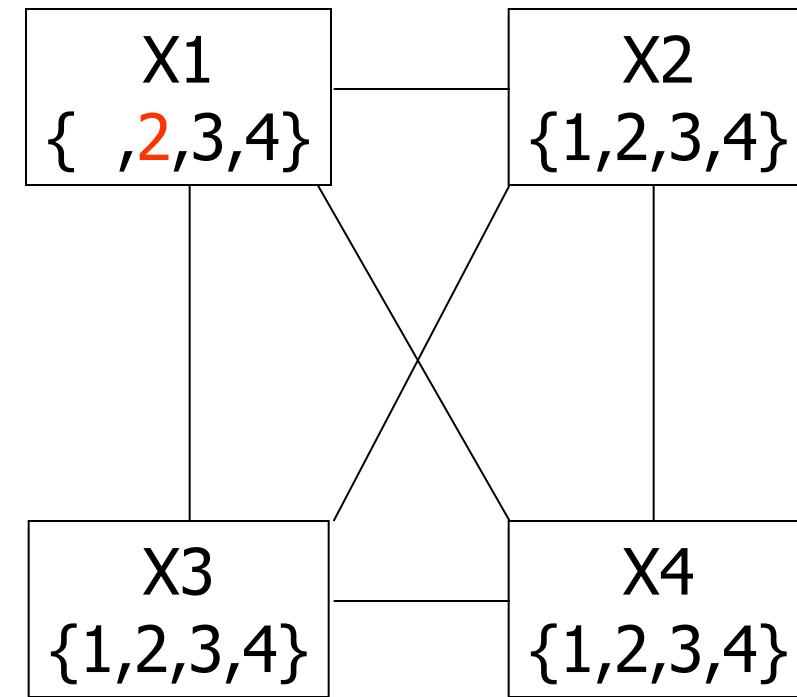
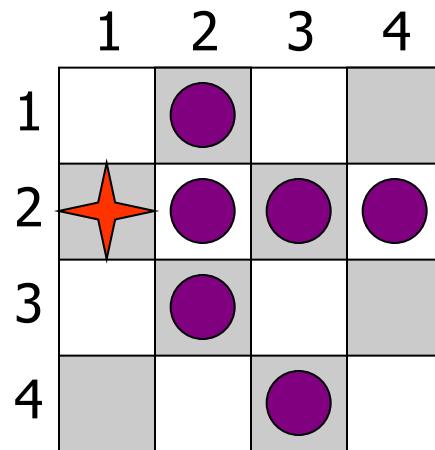
Example: 4-Queens Problem



Forward check!

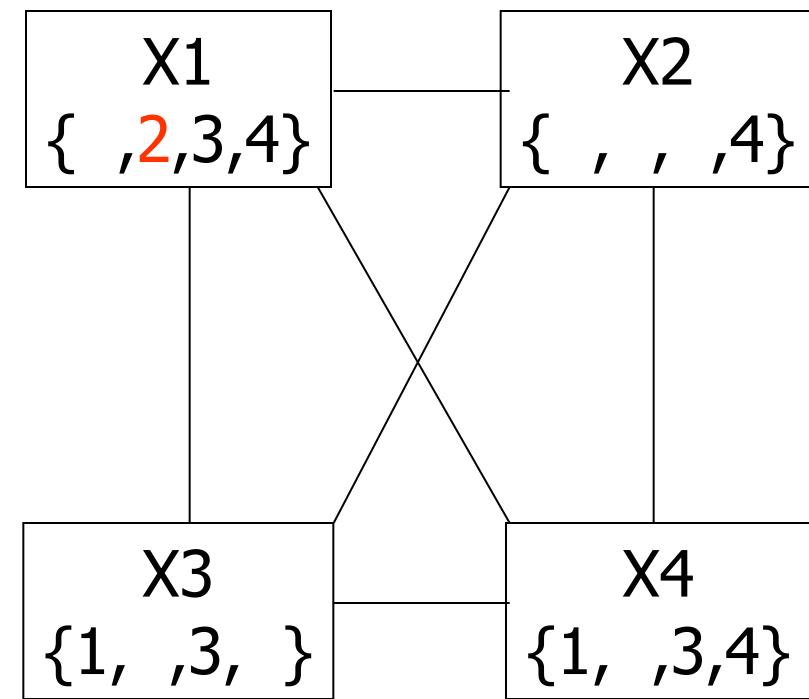
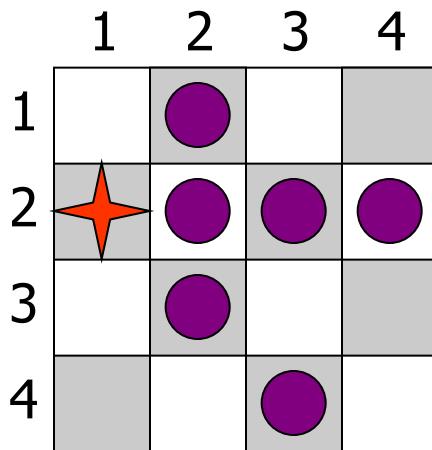
Example: 4-Queens Problem

*Picking up a little later after
two steps of backtracking....*



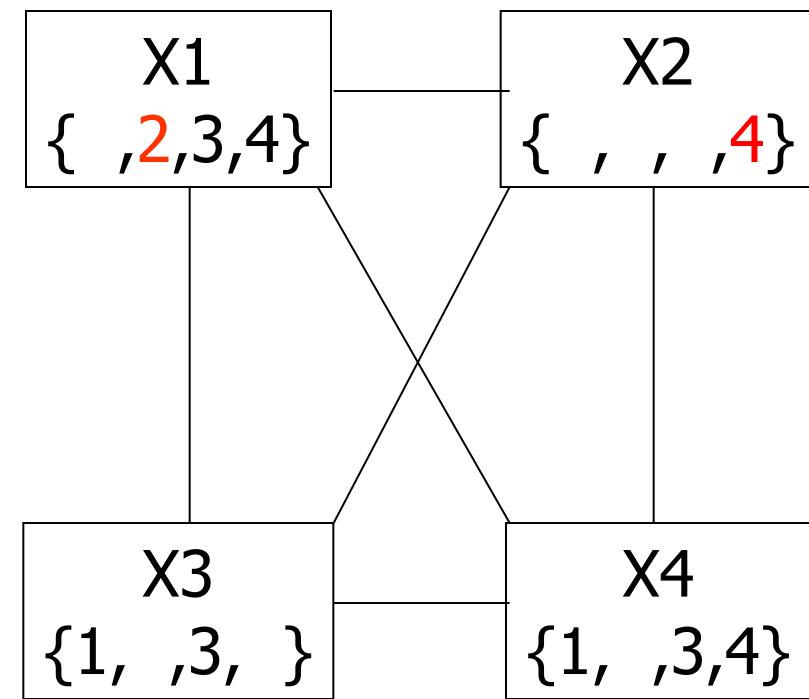
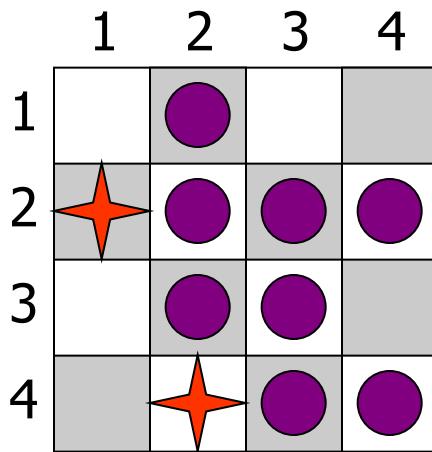
Assign value to unassigned variable

Example: 4-Queens Problem



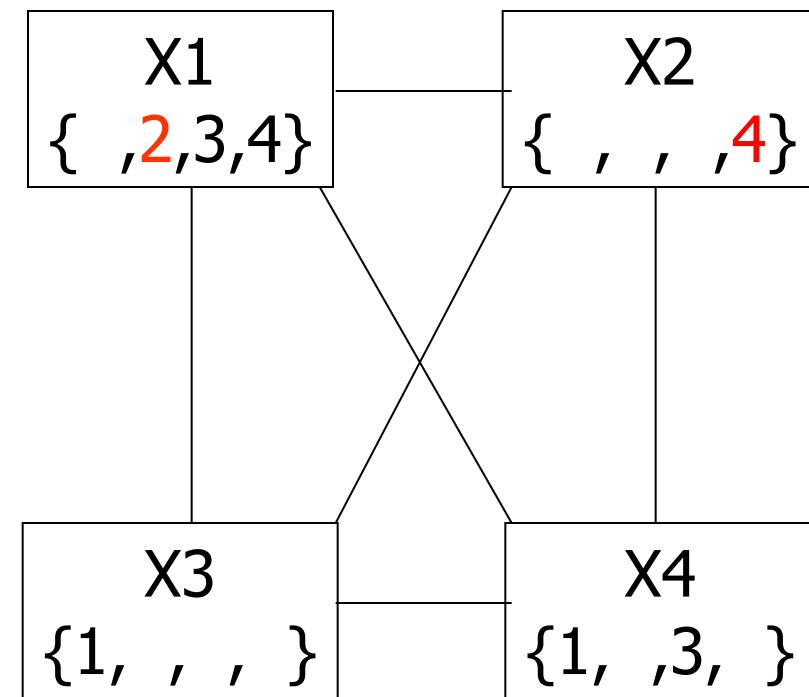
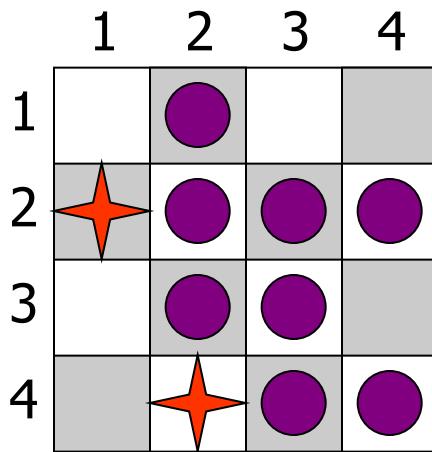
Forward check!

Example: 4-Queens Problem



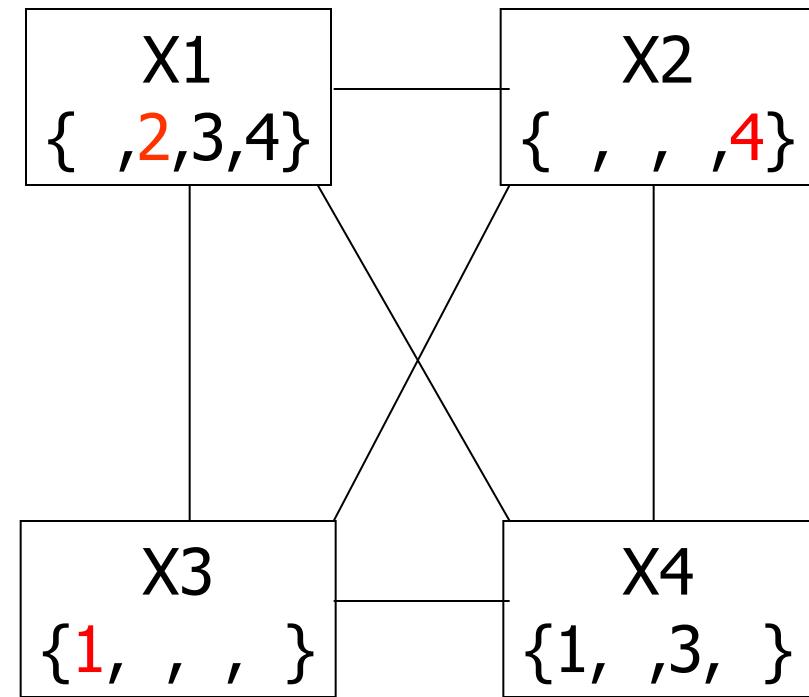
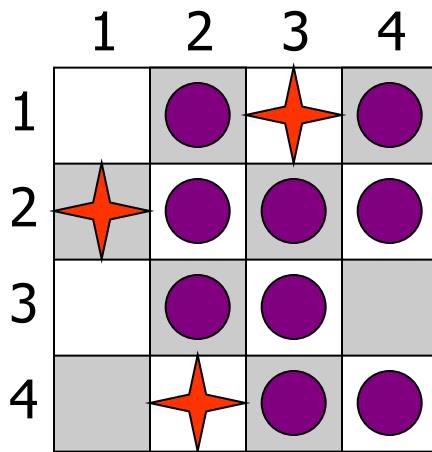
Assign value to unassigned variable

Example: 4-Queens Problem



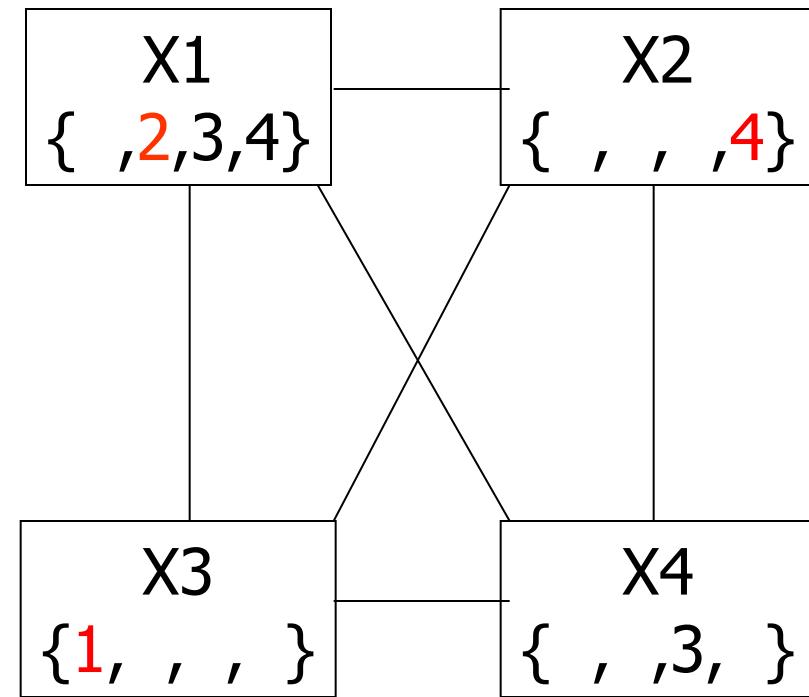
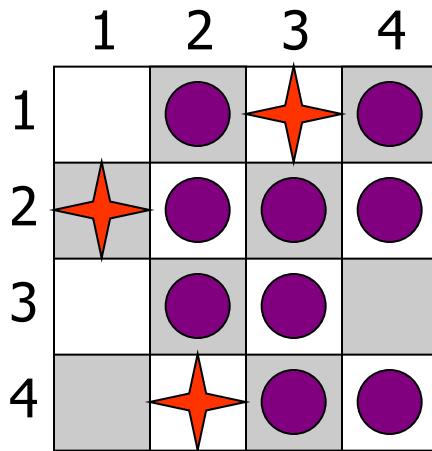
Forward check!

Example: 4-Queens Problem

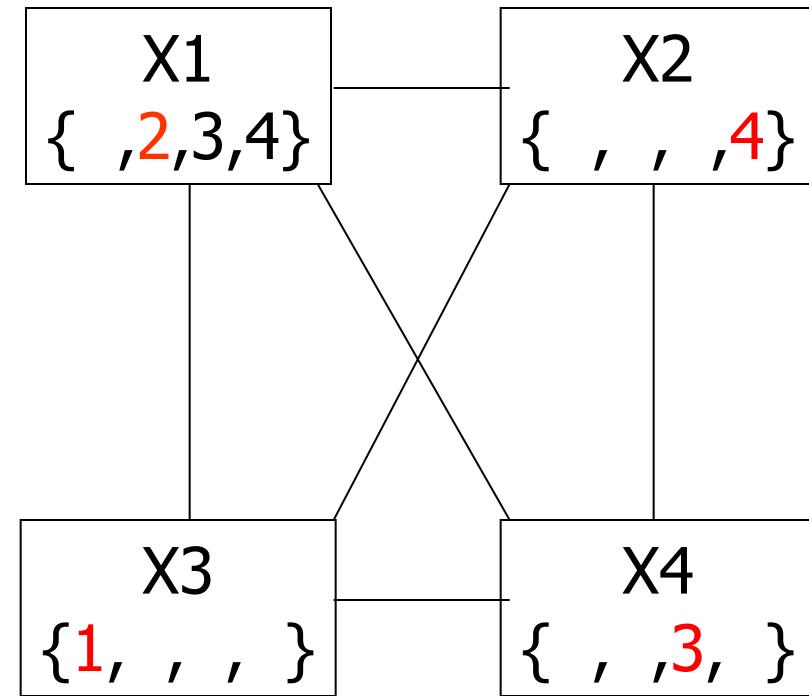
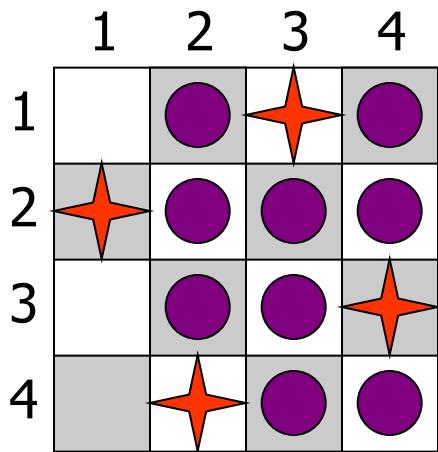


Assign value to unassigned variable

Example: 4-Queens Problem



Example: 4-Queens Problem

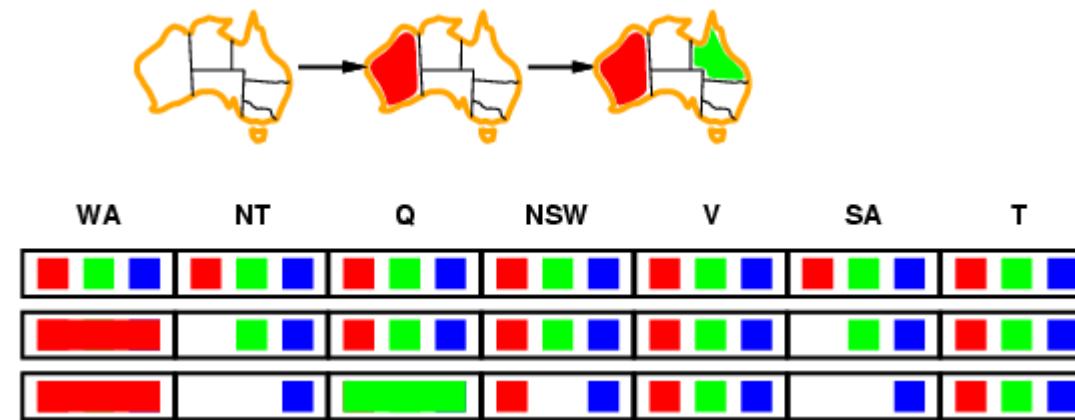


Assign value to unassigned variable

Towards Constraint propagation



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



- NT and SA cannot both be blue!
- **Constraint propagation** goes beyond forward checking & repeatedly enforces constraints locally

Arc Consistency, Constraint Propagation & AC-3

Idea 3 (*big idea*): *Inference* in CSPs

- **CSP solvers combine search *and inference***

Search

- assigning a value to a variable

Constraint propagation (inference)

- Eliminates possible values for a variable if the value would violate **local consistency**

Can do inference first, or intertwine it with search

- You'll investigate this in the Sudoku homework

- **Local consistency**

Node consistency: satisfies unary constraints

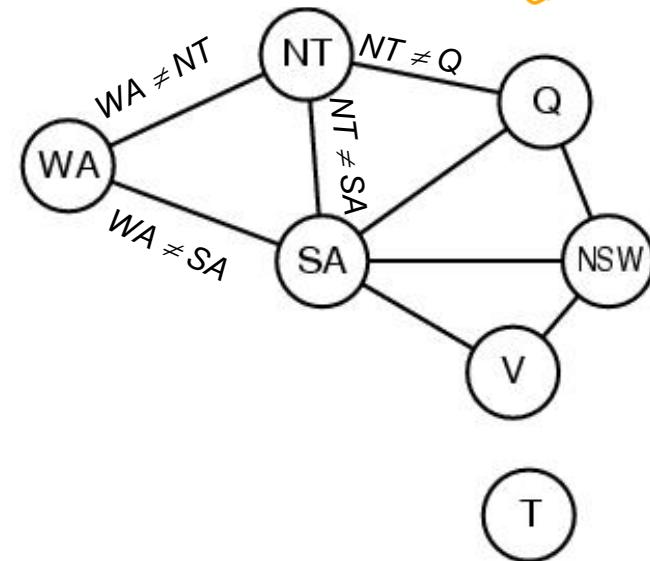
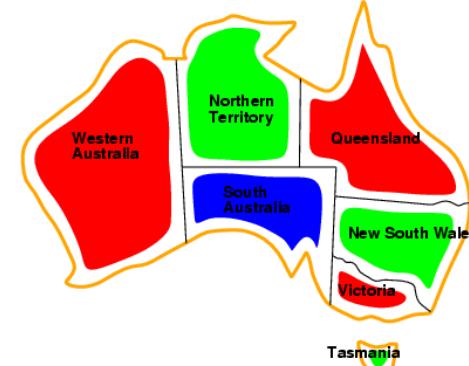
- This is trivial!

Arc consistency: satisfies binary constraints

- (X_i is arc-consistent w.r.t. X_j if for every value v in D_i , there is some value w in D_j that satisfies the binary constraint on the arc between X_i and X_j)

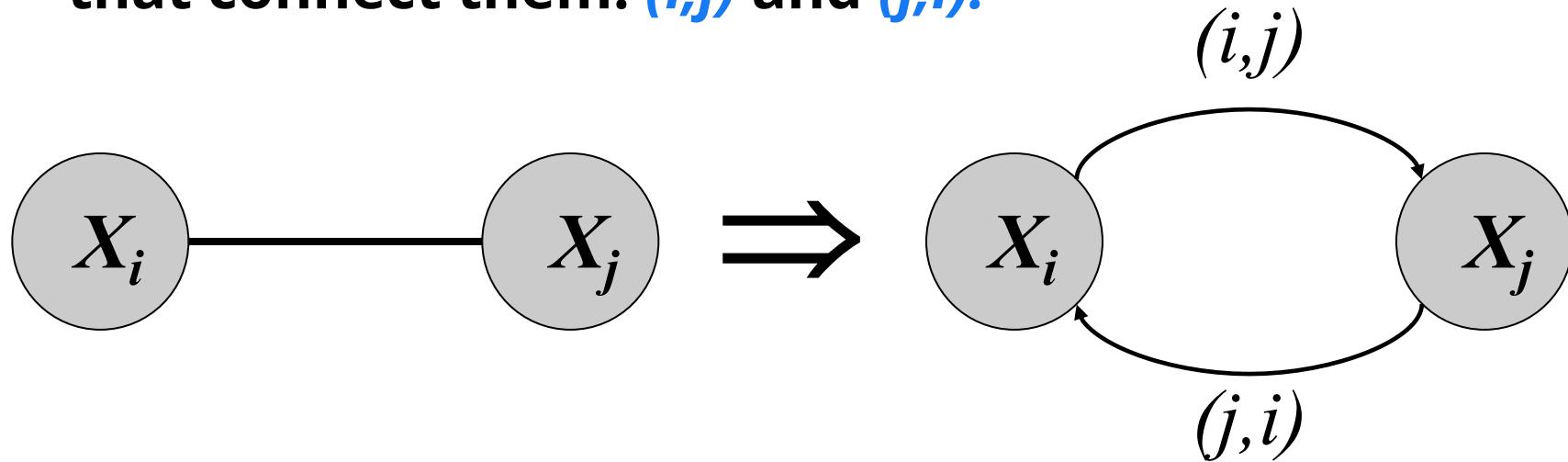
CSP Representations

- **Constraint graph:**
nodes are variables
edges are constraints



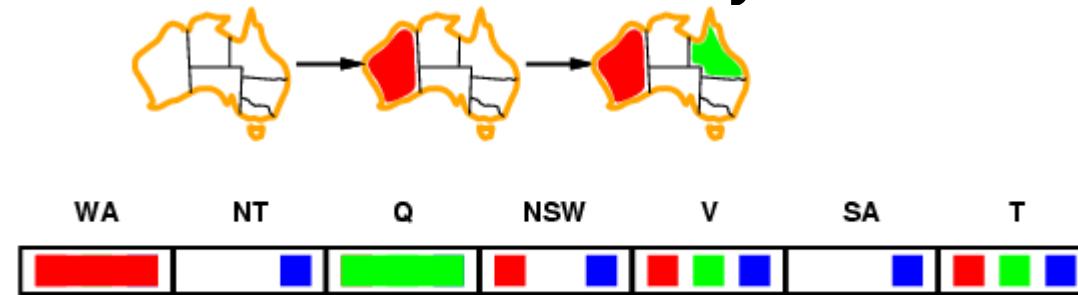
Edges to Arcs: From Constraint Graph to Directed Graph

- Given a pair of nodes X_i and X_j connected by a constraint **edge**, we represent this not by a single undirected edge, but a **pair of directed arcs**.
For a connected pair of nodes X_i and X_j , there are **two** arcs that connect them: (i,j) and (j,i) .



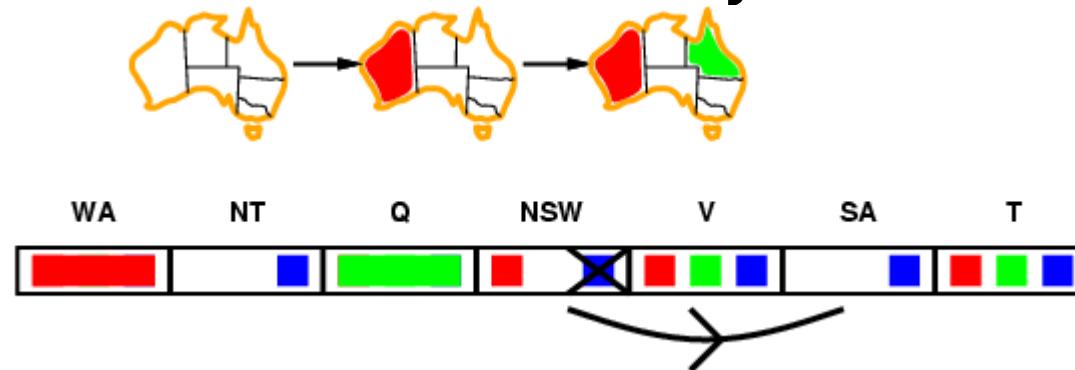
Arc consistency

- Simplest form of propagation makes each arc **consistent**
- $X \rightarrow Y$ is consistent iff
for **every** value x of X there is **some** allowed y



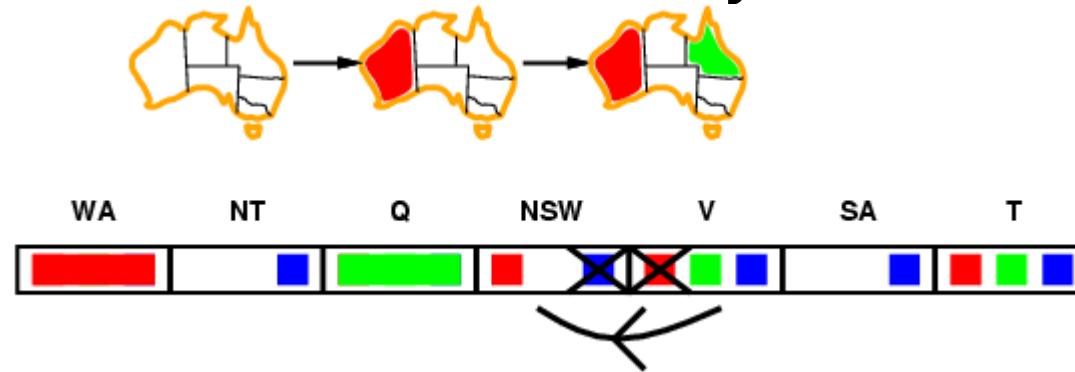
Arc consistency

- Simplest form of propagation makes each arc **consistent**
- $X \rightarrow Y$ is consistent iff
for **every** value x of X there is **some** allowed y



Arc consistency

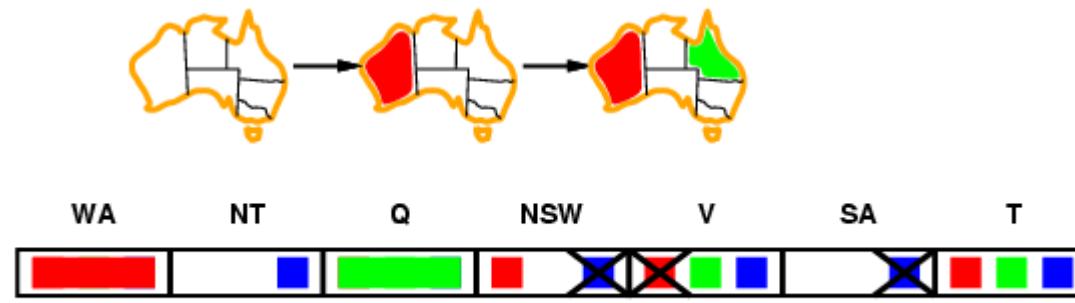
- Simplest form of propagation makes each arc **consistent**
- $X \rightarrow Y$ is consistent iff
for **every** value x of X there is **some** allowed y



- If X loses a value, recheck neighbors of X

Arc consistency

- Simplest form of propagation makes each arc **consistent**
- $X \rightarrow Y$ is consistent iff
for **every** value x of X there is **some** allowed y



- If X loses a value,
- Detects failure earlier than forward checking
- Can be run as a preprocessor or after each assignment

Arc Consistency

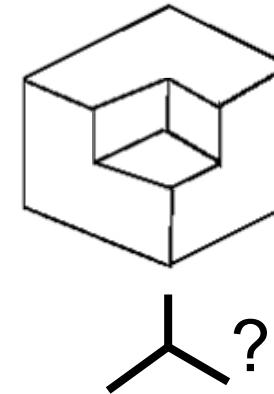
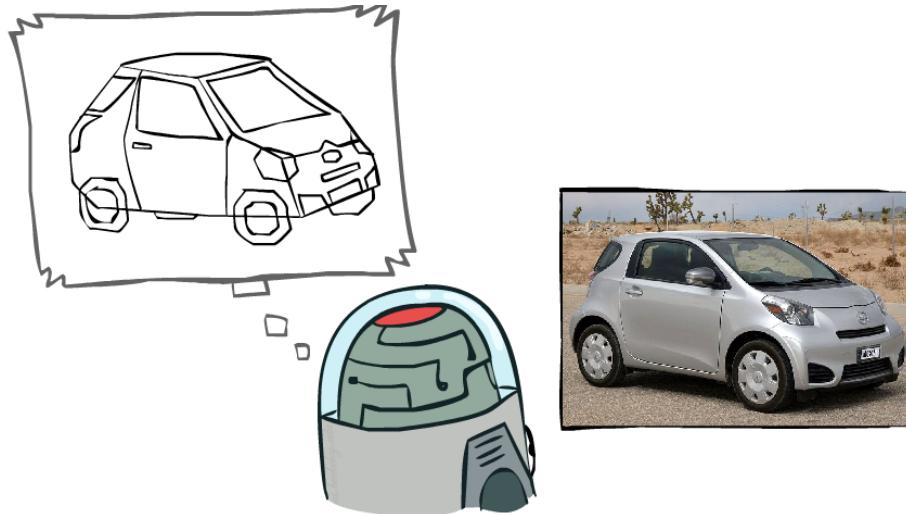
An arc (i,j) is **arc consistent** if and only if every value v on X_i is consistent with some label on Y_j .

To make an arc (i,j) arc consistent,
for each value v on X_i ,
if there is no label on Y_j consistent with v
then remove v from X_i

- Given d values, checking arc (i,j) takes **$O(d^2)$** time worst case

Example: The Waltz Algorithm

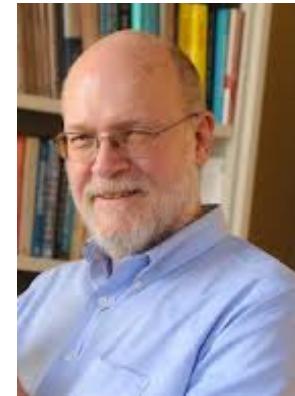
- The Waltz algorithm is for interpreting line drawings of solid polyhedra as 3D objects
- An early example of an AI computation posed as a CSP



- Approach:
 - Each intersection is a variable
 - Adjacent intersections impose constraints on each other
 - Solutions are physically realizable 3D interpretations

Slide credit: Dan Klein and Pieter Abbeel
<http://ai.berkeley.edu>

Replacing Search: Constraint Propagation Invented...



Dave Waltz's insight:

- By **iterating** over the graph, the arc-consistency **constraints** can be **propagated** along arcs of the graph.
- **Search**: Use constraints to **add labels** to find **one solution**
- **Constraint Propagation**: Use constraints to **eliminate labels** to simultaneously find **all solutions**

The Waltz/Mackworth Constraint Propagation Algorithm

1. Assign *every* node in the constraint graph a set of *all* possible values
2. Repeat until there is no change in the set of values associated with any node:
 3. For each node i :
 4. For each neighboring node j in the picture:
 5. Remove any value from i which is not arc consistent with j .

Inefficiencies: Towards AC-3

1. At each iteration, we only need to examine those X_i where at least one neighbor of X_i has lost a value in the previous iteration.
2. If X_i loses a value only because of arc inconsistencies with X_j , we don't need to check X_j on the next iteration.
3. Removing a value on X_i can only make X_j arc-inconsistent with respect to X_i itself. Thus, we only need to check that (j,i) is still arc-consistent.

These insights lead to a much better algorithm...

AC-3

function AC-3(*csp*) return the CSP, possibly with reduced domains

inputs: *csp*, a binary CSP with variables $\{X_1, X_2, \dots, X_n\}$

local variables: *queue*, a queue of arcs initially the arcs in *csp*

while *queue* is not empty **do**

$(X_i, X_j) \leftarrow \text{REMOVE-FIRST}(\text{queue})$

if *REMOVE-INCONSISTENT-VALUES*(X_i, X_j) **then**

for each X_k in *NEIGHBORS*[X_i] – $\{X_j\}$ **do**
 add (X_k, X_i) to *queue*

function REMOVE-INCONSISTENT-VALUES(X_i, X_j) return *true* iff we remove a value

***removed* $\leftarrow \text{false}$**

for each x in *DOMAIN*[X_i] **do**

if no value y in *DOMAIN*[X_j] **allows** (x, y) **to satisfy**
 the constraints between X_i **and** X_j

then delete x **from** *DOMAIN*[X_i]; ***removed* $\leftarrow \text{true}$**

return *removed*

AC-3: Worst Case Complexity Analysis

- All nodes can be connected to *every* other node,
so each of n nodes must be compared against $n-1$ other nodes,
so total # of arcs is $2*n*(n-1)$, i.e. $O(n^2)$
- If there are d values, checking arc (i,j) takes $O(d^2)$ time
- Each arc (i,j) can only be inserted into the queue d times
- Worst case complexity: $O(n^2d^3)$

(For *planar* constraint graphs, the number of arcs can only be *linear in N* and the time complexity is only $O(nd^3)$)

CSPs and Search

- **Assumptions about the world: a single agent, deterministic actions, fully observed state, discrete state space**

- **Planning: sequences of actions**

The path to the goal is the important thing
Paths have various costs, depths
Heuristics give problem-specific guidance

- **Identification: assignments to variables**

The goal itself is important, not the path
All paths at the same depth (for some formulations)
CSPs are specialized for identification problems

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

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			

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