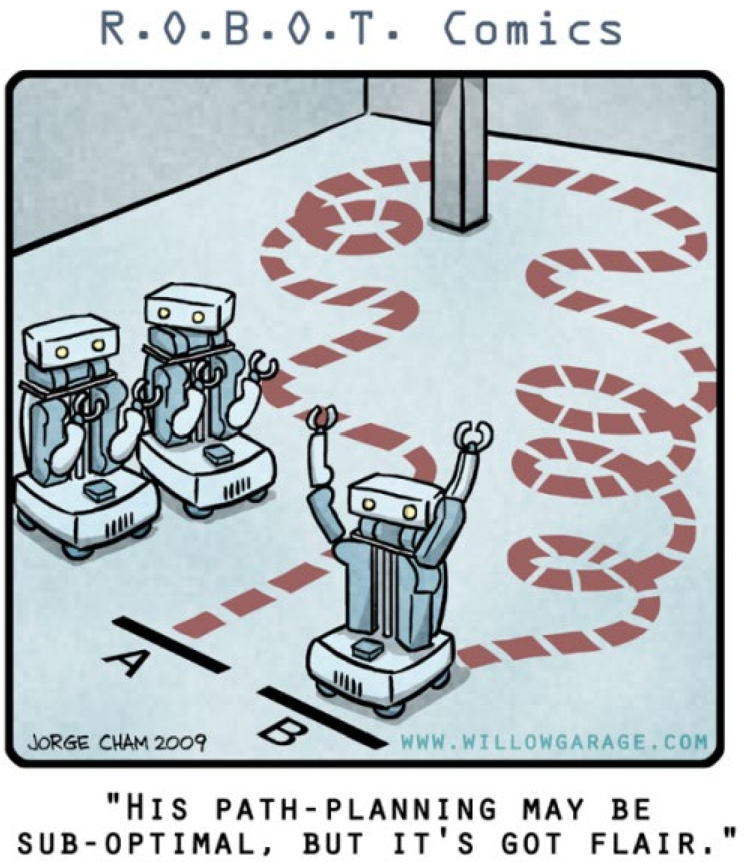


CS 4649/7649 Robot Intelligence: Planning

Constraints I: Constraint Programs; Arc Consistency

CS 4649/7649 – Asst. Prof. Matthew Gombolay

Slides adapted from:
6.034 Tomas Lozano Perez
16.410 Sertac Karaman
Russell and Norvig AIMA

Assignments

- Due Today, 8/31
  - Reading: Ch. 10
- Due Wednesday, 9/02
  - Pset 2 due at 11:59 PM Eastern
- Due Monday, 9/07
  - **Labor Day Holiday**

Outline

- Constraint Satisfaction Problems
- Solving CSPs
- Case Study: Scheduling

Constraint Satisfaction Problems (CSP)

Input: A CSP is a 3-tuple (i.e., triple) $\langle V, D, C \rangle$ where:

- V is a set of variables V_i
- D is a set of variable domains,
  - The domain of variable V_i is denoted D_i
- C is the set of constraints on assignments to V
  - Each constraint $C_j = \langle S_j, R_j \rangle$ specifies allowed variable assignments
  - S_j, the constraint's scope, is a subset of variables V
  - R_j, the constraint's relation, is a set of assignments to S_j

Output: A full assignment to V from elements of D such that all constraints C are satisfied.

Constraint Satisfaction Problems (CSP)

Example: “Provide one A and two B’s.”

- $V = \{A, B\}$, each with domains $D_i = \{1,2\}$
- $C = \begin{Bmatrix} \langle\{A, B\}, \{\langle 1,2\rangle, \langle 1,1\rangle\}\rangle, \\ \langle\{A, B\}, \{\langle 1,2\rangle, \langle 2,2\rangle\}\rangle \end{Bmatrix}$

“one A”

“two B’s”

- Output $\langle 1,2\rangle$

Conventions

- List scope in subscript
- Specify one constraint per scope

Example: “Provide one A and two B’s.”

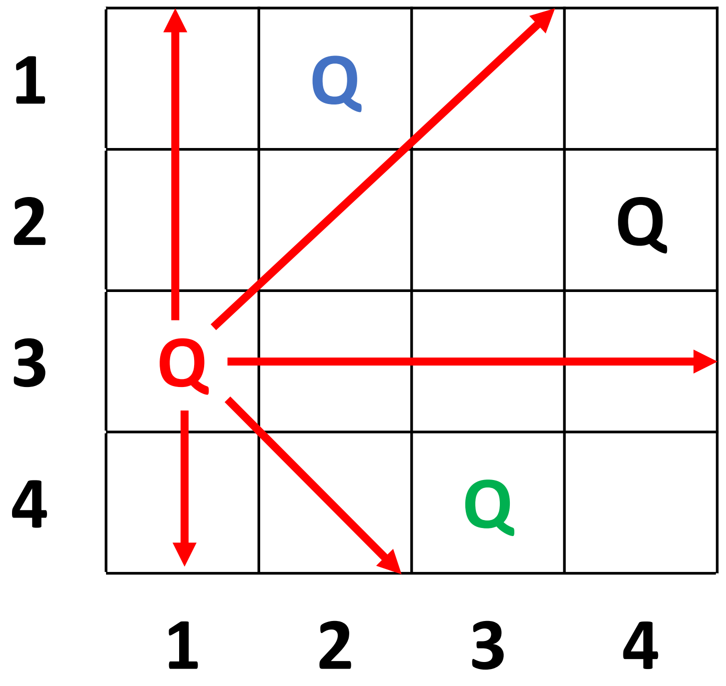
- $C = \{C_{AB}\}$
  $C_{AB} = \{\langle 1,2 \rangle\}$

- $C = \{C_A, C_B\}$
  $C_A = \{\langle 1 \rangle\}$
  $C_B = \{\langle 1 \rangle\}$

Good Encodings are Essential: 4 Queens

4 Queens Problem:

- Place 4 queens on a 4x4 chessboard so that no other queen can attack another



How big is the encoding?

Variables Chessboard positions

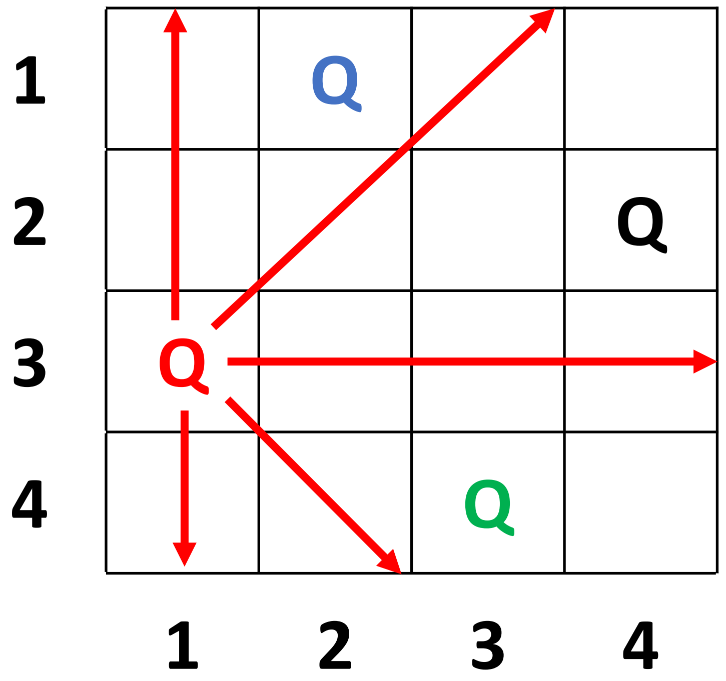
Domains Queen 1-4 or blank

Constraints Two positions on a line (vertical, horizontal, diagonal) cannot both be queens.

Good Encodings are Essential: 4 Queens

What is a better encoding?

- Assume one queen per column
- Determine what row each queen should be in



Variables: Q_1, Q_2, Q_3, Q_4

Domains: $\{1,2,3,4\}$

Constraints: $Q_i \neq Q_j$ "On different rows"

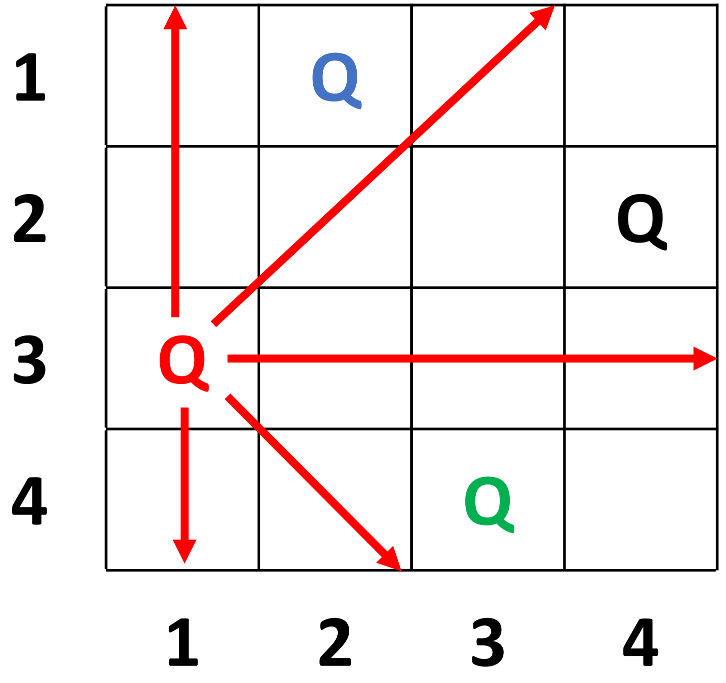
$|Q_i - Q_j| \neq |i - j|$ "Stay off diagonals"

Example: $C_{1,2} = \{(1,3),(1,4),(2,4),(3,1),(4,1),(4,2)\}$

Good Encodings are Essential: 4 Queens

What is a better encoding?

- Assume one queen per column
- Determine what row each queen should be in



Variables: Q_1, Q_2, Q_3, Q_4

Domains: $\{1,2,3,4\}$

Constraints: $Q_i \neq Q_j$ "On different rows

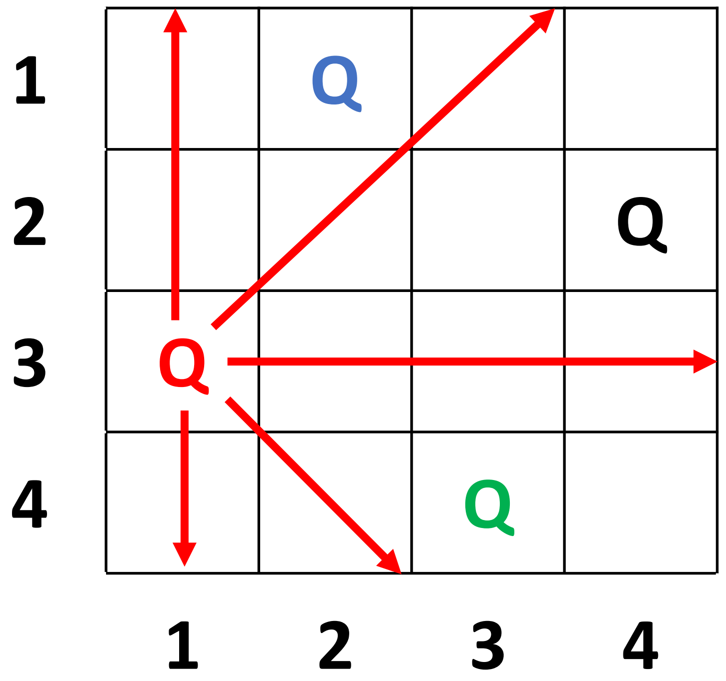
$|Q_i - Q_j| \neq |i - j|$ "Stay off diagonals"

Example: $C_{1,3} =?$

Good Encodings are Essential: 4 Queens

What is a better encoding?

- Assume one queen per column
- Determine what row each queen should be in



Variables: Q_1, Q_2, Q_3, Q_4

Domains: $\{1,2,3,4\}$

Constraints: $Q_i \neq Q_j$ "On different rows"

$|Q_i - Q_j| \neq |i - j|$ "Stay off diagonals"

Example: $C_{1,3} = \{(1,2), (1,4), (2,1), (2,3), (3,2), (3,4), (4,1), (4,3)\}$

General class of CSPs

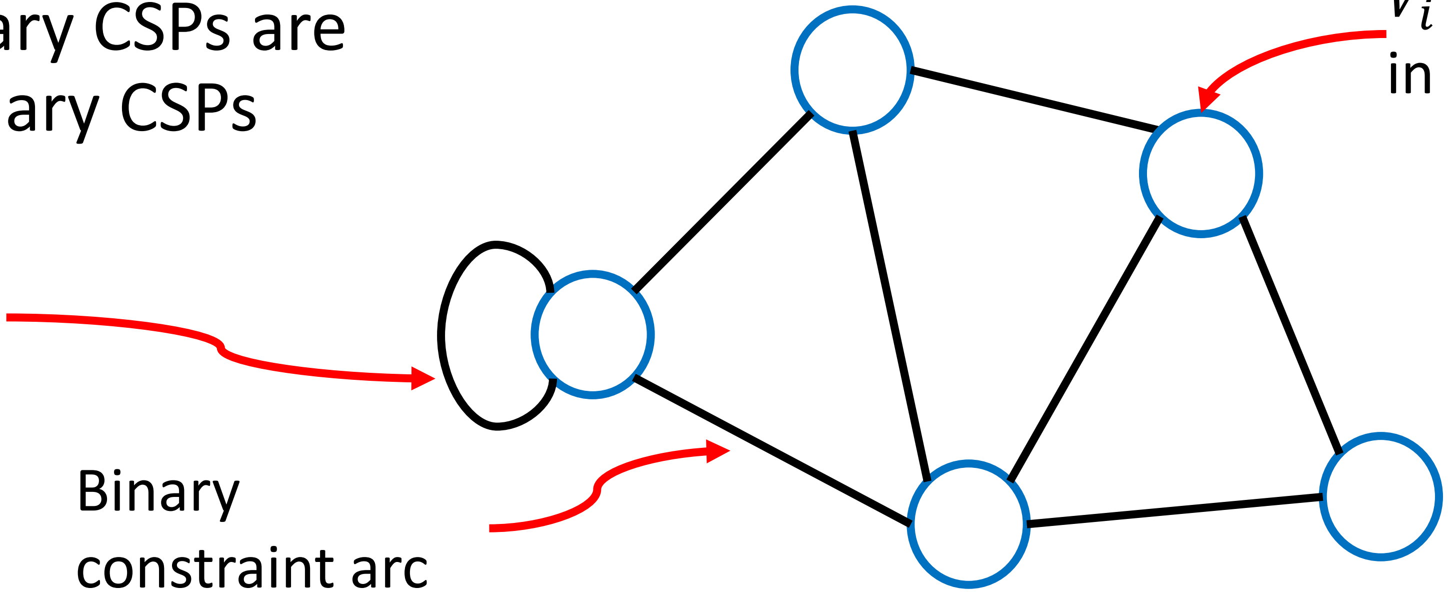
<u>Finite Domain, Binary CSPs</u>

- Each constraint relates at most two variables
- Each variable domain is finite

<u>Property:</u> all n-ary CSPs are reducible to binary CSPs

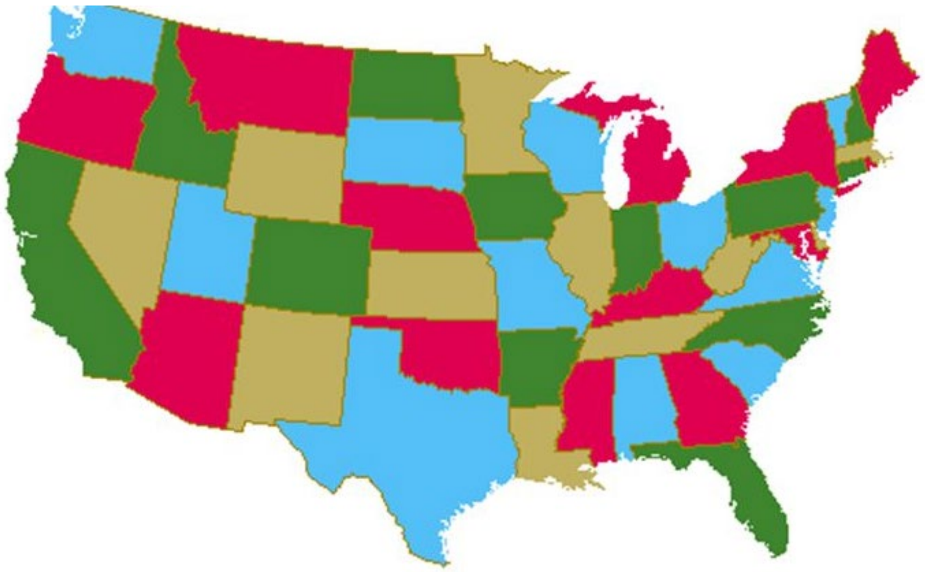
<u>Depict as a Constraint Graph</u>

- Nodes (vertices) are variables
- Arcs (edges) are binary constraints.



Example: Graph Coloring

Pick colors for map regions without coloring adjacent regions with the same color



Credit: Berniece Houston

Variables regions

Domains allowed colors

Constraints adjacent regions must have different colors

Outline

- Constraint Satisfaction Problems
- Solving CSPs
  - Arc-consistency and propagation
  - Analysis of constraint propagation (next lecture)
  - Search student (next lecture)
- Case Study: Scheduling

Good News / Bad News

Good news:

- Very general & interesting family of problems
- Problem formulation used extensively in autonomy and decision-making applications.

Bad new:

- Includes NP-Hard problems

Algorithmic Design Paradigm

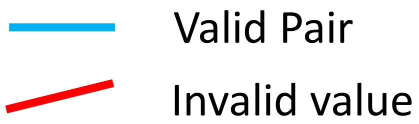
Solving CSPs involves a combination of:

Today

1. Inference:
   - Solve partially by eliminating values that cannot be part of any solution (constraint propagation)
   - Make implicit constraints explicit

Next Time

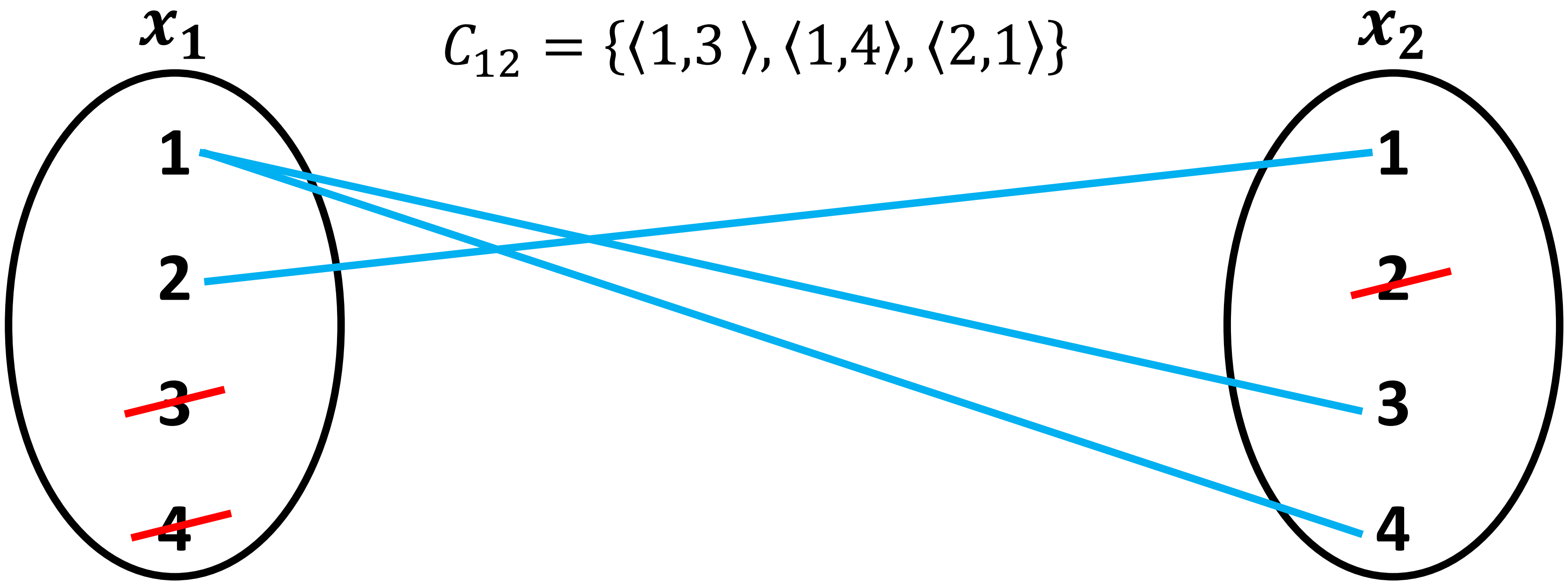
2. Search:
   - Try alternative assignments against constraints



Directed Arc Consistency

Idea: Eliminate values of variable domain that can never satisfy a specified constraint (an arc)

$C_{12} = \{\langle 1,3 \rangle, \langle 1,4 \rangle, \langle 2,1 \rangle\}$



Definition: arc $\langle x_i, x_j \rangle$ is arc consistent if $\langle x_i, x_j \rangle$ and $\langle x_j, x_i \rangle$ are directed arc consistent, i.e. $\forall a_i \in D_i, \exists\, a_j \in D_j | \langle a_i, a_j \rangle \in C_{ij}$.

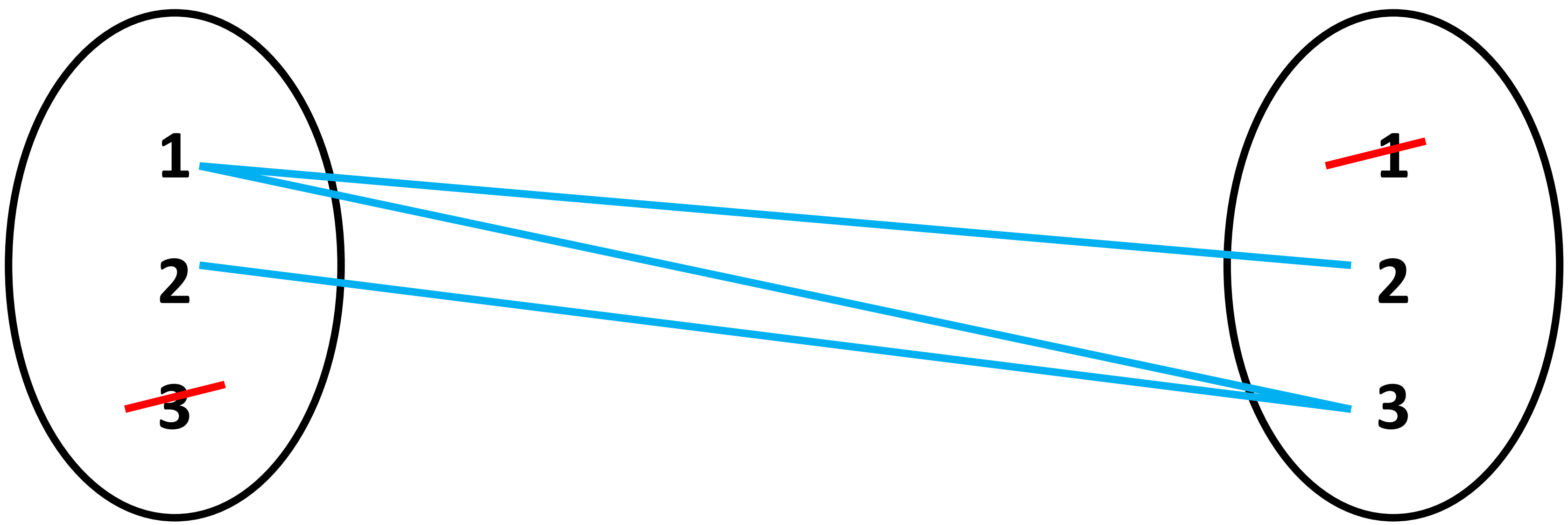
Valid Pair

Invalid value

Arc Consistency

“Assignments to X and Y are valid if the value for X is less than the value for y”

$$C_{XY} = \{\langle i, j\rangle | i < j, \forall i \in D_i, j \in D_j\}$$



Revise: A directed arc consistency procedure

Definition: $\langle x_i, x_j \rangle$ is arc consistent if $\forall a_i \in D_i, \exists\, a_j \in D_j | \langle a_i, a_j \rangle \in C_{ij}$

Revise(x_i, x_j)

Input: Variables x_i and x_j with domains D_i and D_j and constraint relation R_{ij}

Output: Pruned D_i such that x_i is directed arc-consistent relative to x_j

1. FOR each $a_i \in D_i$
2. IF there is no $a_j \in D_j$ such that $\langle a_i, a_j \rangle \in R_{ij}$ THEN
3. Delete a_i from D_i
4. ENDIF
5. ENDFOR

Full Arc Consistency over all Constraints via Constraint Propagation

Definition: $\langle x_i, x_j \rangle$ is arc consistent if $\forall a_i \in D_i, \exists\ a_j \in D_j | \langle a_i, a_j \rangle \in C_{ij}$

Constraint Propagation:

To achieve (directed) arc consistency over CSP:

1. For every arc C_{ij} in CSP, with tail domain D_i, call Revise(x_i, x_j)
2. Repeat until quiescence:
   - If an element was deleted from D_i, then repeat Step 1 //(AC-1)

Full Arc-Consistency via AC-1

AC-1 (CSP)
Input: CSP = $\langle X, D, C \rangle$
Output: CSP', the largest arc-consistent subset of CSP

1. WHILE (domains are being changed)
2. FOR every $C_{ij} \in C$
3. Revise(x_i, x_j)
4. Revise(x_j, x_i)
5. ENDFOR
6. ENDWHILE

Full Arc Consistency over all Constraints via Constraint Propagation

Definition: $\langle x_i, x_j \rangle$ is arc consistent if $\forall a_i \in D_i, \exists\ a_j \in D_j | \langle a_i, a_j \rangle \in C_{ij}$

Constraint Propagation:

To achieve (directed) arc consistency over CSP:

1. For every arc C_{ij} in CSP, with tail domain D_i, call Revise(x_i, x_j)
2. Repeat until quiescence:
   - If an element was deleted from D_i, then repeat Step 1 //(AC-1)

Mid-lecture Break

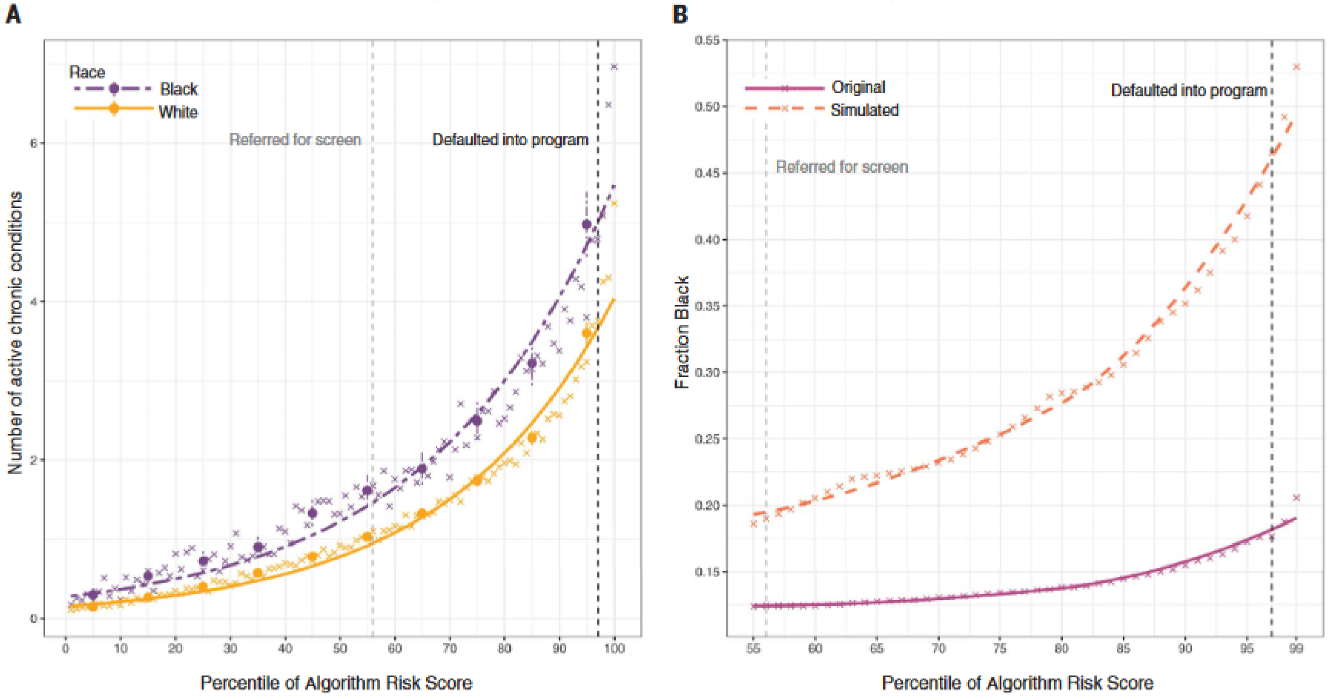


Fig. 1. Number of chronic illnesses versus algorithm-predicted risk, by race. (**A**) Mean number of chronic conditions by race, plotted against algorithm risk score. (**B**) Fraction of Black patients at or above a given risk score for the original algorithm ("original") and for a simulated scenario that removes algorithmic bias ("simulated": at each threshold of risk, defined at a given percentile on the *x* axis, healthier Whites above the threshold are replaced with less healthy Blacks below the threshold, until the marginal patient is equally healthy). The × symbols show risk percentiles by race; circles show risk deciles with 95% confidence intervals clustered by patient. The dashed vertical lines show the auto-identification threshold (the black line, which denotes the 97th percentile) and the screening threshold (the gray line, which denotes the 55th percentile).

Obermeyer, Z., Powers, B., Vogeli, C. and Mullainathan, S., 2019. Dissecting racial bias in an algorithm used to manage the health of populations. Science, 366(6464), pp.447-453.

New Study Blames Algorithm For Racial Discrimination, Ignores Physician Bias

Robert Pearl, M.D. Contributor
Healthcare

GETTY

A recent study published in *Science*, one of the world's leading academic journals, found that a predictive healthcare algorithm discriminated against black patients.

The tool, created by Optum, was designed to identify high-risk patients with untreated chronic diseases, thereby helping administrators re-distribute medical resources to those who'd benefit the most. But there was a glitch in the algorithm, according to researchers. Rather than ranking the needs of patients based on the severity or complexity of their illnesses, the algorithm relied on a surrogate measure: the cost of each patient's past treatments. The problem with that is black patients receive less care than white patients with the same disease burden and, therefore, account for less medical spending on average.

When the researchers went back and re-ranked patients by their illnesses (rather than the cost of care), the percentage of black patients who should have been enrolled in specialized care programs jumped from 17.7% to 46.5%.

Full Arc Consistency over all Constraints via Constraint Propagation

Definition: $\langle x_i, x_j \rangle$ is arc consistent if $\forall a_i \in D_i, \exists\ a_j \in D_j | \langle a_i, a_j \rangle \in C_{ij}$

Constraint Propagation:

To achieve (directed) arc consistency over CSP:

1. For every arc C_{ij} in CSP, with tail domain D_i, call Revise(x_i, x_j)
2. Repeat until quiescence:
   - If an element was deleted from D_i, then repeat Step 1 //(AC-1)
   - OR call Revise on each arc with head D_i //(AC-3)
     (use FIFO Q, remove duplicates)

Full Arc-Consistency via AC-3 (Waltz CP)

AC-3 (CSP)

Input: CSP = $\langle X, D, C \rangle$

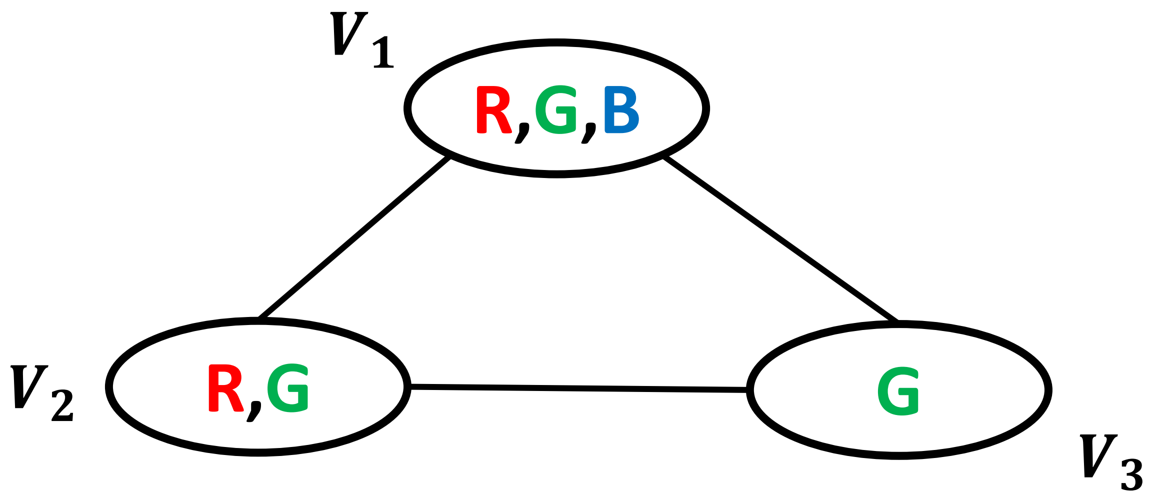
Output: CSP', the largest arc-consistent subset of CSP

1. FOR every $C_{ij} \in C$
2. $\quad Q \leftarrow Q \cup \{\langle x_i, x_j \rangle, \langle x_j, x_i \rangle\}$
3. ENDFOR
4. While $Q \neq \emptyset$
5. $\quad$ Select and delete arc $\langle x_i, x_j \rangle$ from Q
6. $\quad$ Revise(x_i, x_j)
7. $\quad$ IF Revise(x_i, x_j) caused a change to D_i
8. $\quad\quad Q \leftarrow Q \cup \{\langle x_k, x_i \rangle | k \neq i, k \neq j\}$
9. $\quad$ ENDIF
10. ENDWHILE

Constraint Propagation Example AC-3

Graph Coloring

(initial domains)

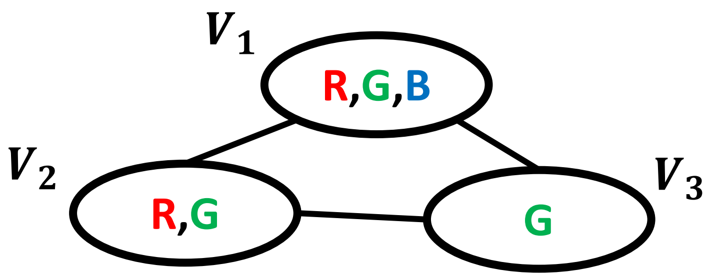


Each undirected arc denotes two directed arcs

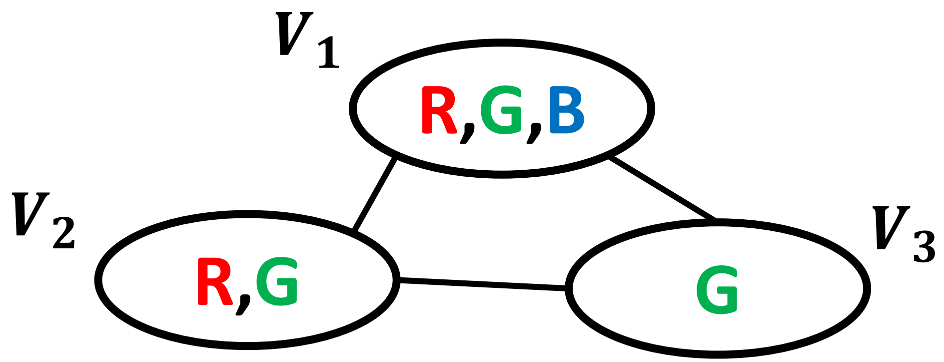
Constraint Propagation Example AC-3

Graph Coloring

(initial domains)



Arc examined	Value Deleted



Arcs to examine

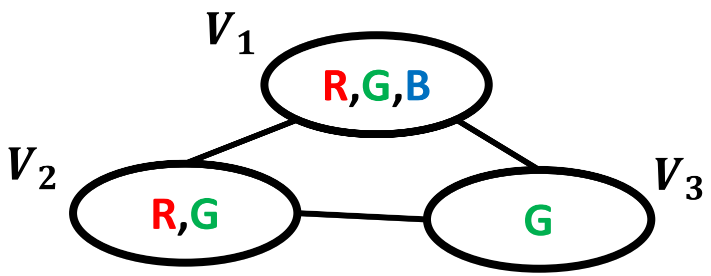
$a_{1-2}, a_{1-3}, a_{2-3}$

- Introduce queue of arcs to be examined
- Start by adding all arcs to the queue

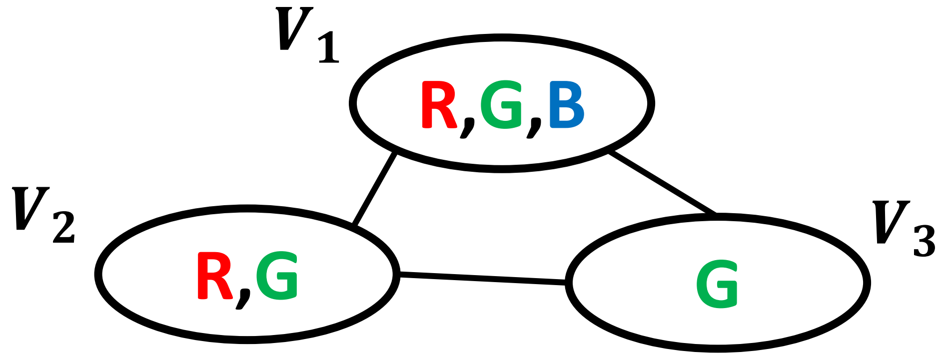
Constraint Propagation Example AC-3

Graph Coloring

(initial domains)



Arc examined	Value Deleted
$a_{1\to2}$	



Arcs to examine

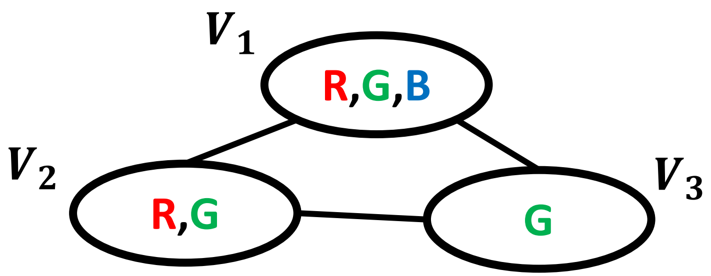
$a_{2\to1}, a_{1-3}, a_{2-3}$

- Delete unmentioned tail values

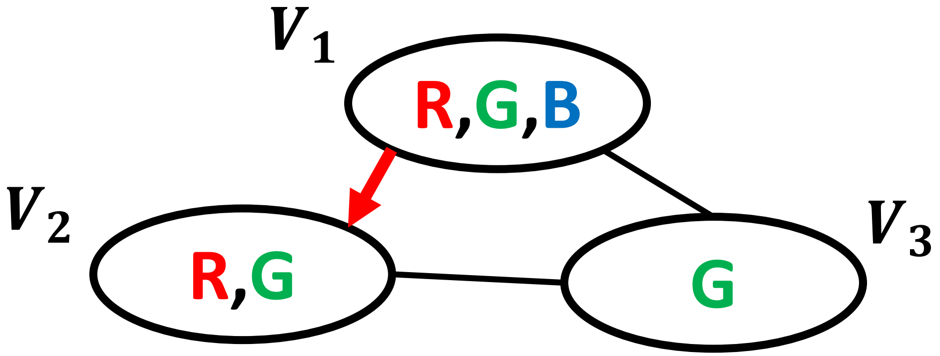
Constraint Propagation Example AC-3

Graph Coloring

(initial domains)



Arc examined	Value Deleted
$a_{1\to 2}$	none



Arcs to examine

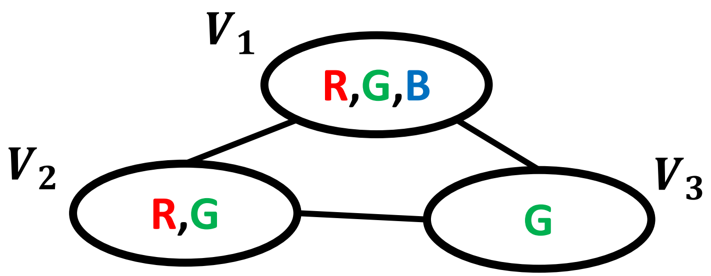
$a_{2\to 1}, a_{1-3}, a_{2-3}$

- Delete unmentioned tail values

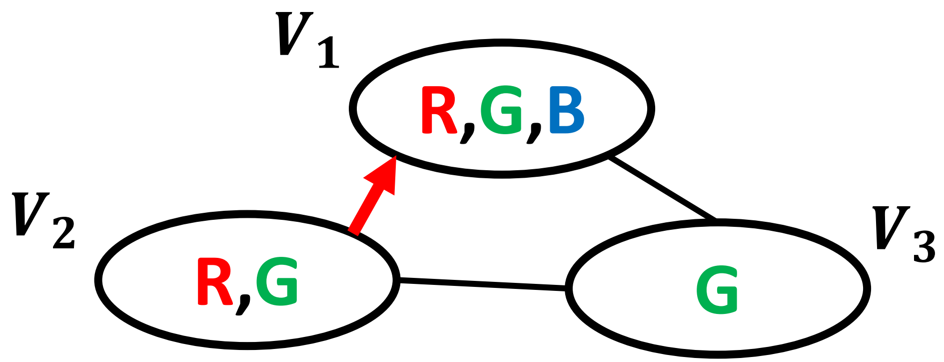
Constraint Propagation Example AC-3

Graph Coloring

(initial domains)



Arc examined	Value Deleted
$a_{1 \to 2}$	none
$a_{2 \to 1}$	



Arcs to examine

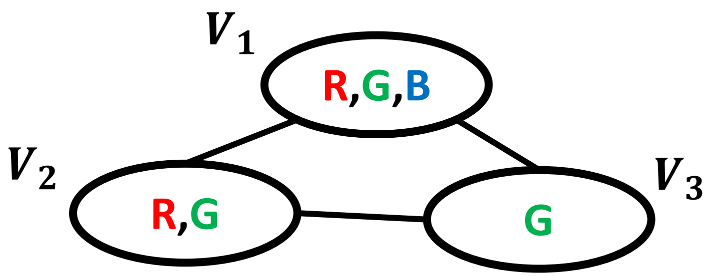
a_{1-3}, a_{2-3}

- Delete unmentioned tail values

Constraint Propagation Example AC-3

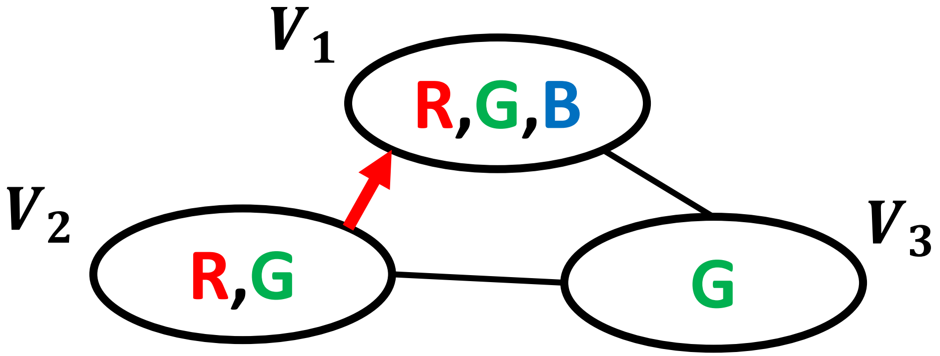
Graph Coloring

(initial domains)



Arc examined	Value Deleted
$a_{1\to2}$	none
$a_{2\to1}$	none

- Delete unmentioned tail values



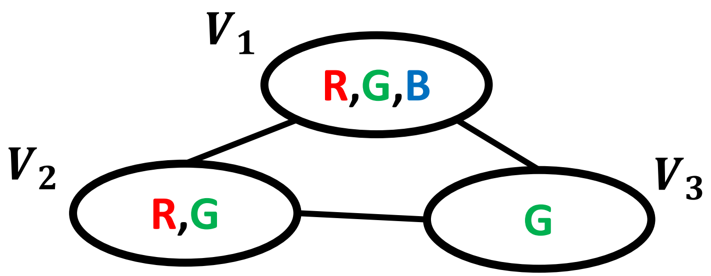
Arcs to examine

a_{1-3}, a_{2-3}

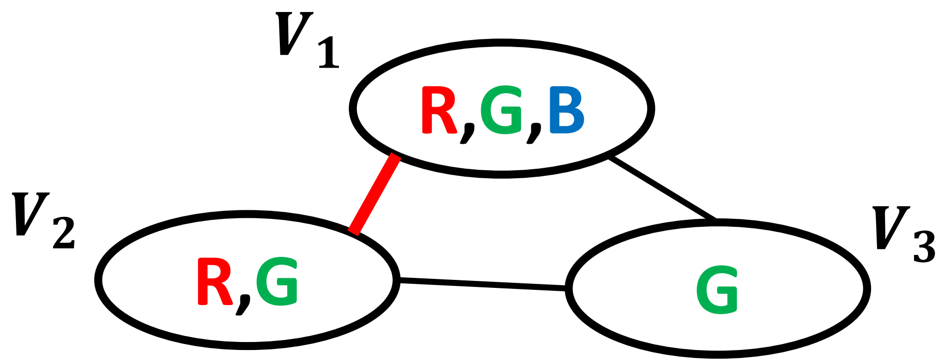
Constraint Propagation Example AC-3

Graph Coloring

(initial domains)



Arc examined	Value Deleted
a_{1-2}	none



Arcs to examine

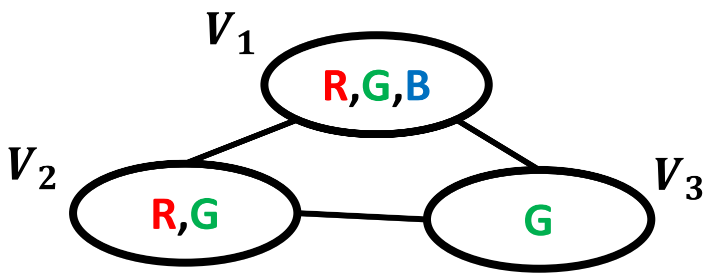
a_{1-3}, a_{2-3}

- Delete unmentioned tail values

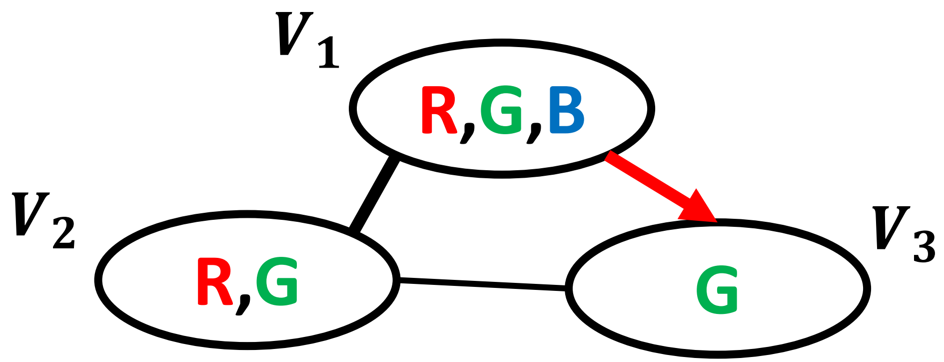
Constraint Propagation Example AC-3

Graph Coloring

(initial domains)



Arc examined	Value Deleted
a_{1-2}	none
$a_{1\rightarrow 3}$	



Arcs to examine

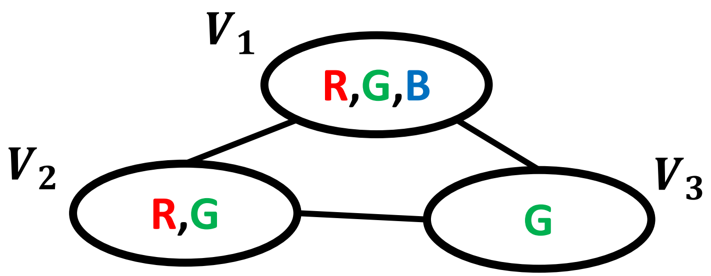
$a_{3\rightarrow 1}, a_{2-3}$

- Delete unmentioned tail values

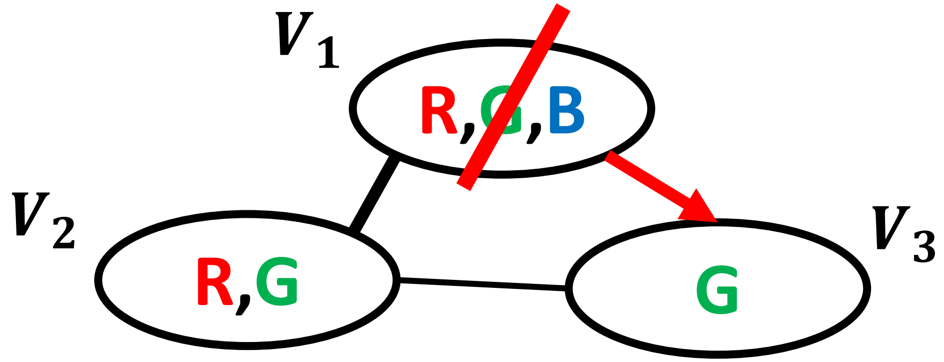
Constraint Propagation Example AC-3

Graph Coloring

(initial domains)



Arc examined	Value Deleted
a_{2-1}	none
$a_{1\to3}$	$V_1(G)$



Arcs to examine

$a_{3\to1}, a_{2-3}, a_{2\to1}, a_{3\to1}$

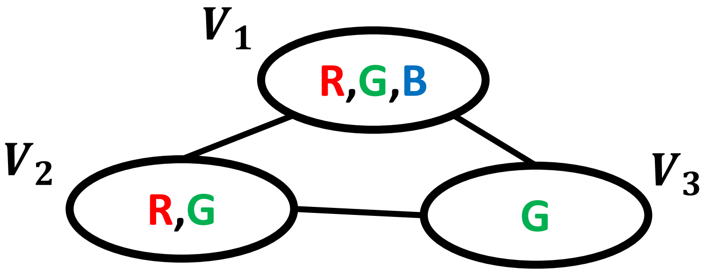
IF An element of a variable's domain is removed

THEN Add all arcs to that variable to the examination queue

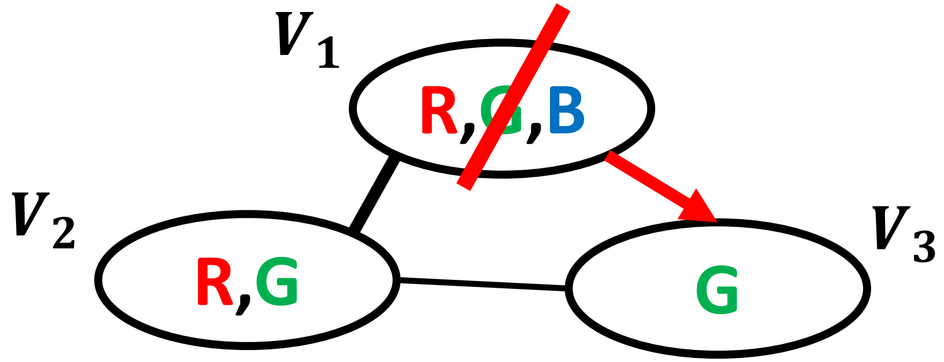
Constraint Propagation Example AC-3

Graph Coloring

(initial domains)



Arc examined	Value Deleted
a_{2-1}	none
$a_{1\to3}$	$V_1(G)$



Arcs to examine

$a_{3\to1}, a_{2-3}, a_{2\to1}, a_{3\to1}$

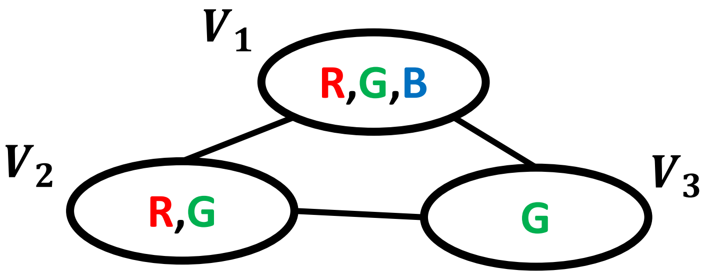
IF An element of a variable's domain is removed

THEN Add all arcs to that variable to the examination queue

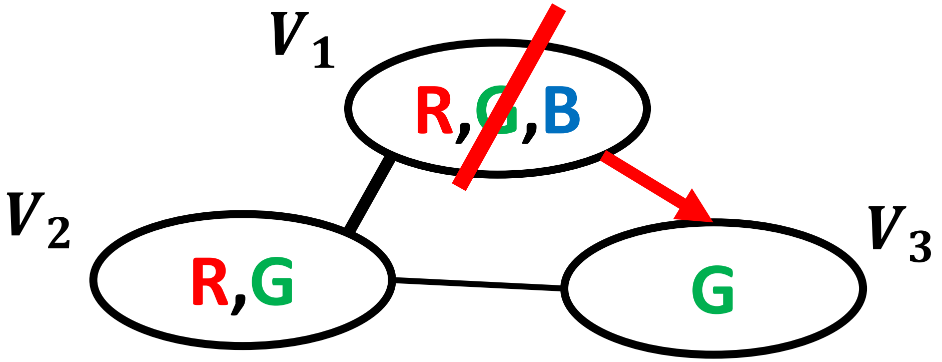
Constraint Propagation Example AC-3

Graph Coloring

(initial domains)



Arc examined	Value Deleted
a_{2-1}	none
$a_{1\to3}$	$V_1(G)$



Arcs to examine

$a_{3\to1}, a_{2-3}, a_{2\to1}, \cancel{a_{3\to1}}$

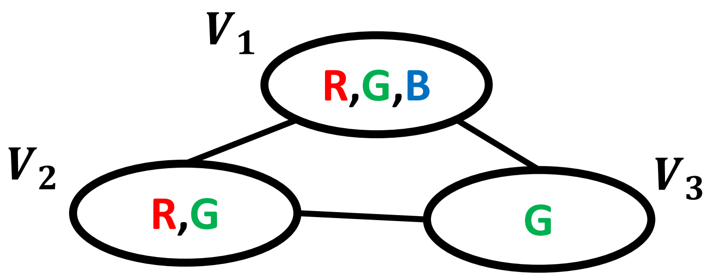
IF An element of a variable's domain is removed

THEN Add all arcs to that variable to the examination queue

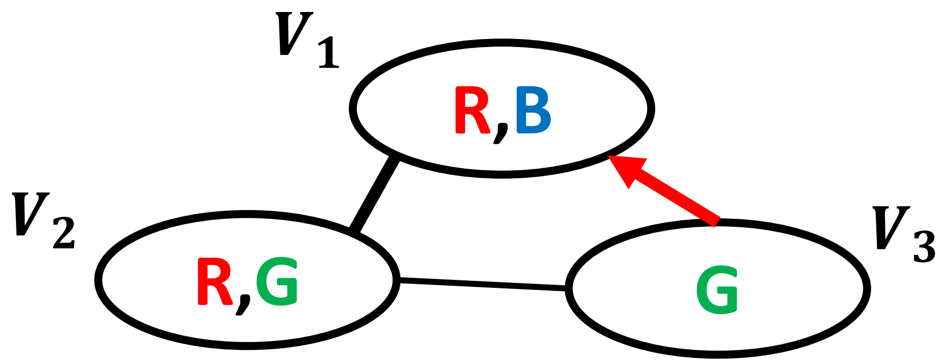
Constraint Propagation Example AC-3

Graph Coloring

(initial domains)



Arc examined	Value Deleted
a_{2-1}	none
$a_{1\to3}$	$V_1(G)$
$a_{3\to1}$	



Arcs to examine

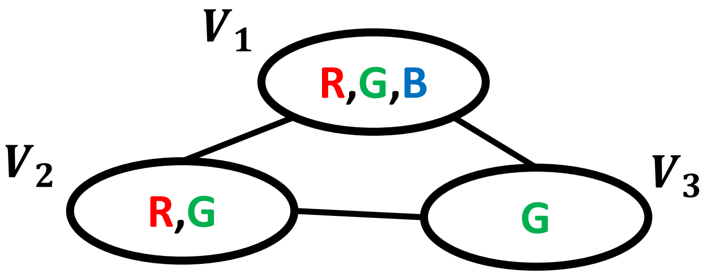
$a_{2-3}, a_{2\to1}$

IF An element of a variable's domain is removed
THEN Add all arcs to that variable to the examination queue

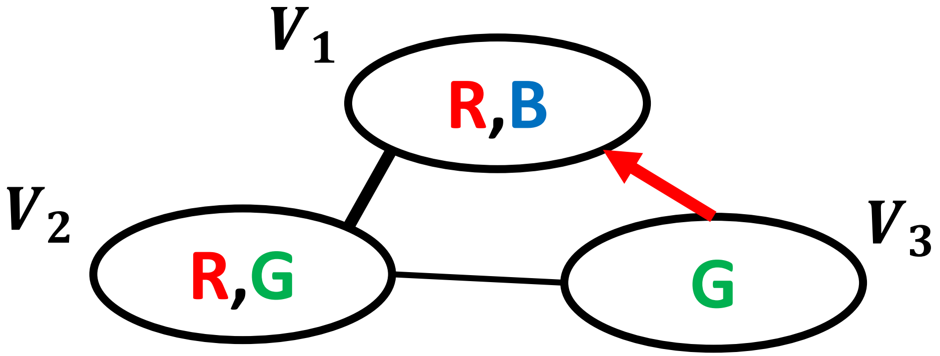
Constraint Propagation Example AC-3

Graph Coloring

(initial domains)



Arc examined	Value Deleted
a_{2-1}	none
$a_{1\to3}$	$V_1(G)$
$a_{3\to1}$	none



Arcs to examine

$a_{2-3}, a_{2\to1}$

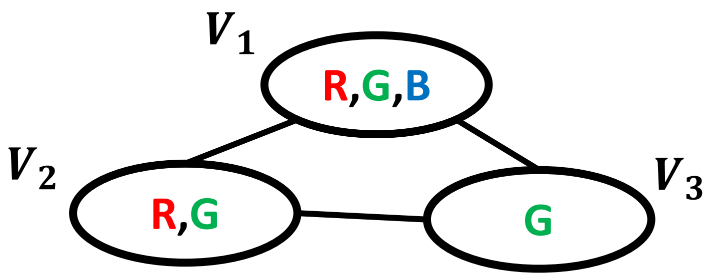
IF An element of a variable's domain is removed

THEN Add all arcs to that variable to the examination queue

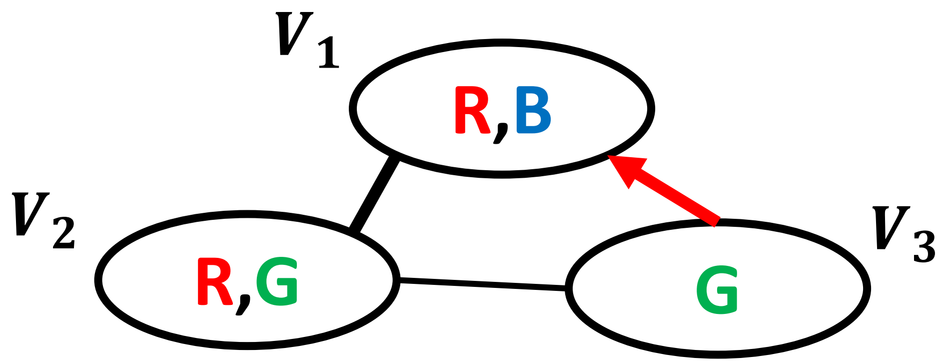
Constraint Propagation Example AC-3

Graph Coloring

(initial domains)



Arc examined	Value Deleted
a_{2-1}	none
a_{1-3}	$V_1(G)$



Arcs to examine

$a_{2-3}, a_{2\rightarrow1}$

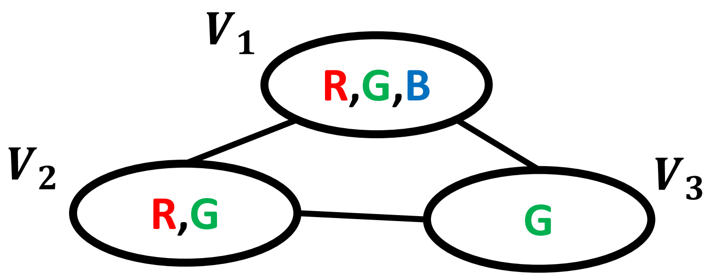
IF An element of a variable's domain is removed

THEN Add all arcs to that variable to the examination queue

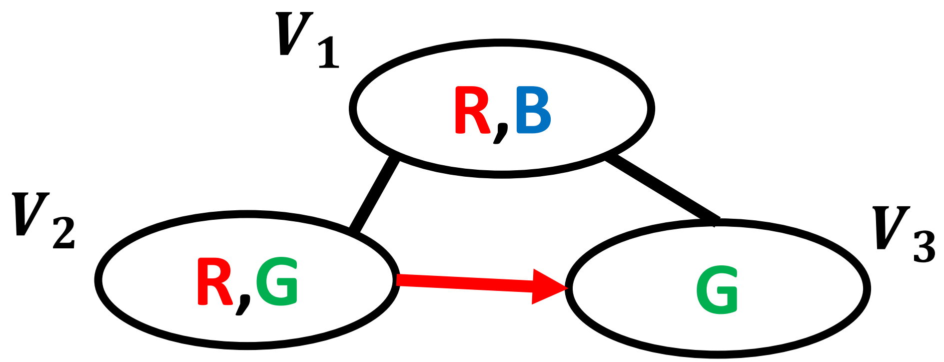
Constraint Propagation Example AC-3

Graph Coloring

(initial domains)



Arc examined	Value Deleted
a_{2-1}	none
a_{1-3}	$V_1(G)$
$a_{2\rightarrow3}$	



Arcs to examine

$a_{3\rightarrow2}, a_{2\rightarrow1}$

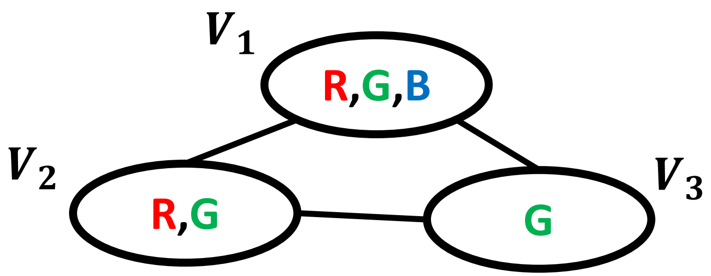
IF An element of a variable's domain is removed

THEN Add all arcs to that variable to the examination queue

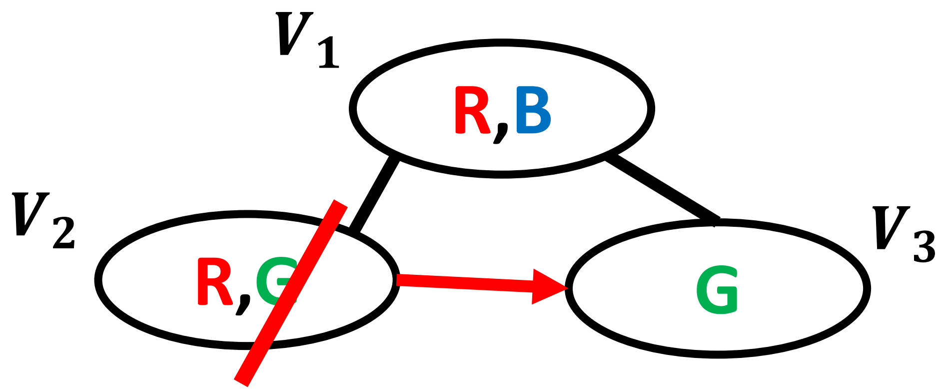
Constraint Propagation Example AC-3

Graph Coloring

(initial domains)



Arc examined	Value Deleted
a_{2-1}	none
a_{1-3}	$V_1(G)$
$a_{2\to3}$	



Arcs to examine

$a_{3\to2}, a_{2\to1}$

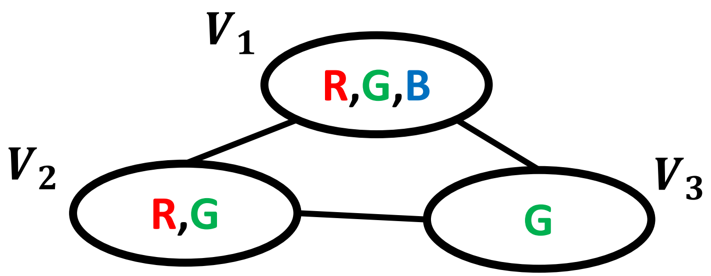
IF An element of a variable's domain is removed

THEN Add all arcs to that variable to the examination queue

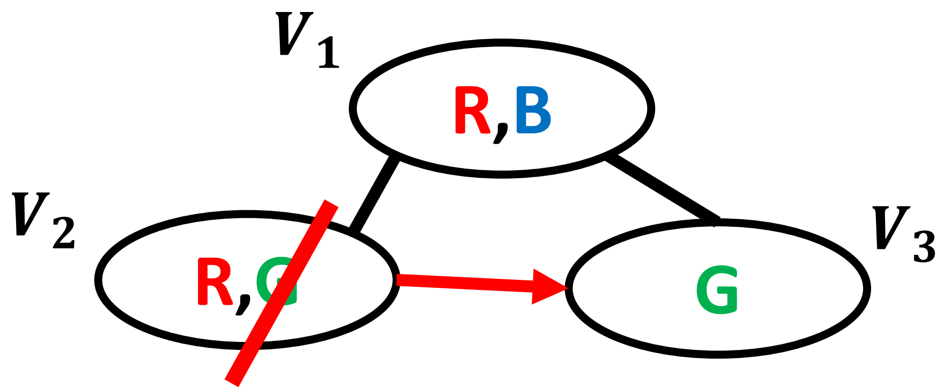
Constraint Propagation Example AC-3

Graph Coloring

(initial domains)



Arc examined	Value Deleted
a_{2-1}	none
a_{1-3}	$V_1(G)$
$a_{2\to3}$	$V_2(G)$



Arcs to examine

$a_{3\to2}, a_{2\to1}, a_{1\to2}$

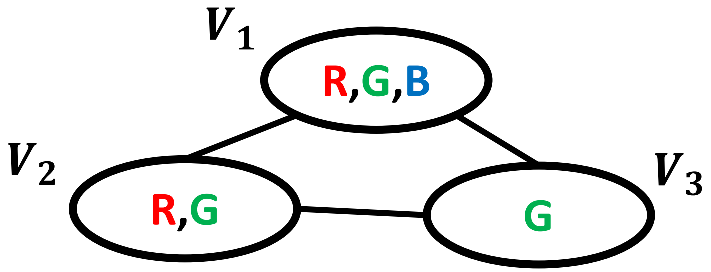
IF An element of a variable's domain is removed

THEN Add all arcs to that variable to the examination queue

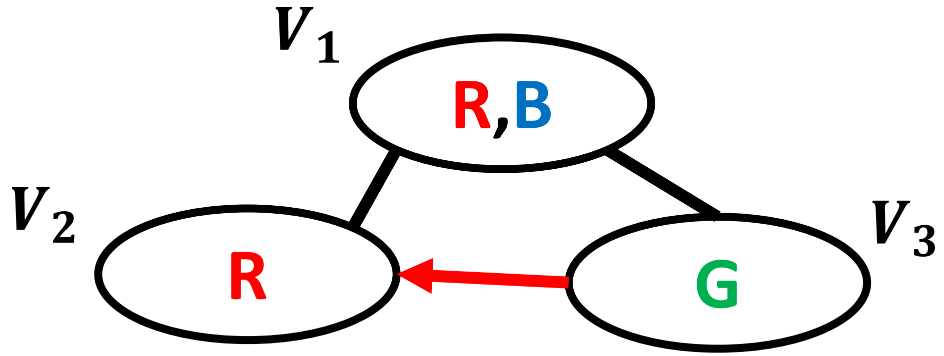
Constraint Propagation Example AC-3

Graph Coloring

(initial domains)



Arc examined	Value Deleted
a_{2-1}	none
a_{1-3}	$V_1(G)$
$a_{2\to3}$	$V_2(G)$
$a_{3\to2}$	



Arcs to examine

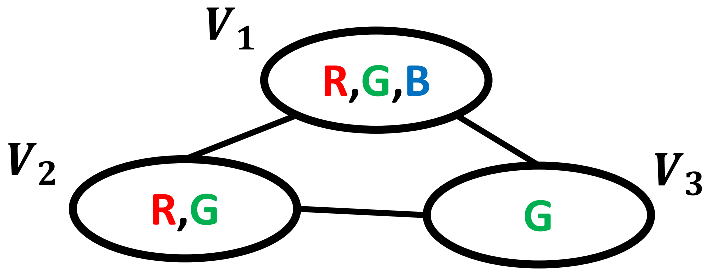
$a_{2\to1}, a_{1\to2}$

IF An element of a variable's domain is removed
THEN Add all arcs to that variable to the examination queue

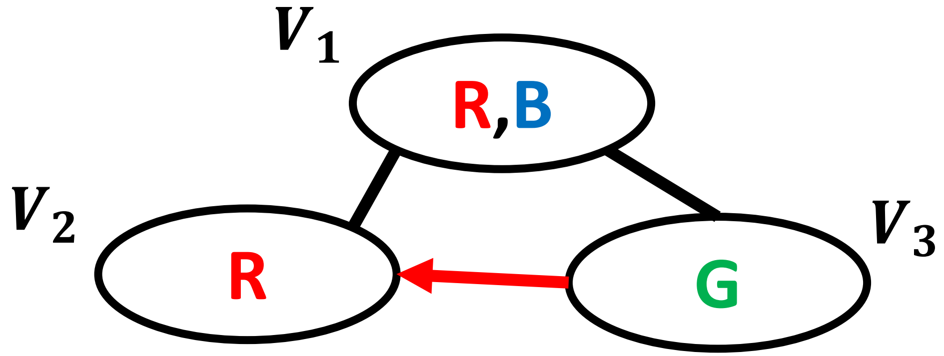
Constraint Propagation Example AC-3

Graph Coloring

(initial domains)



Arc examined	Value Deleted
a_{2-1}	none
a_{1-3}	$V_1(G)$
$a_{2\to3}$	$V_2(G)$
$a_{3\to2}$	none



Arcs to examine

$a_{2\to1}, a_{1\to2}$

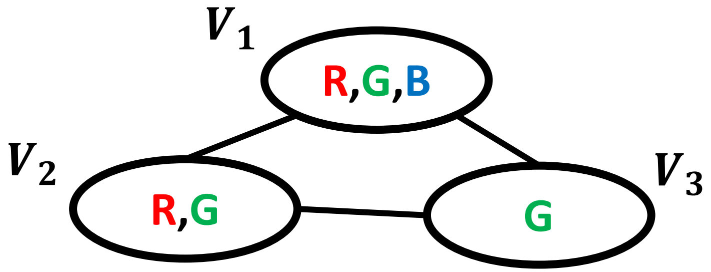
IF An element of a variable's domain is removed

THEN Add all arcs to that variable to the examination queue

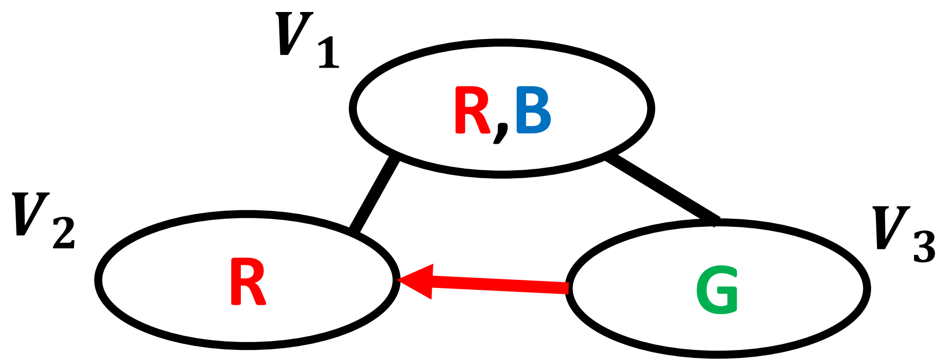
Constraint Propagation Example AC-3

Graph Coloring

(initial domains)



Arc examined	Value Deleted
a_{2-1}	none
a_{1-3}	$V_1(G)$
a_{2-3}	$V_2(G)$



Arcs to examine

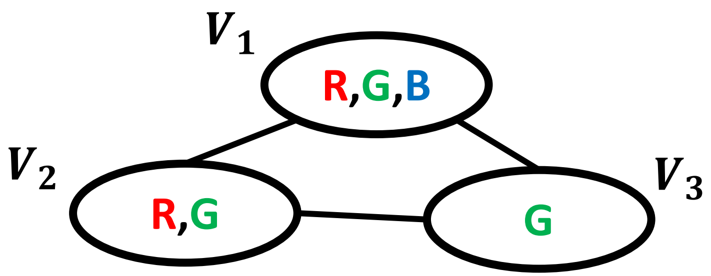
$a_{2\to 1}, a_{1\to 2}$

IF An element of a variable's domain is removed
THEN Add all arcs to that variable to the examination queue

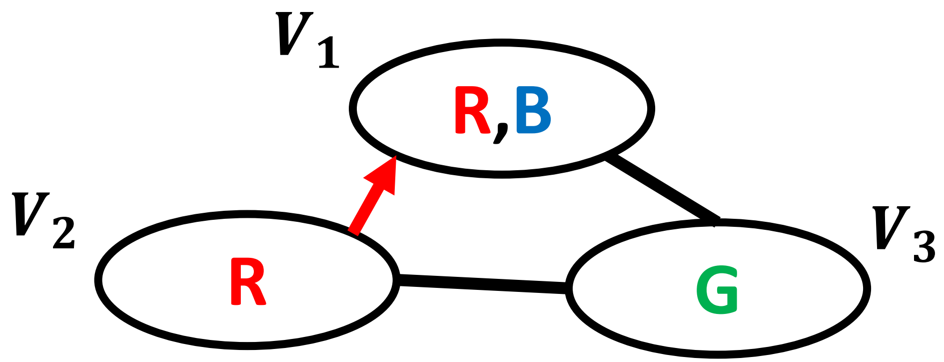
Constraint Propagation Example AC-3

Graph Coloring

(initial domains)



Arc examined	Value Deleted
a_{2-1}	none
a_{1-3}	$V_1(G)$
a_{2-3}	$V_2(G)$
$a_{2\to 1}$	



Arcs to examine

$a_{1\to 2}$

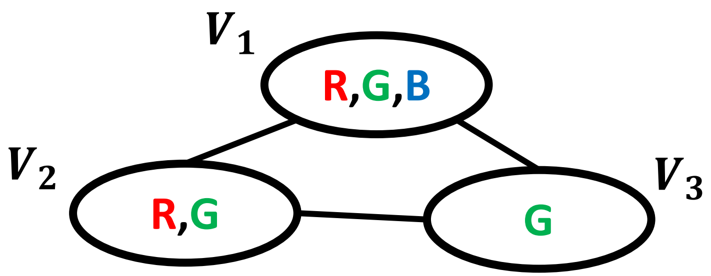
IF An element of a variable's domain is removed

THEN Add all arcs to that variable to the examination queue

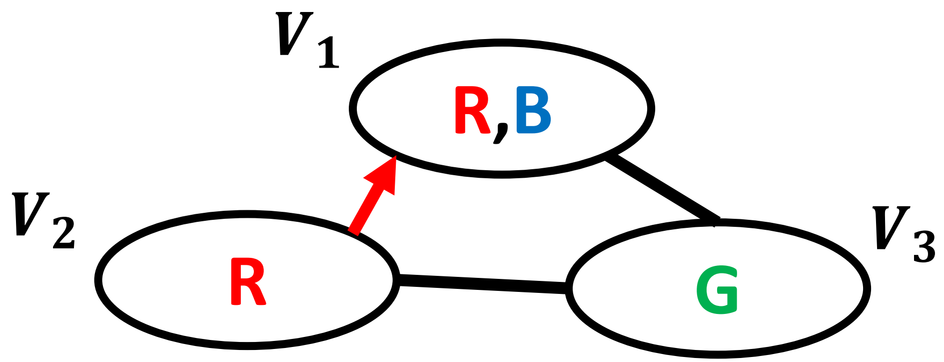
Constraint Propagation Example AC-3

Graph Coloring

(initial domains)



Arc examined	Value Deleted
a_{2-1}	none
a_{1-3}	$V_1(G)$
a_{2-3}	$V_2(G)$
$a_{2\to1}$	none



Arcs to examine

$a_{1\to2}$

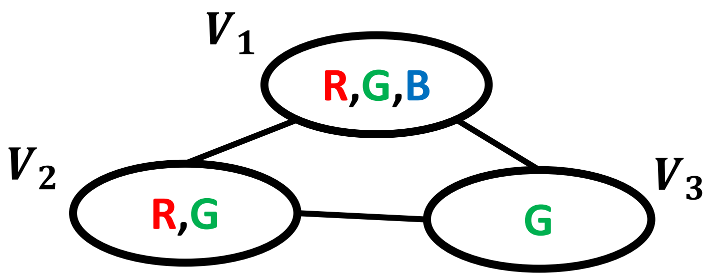
IF An element of a variable's domain is removed

THEN Add all arcs to that variable to the examination queue

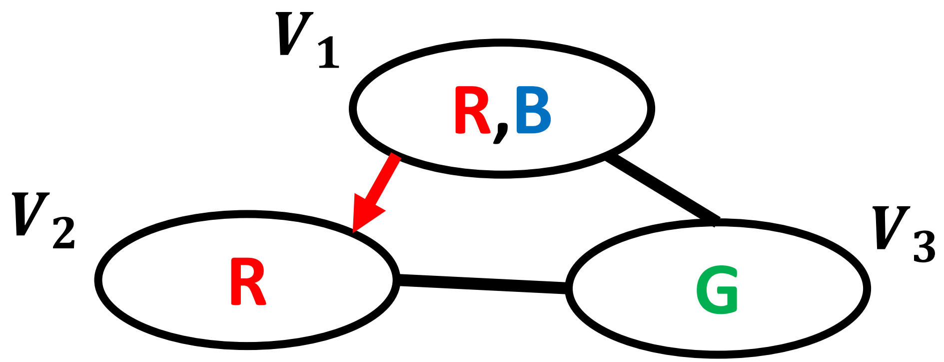
Constraint Propagation Example AC-3

Graph Coloring

(initial domains)



Arc examined	Value Deleted
a_{2-1}	none
a_{1-3}	$V_1(G)$
a_{2-3}	$V_2(G)$
$a_{2\rightarrow 1}$	none
$a_{1\rightarrow 2}$	



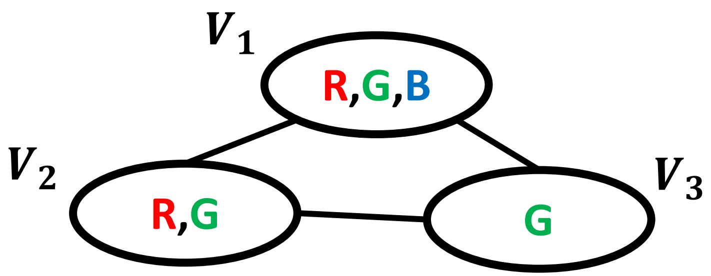
Arcs to examine

IF An element of a variable's domain is removed
THEN Add all arcs to that variable to the examination queue

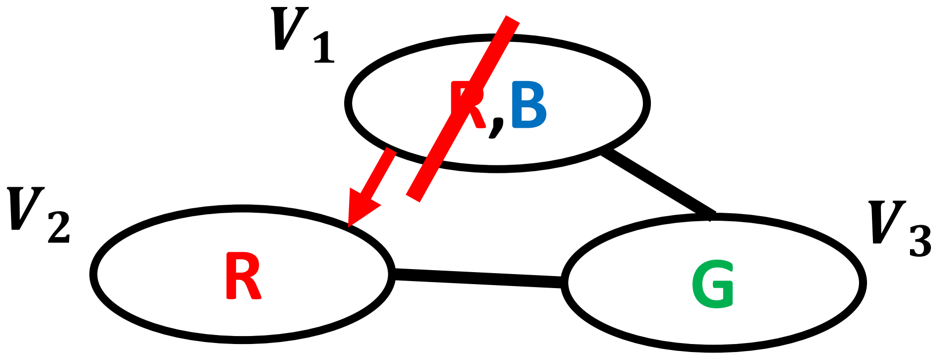
Constraint Propagation Example AC-3

Graph Coloring

(initial domains)



Arc examined	Value Deleted
a_{2-1}	none
a_{1-3}	$V_1(G)$
a_{2-3}	$V_2(G)$
$a_{2\rightarrow1}$	none
$a_{1\rightarrow2}$	$V_1(R)$



Arcs to examine

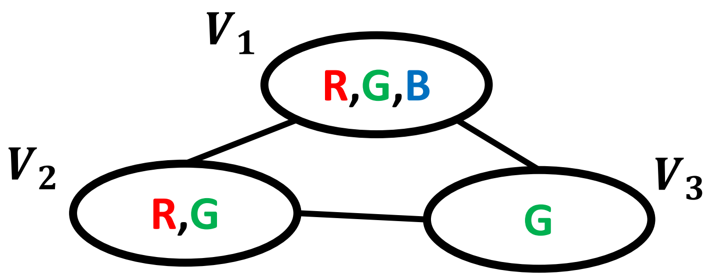
IF An element of a variable's domain is removed

THEN Add all arcs to that variable to the examination queue

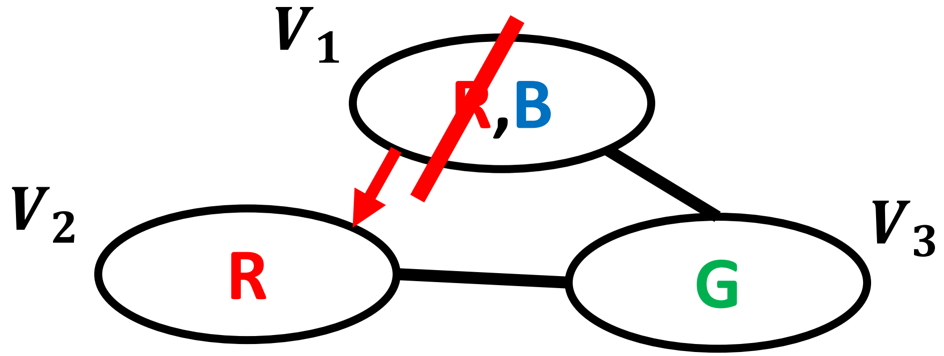
Constraint Propagation Example AC-3

Graph Coloring

(initial domains)



Arc examined	Value Deleted
a_{2-1}	none
a_{1-3}	$V_1(G)$
a_{2-3}	$V_2(G)$
$a_{2\to 1}$	none
$a_{1\to 2}$	$V_1(R)$



Arcs to examine

$a_{2\to 1}, a_{3\to 1}$

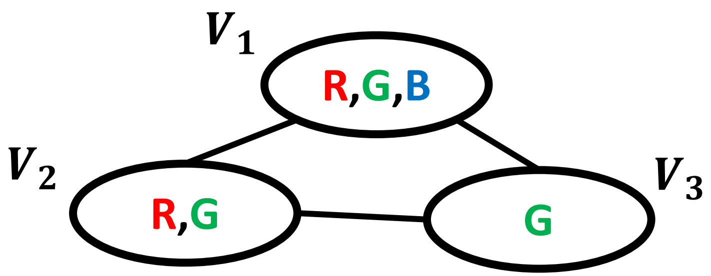
IF An element of a variable's domain is removed

THEN Add all arcs to that variable to the examination queue

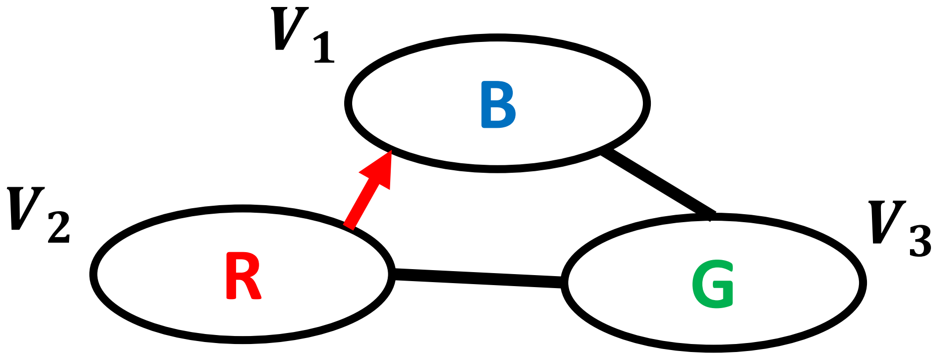
Constraint Propagation Example AC-3

Graph Coloring

(initial domains)



Arc examined	Value Deleted
a_{2-1}	none
a_{1-3}	$V_1(G)$
a_{2-3}	$V_2(G)$
a_{2-1}	$V_1(R)$
$a_{2\to 1}$	



Arcs to examine

$a_{3\to 1}$

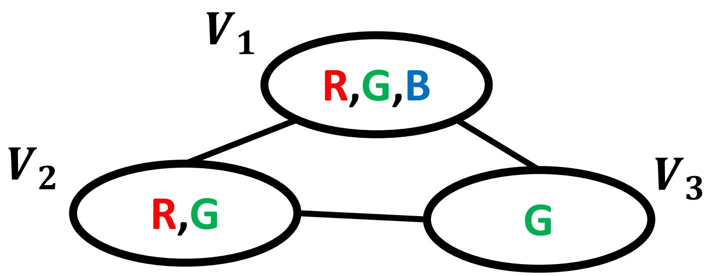
IF An element of a variable's domain is removed

THEN Add all arcs to that variable to the examination queue

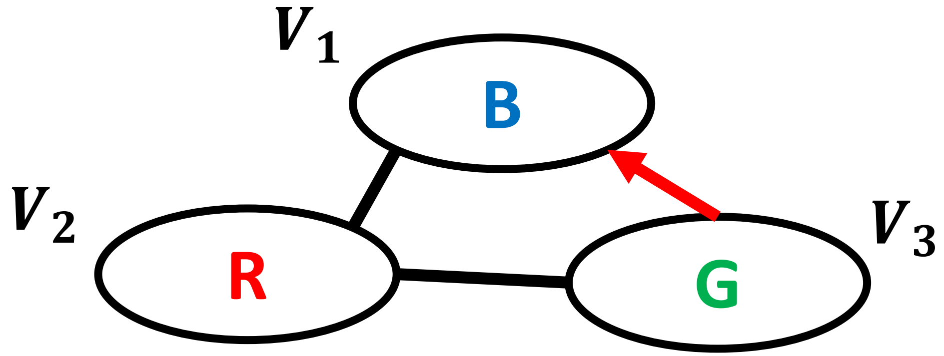
Constraint Propagation Example AC-3

Graph Coloring

(initial domains)



Arc examined	Value Deleted
a_{2-1}	none
a_{1-3}	$V_1(G)$
a_{2-3}	$V_2(G)$
a_{2-1}	$V_1(R)$
$a_{2\to1}$	none
$a_{3\to1}$	



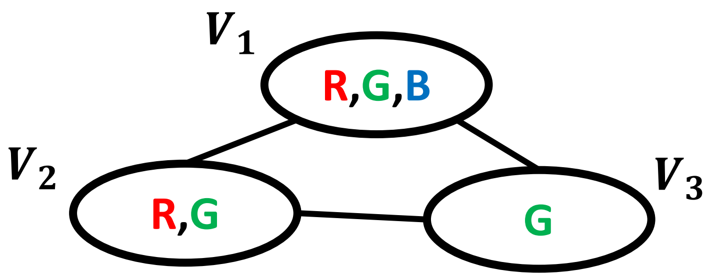
Arcs to examine

IF An element of a variable's domain is removed
THEN Add all arcs to that variable to the examination queue

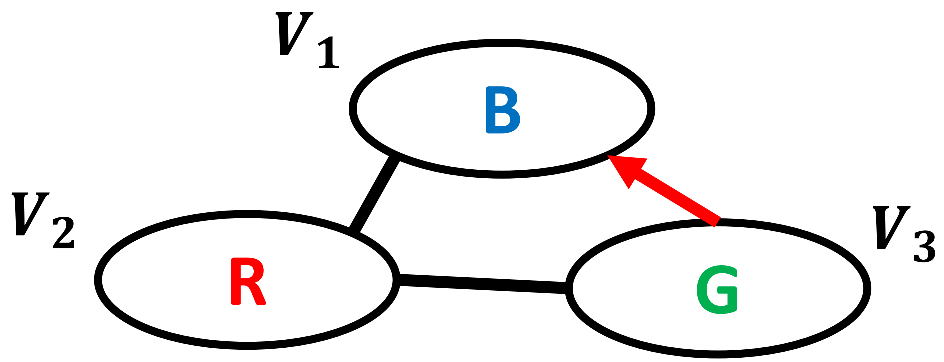
Constraint Propagation Example AC-3

Graph Coloring

(initial domains)



Arc examined	Value Deleted
a_{2-1}	none
a_{1-3}	$V_1(G)$
a_{2-3}	$V_2(G)$
a_{2-1}	$V_1(R)$
$a_{2\rightarrow 1}$	none
$a_{3\rightarrow 1}$	none



Arcs to examine

IF examination of queue is empty
THEN arc (pairwise) consistent

To solve CSPs, we combine arc consistency and search

Today

1. Arc consistency (constraint propagation),
   - Eliminates values that are shown locally to not be a part of any solution
2. Search
   - Explores consequences of committing to particular assignments
   - Methods incorporating search:
     - Standard Search
     - Backtrack Search (BT)
     - BT with Forward Checking (FC)
     - Dynamic Variable Ordering (DVO)
     - Iterative Repair
     - Back jumping (BJ)

Next Time

Outline

- Constraint Satisfaction Problems
- Solving CSPs
  - Arc-consistency and propagation
  - Analysis of constraint propagation (next lecture)
  - Search student (next lecture)
- Case Study: Scheduling

Real World Example: Scheduling as a CSP

Choose time of activities:

- Shifts of nurses in the Labor & Delivery Ward
- Classes taken for a degree
- Tennis matches for Wimbledon

Variables	activities
Domains	possible start times (or “chunks” of time)
Constraints	1) Activities that use the same resource cannot overlap in time
	2) Prerequisites are satisfied (round of 128 before 64)

Case Study: Course Scheduling

Given:

- 32 required courses
- 8 terms (Fall 2018, Spring 2019, ...)

Find: a feasible schedule

Constraints:

  - Pre-requisites are satisfied
  - Courses offered only during certain times
  - Limited number of courses can be taken per term (e.g., 4) and
  - Avoid time conflicts between courses

Note: traditional CSPs are not for expressing (soft) <u>preferences</u>

e.g., minimize difficulty, balance subject areas, etc.

But see recent research on valued CSPs!

Alternate formulations for variables, domains

Variables

A. 1 variable per Term

(Fall ’18), (Spring ’19), …

Domains

All legal combinations of 4 courses, all offered during that term

A. 1 variable per Term-Slot

Subdivide each term

into 4 course slots

(Fall ‘18, 1), (Fall ‘18, 2),

(Fall ‘18, 3), (Fall ‘18, 4)

All courses offered during that term.

A. 1 variable per Course

Terms or term-slots

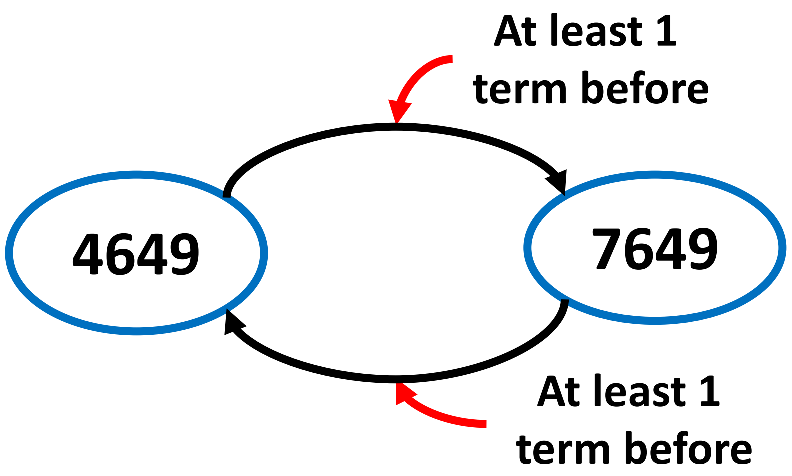
Term-slots make it easier to express the constraint limiting the number of courses per term

Encoding Constraints

Assume: Variables = Courses, Domains = term-slots

Constraints:

Pre-requisite →

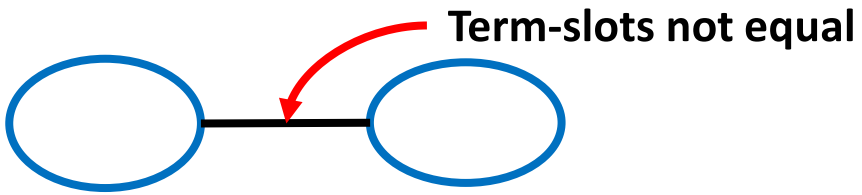


For pairs of courses that must be ordered

Courses offered only during certain terms →

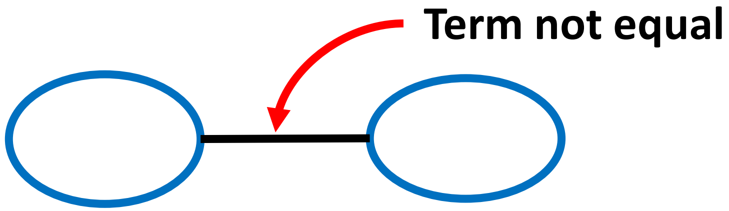
Filter domain

Limit # courses →



Use term-slots once

Avoid time conflicts →



For course pairs offered at same time or overlapping times

What you should be able to do…

- Provide the definition of a CSP as a 3-tuple
- Manually follow (i.e., by hand) and implement (i.e., write a computer program to perform) AC-1 and AC-3
- Model real-world problems as a CSP