Constraint Satisfaction Problems: Formulation, Arc Consistency & Propagation

Brian C. Williams

16.410-13

October 13th, 2004

Slides adapted from:
6.034 Tomas Lozano Perez
and AIMA Stuart Russell & Peter Norvig

Reading Assignments: Constraints

Readings:

- Lecture Slides (most material in slides only, READ ALL).
- AIMA Ch. 5 – Constraint Satisfaction Problems (CSPs)
- AIMA Ch. 24.4 pp. 881-884 – Visual Interpretation of line drawings as solving CSPs.

Problem Set #5:

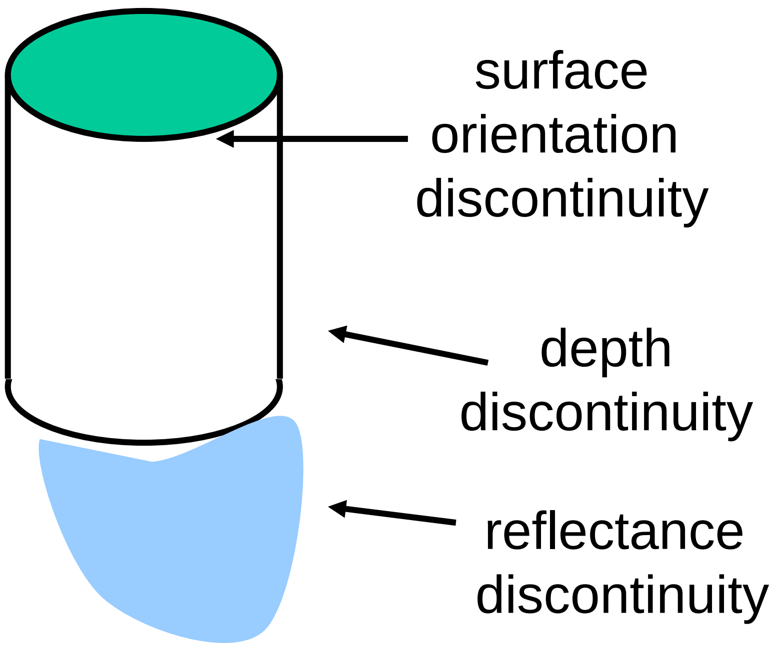
- Covers constraints.
- Online.
- Out Thursday morning, October 14th.
- Due Wednesday, October 20th.
- Get started early!

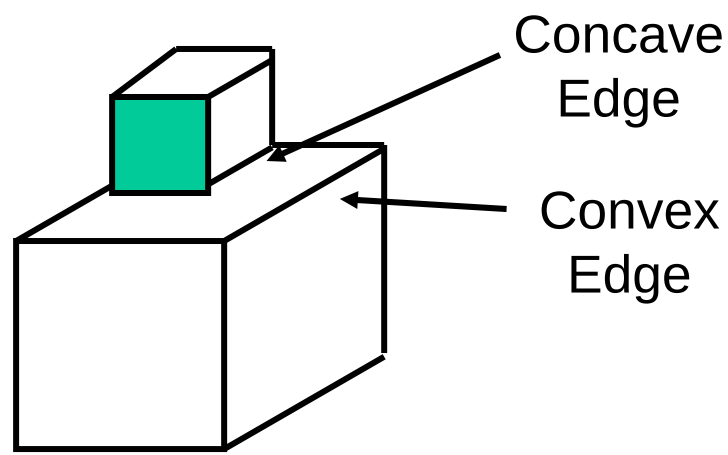
Outline

- Constraint satisfaction problems (CSP)
- Solving CSPs
  - Arc-consistency and propagation
  - Analysis of constraint propagation
  - Search (next lecture)

Line Labeling In Visual Interpretation

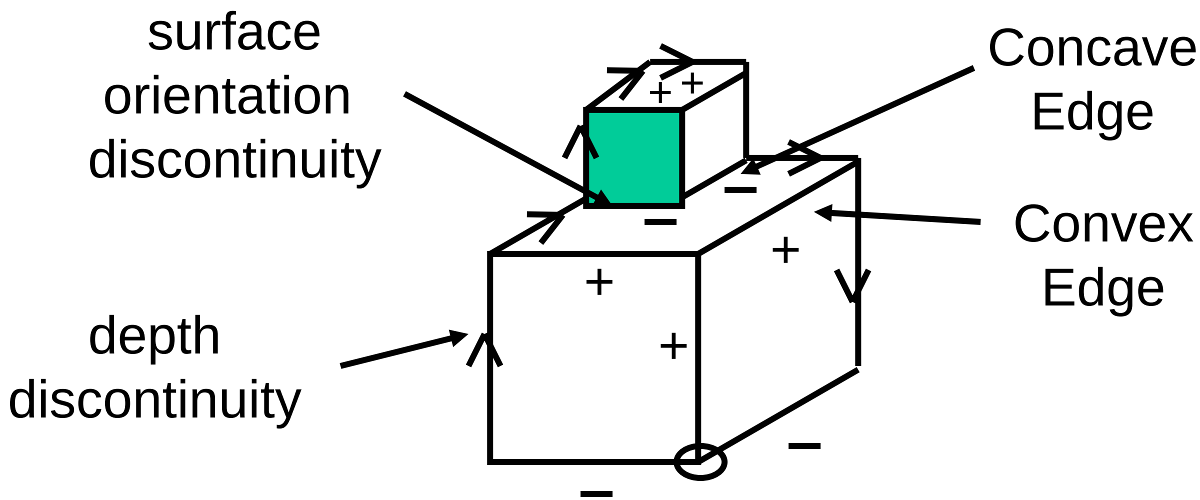
Problem: Given line drawing, assign consistent types to each edge.





Huffman Clowes (1971): Opaque, trihedral solids.
No surface marks.

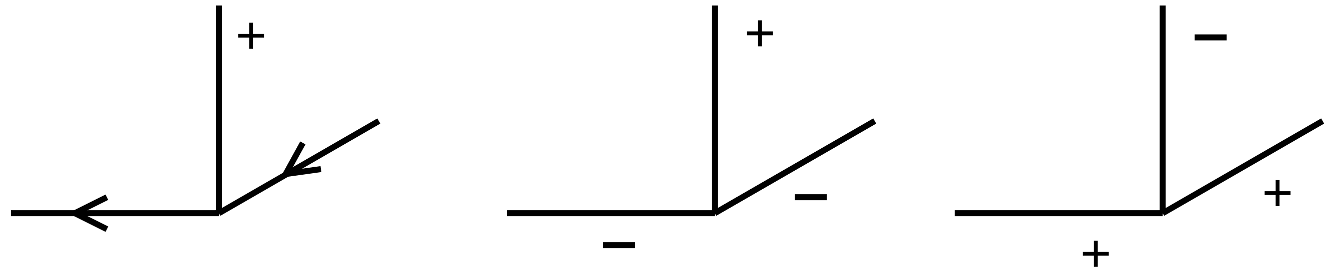
Line Labeling In Visual Interpretation



Constraint:

13 Physically realizable

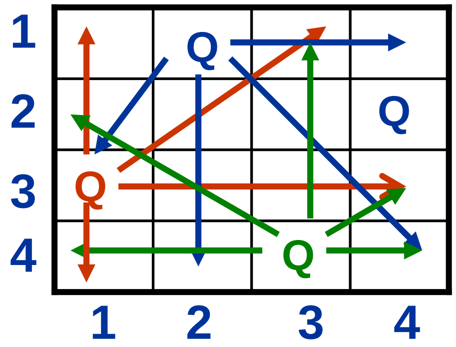
vertex labelings



Huffman Clowes (1971): Opaque, trihedral solids.
No surface marks.

Constraint Satisfaction Problems

4 Queens Problem:
Place 4 queens on a 4x4 chessboard so that no queen can attack another.



How do we formulate?

Variables	Chessboard positions
Domains	Queen 1-4 or blank
Constraints	Two positions on a line (vertical, horizontal, diagonal) cannot both be Q

Constraint Satisfaction Problem (CSP)

A Constraint Satisfaction Problem is a triple <V,D,C>, 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 a set of constraints on assignments to V
  - Each constraint specifies a set of one or more allowed variable assignments.

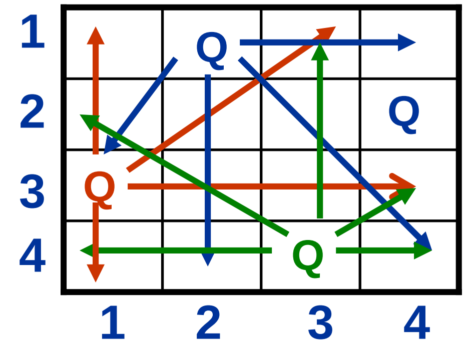
Example:

- A,B in {1,2}
- C = {{<1,2><2,1>}}

A CSP Solution: is any assignment to V, such that all constraints in C are satisfied.

Good Encodings Are Essential: 4 Queens

4 Queens Problem:
Place 4 queens on a 4x4 chessboard so that no queen can attack another.



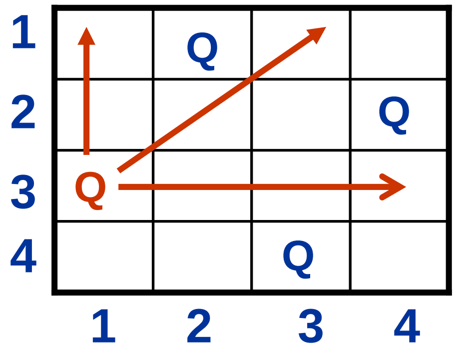
How big is the encoding?

Variables	Chessboard positions
Domains	Queen 1-4 or blank
Constraints	Two positions on a line (vertical, horizontal, diagonal) cannot both be Q

What is a better encoding?

Good Encodings Are Essential: 4 Queens

Place queens so that no queen can attack another.



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

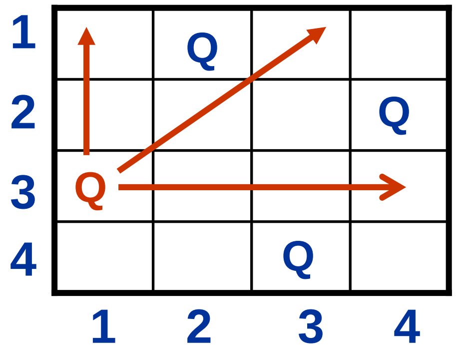
Constraints $Q_i <> Q_j$ On different rows

$|Q_i - Q_j| <> |i-j|$ Stay off the diagonals

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

Good Encodings Are Essential: 4 Queens

Place queens so that no queen can attack another.



Variables $Q_1, Q_2, Q_3, Q_4,$

Domains {1, 2, 3, 4}

Constraints $Q_i \text{<>} Q_j$ On different rows

$|Q_i - Q_j| \text{<>} |i-j|$ Stay off the diagonals

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

What is C_{13}?

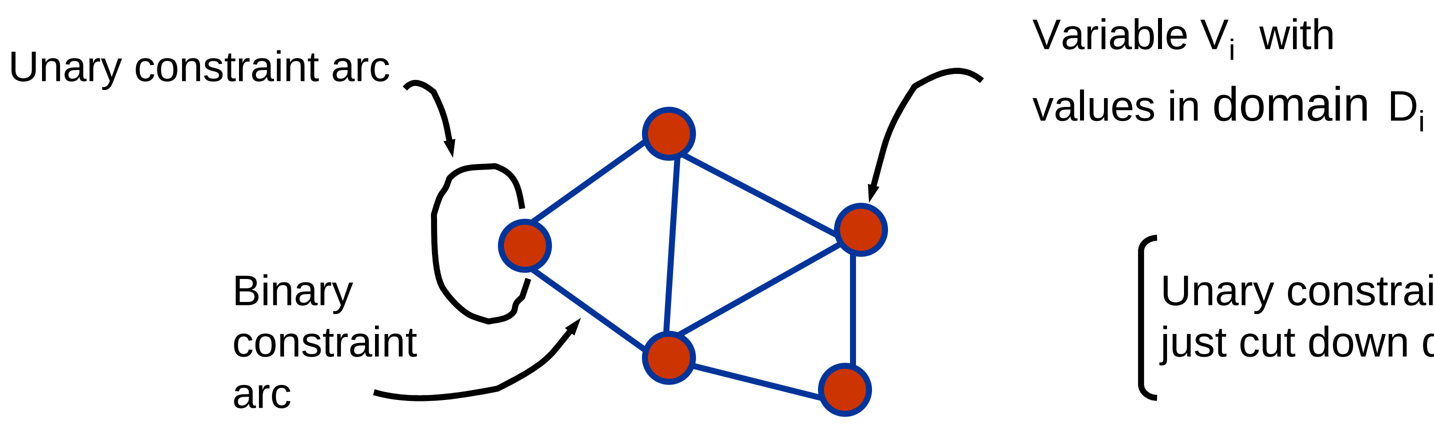
A general class of CSPs

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

- each constraint relates at most two variables.
- each variable domain is finite.
- all n-ary CSPs reducible to binary CSPs.

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

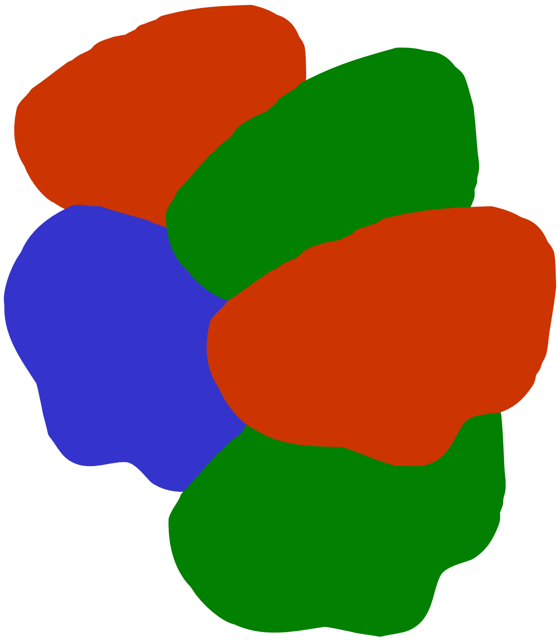
- Nodes are variables.
- Arcs are binary constraints.



[Unary constraints just cut down domains]

Example: CSP Classic - Graph Coloring

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



Variables

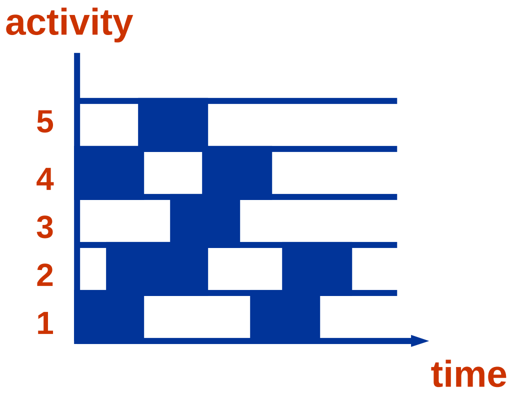
Domains

Constraints

Real World Example: Scheduling as a CSP

Choose time for activities:

- Observations on Hubble telescope.
- Jobs performed on machine tools.
- Terms to take required classes.



Variables are activities

Domains sets of possible start times (or “chunks” of time)

Constraints
1. Activities that use the same resource cannot overlap in time, and
2. Preconditions are satisfied.

Case Study: Course Scheduling

Given:

- 40 required courses (8.01, 8.02, ... 6.840), and
- 10 terms (Fall 1, Spring 1, ... , Spring 5).

Find: a legal schedule.

Constraints

Note, traditional CSPs are not for expressing (soft) preferences

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

But see recent work on semi-ring CSPs!

Alternative formulations for variables & values

VARIABLES **DOMAINS**

A. **1 var per Term**

(Fall 1) (Spring 1)
(Fall 2) (Spring 2) . . .

B. **1 var per Term-Slot**

subdivide each term
into 4 course slots:

(Fall 1,1) (Fall 1, 2)
(Fall1, 3) (Fall 1, 4)

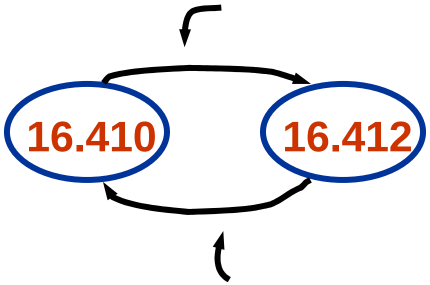
C. **1 var per Course**

Encoding Constraints

Assume: Variables = Courses, Domains = term-slots

Constraints:

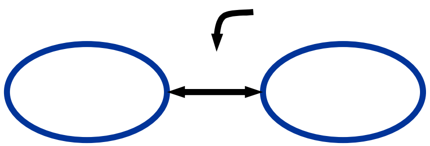
Prerequisite ➡



For each course and one of its prerequisites.

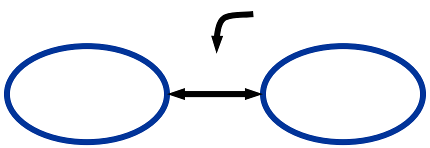
Courses offered only during certain terms ➡

Limit # courses ➡



for all pairs of vars.

Avoid time conflicts ➡



For pairs of courses offered at same time

Good News / Bad News

Good News

- very general & interesting family of problems.
- Problem formulation extensively used in autonomy and aerospace applications.

Bad News

includes NP-Hard (intractable) problems

Outline

- Constraint satisfaction problems (CSP)
- Solving CSPs
  - Arc-consistency and propagation
  - Analysis of constraint propagation
  - Search (next lecture)

Solving CSPs

Solving CSPs involves some combination of:

1. Constraint propagation (inference)
   - Eliminates values that can't be part of any solution.
2. Search
   - Explores alternate valid assignments.

Arc Consistency

Arc consistency eliminates values of each variable domain that can never satisfy a particular constraint (an arc).

$$V_i \xrightarrow{=} V_j$$

$$\{1,2,\cancel{3}\} \qquad \{1,2\}$$

- Directed arc (V_i, V_j) is arc consistent if
  - For every x in D_i, there exists some y in D_j such that assignment (x,y) is allowed by constraint C_{ij}
  - Or $\forall x \in D_i\ \exists y \in D_j$ such that (x,y) is allowed by constraint C_{ij}

    where
    - $\forall$ denotes "for all"
    - $\exists$ denotes "there exists"
    - $\in$ denotes "in"

Arc Consistency

Arc consistency eliminates values of each variable domain that can never satisfy a particular constraint (an arc).

$$V_i \xrightarrow{=} V_j$$

$$\{1,2,\cancel{3}\} \qquad \{1,2\}$$

- Directed arc (V_i, V_j) is arc consistent if
  - $\forall x \in D_i\ \exists y \in D_j$ such that (x,y) is allowed by constraint C_{ij}

Example: Given: Variables V_1 and V_2 with Domain {1,2,3,4}

Constraint: {(1, 3) (1, 4) (2, 1)}

What is the result of arc consistency?

Achieving Arc Consistency via Constraint Propagation

Arc consistency eliminates values of each variable domain that can never satisfy a particular constraint (an arc).

- Directed arc (V_i, V_j) is arc consistent if $\forall x \in D_i \; \exists y \in D_j$ such that (x,y) is allowed by constraint C_{ij}

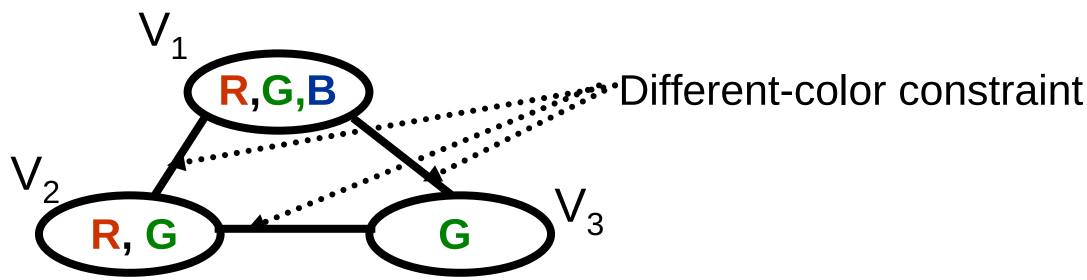
Constraint propagation: To achieve arc consistency:

- Delete every value from each tail domain D_i of each arc that fails this condition,
- Repeat until quiescence:
  - If element deleted from D_i then
    - check directed arc consistency for each arc with head D_i
  - Maintain arcs to be checked on FIFO queue (no duplicates).

Constraint Propagation Example

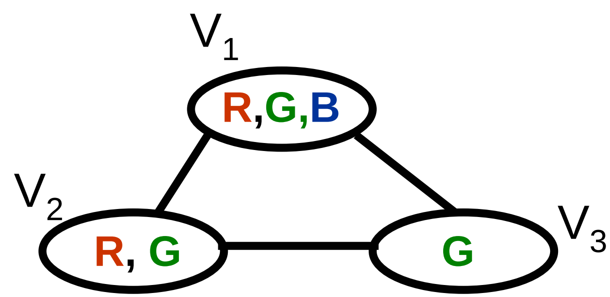
Graph Coloring

Initial Domains



Each undirected constraint arc denotes two directed constraint arcs.

Arc examined	Value deleted



Arcs to examine

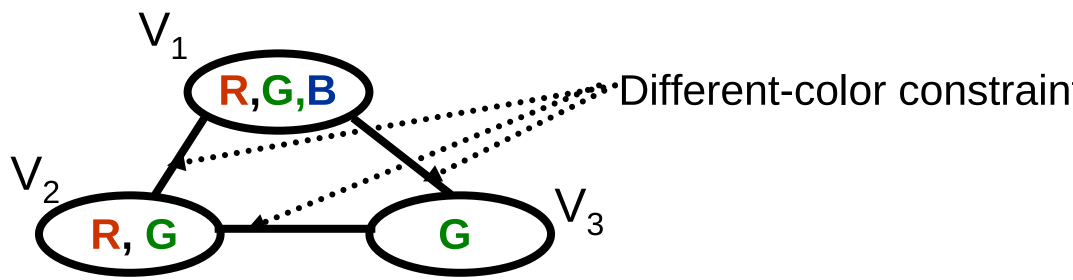
V_1-V_2, V_1-V_3, V_2-V_3

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

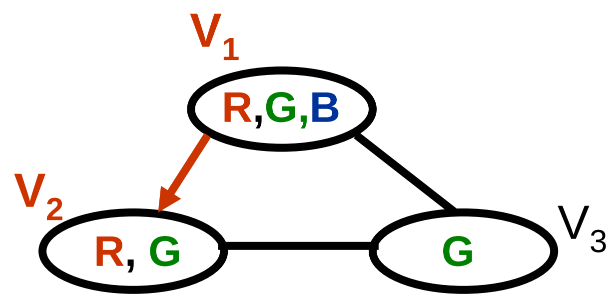
Constraint Propagation Example

Graph Coloring

Initial Domains



Arc examined	Value deleted
$V_1 > V_2$	none



Arcs to examine

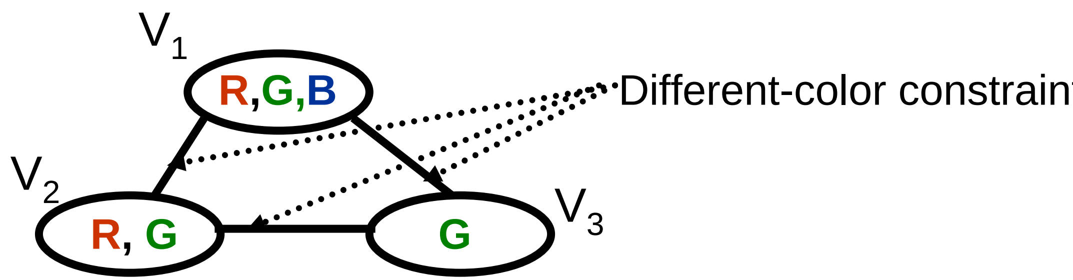
$V_1 < V_2$, $V_1 - V_3$, $V_2 - V_3$

- Delete unmentioned tail values
- $V_i - V_j$ denotes two arcs between V_i and V_j.
- Vi < Vj denotes an arc from V_j and V_i.

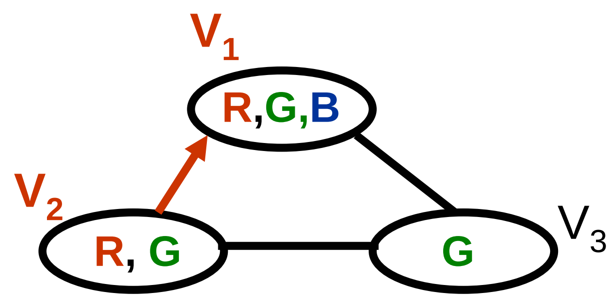
Constraint Propagation Example

Graph Coloring

Initial Domains



Arc examined	Value deleted
$V_1 > V_2$	none
$V_2 > V_1$	**none**



Arcs to examine

V_1-V_3, V_2-V_3

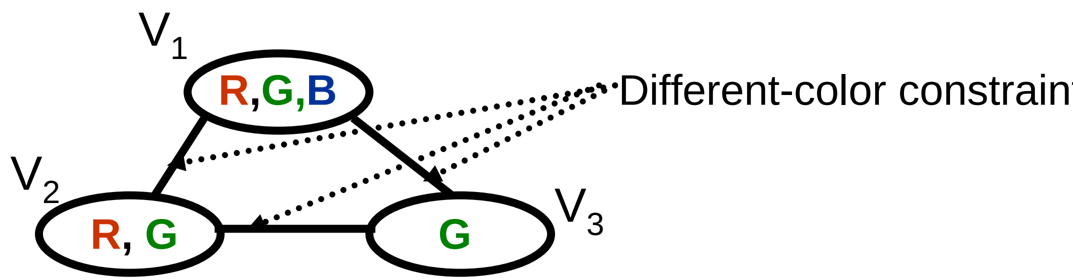
- Delete unmentioned tail values

- V_i – V_j denotes two arcs between V_i and V_j.
- Vi < Vj denotes an arc from V_j and V_i.

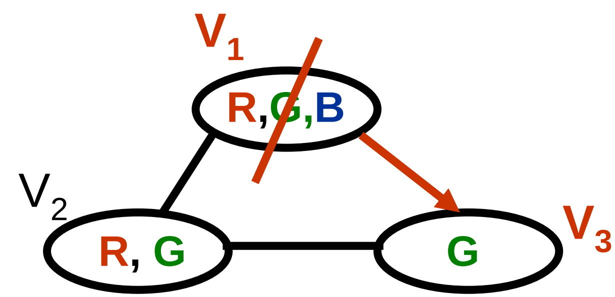
Constraint Propagation Example

Graph Coloring

Initial Domains



Arc examined	Value deleted
$V_1 – V_2$	none
$V_1 > V_3$	V_1(**G**)



Arcs to examine

$V_1<V_3$, V_2-V_3, **$V_2>V_1$**, ~~**$V_1<V_3$**,~~

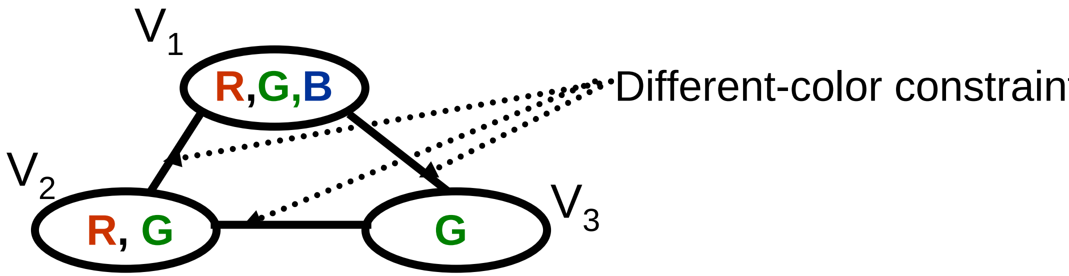
IF **An element of a variable's domain is removed,**

THEN **add all arcs to that variable to the examination queue.**

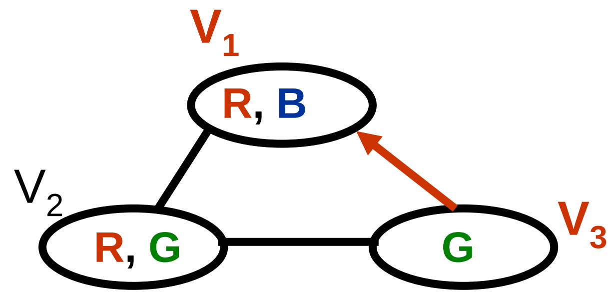
Constraint Propagation Example

Graph Coloring

Initial Domains



Arc examined	Value deleted
$V_1 - V_2$	none
$V_1 > V_3$	V_1(**G**)
$V_1 < V_3$	**none**



Arcs to examine

V_2-V_3, V_2>V_1

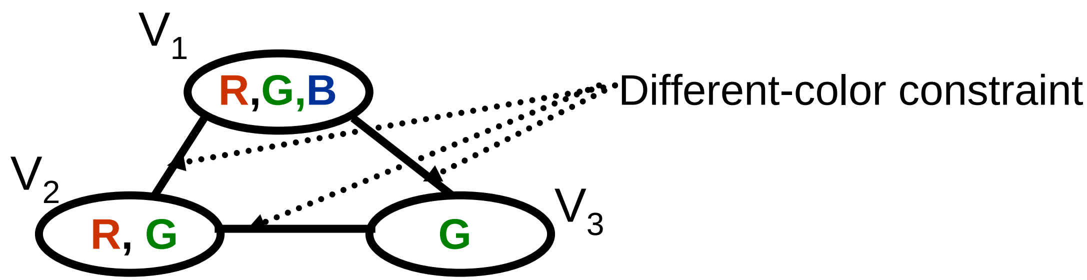
- Delete unmentioned tail values

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

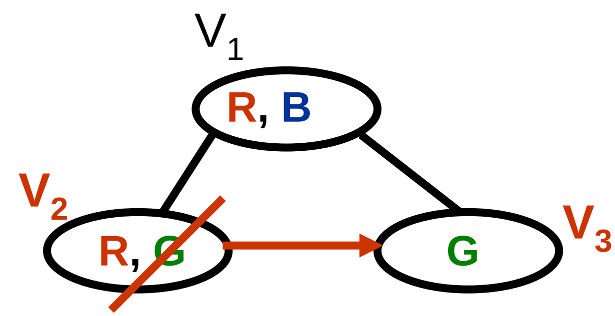
Constraint Propagation Example

Graph Coloring

Initial Domains



Arc examined	**Value deleted**
$V_1 – V_2$	none
V_1-V_3	V_1(G)
$V_2 > V_3$	V_2(G)



Arcs to examine

$V_2<V_3$, $V_2>V_1$, $V_1>V_2$

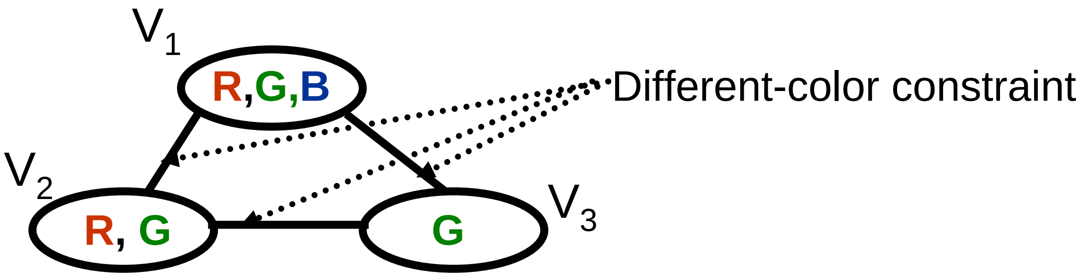
- Delete unmentioned tail values

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

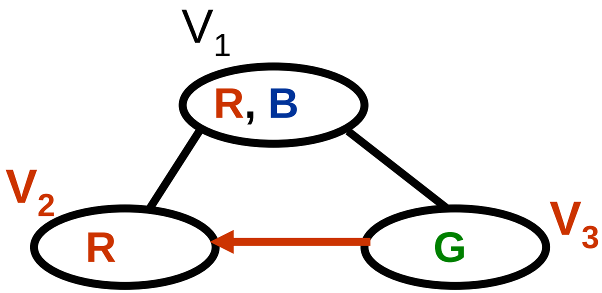
Constraint Propagation Example

Graph Coloring

Initial Domains



Arc examined	Value deleted
$V_1 - V_2$	none
V_1-V_3	V_1(G)
V_2 >V_3	V_2(G)
V_3 > V_2	**none**



Arcs to examine

V_2>V_1 , V_1>V_2

- Delete unmentioned tail values

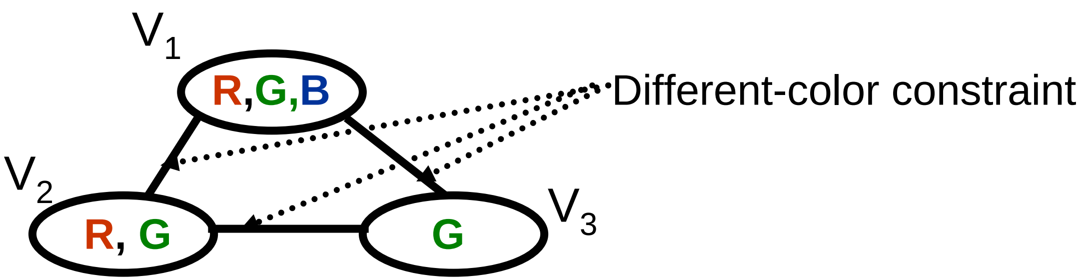
IF **An element of a variable's domain is removed,**

THEN **add all arcs to that variable to the examination queue.**

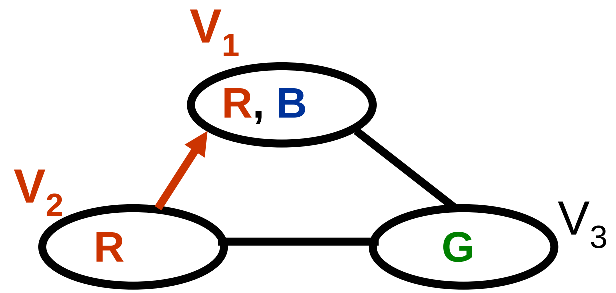
Constraint Propagation Example

Graph Coloring

Initial Domains



Arc examined	**Value deleted**
$V_1 – V_2$	none
V_1-V_3	V_1(G)
V_2 -V_3	V_2(G)
V_2>V_1	**none**



Arcs to examine

V_1>V_2

- Delete unmentioned tail values

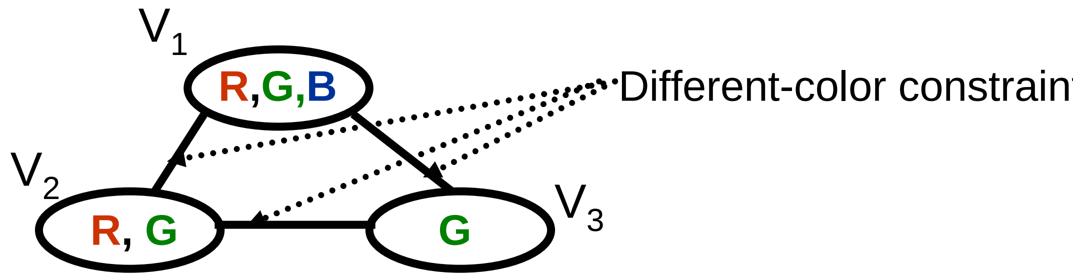
IF **An element of a variable's domain is removed,**

THEN **add all arcs to that variable to the examination queue.**

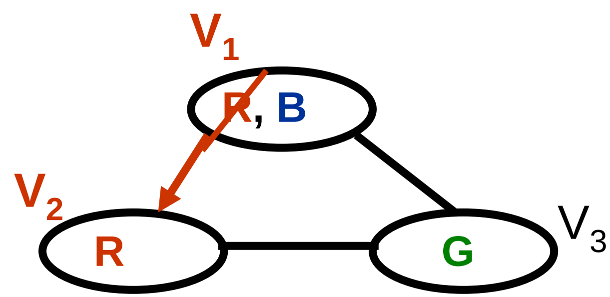
Constraint Propagation Example

Graph Coloring

Initial Domains



Arc examined	Value deleted
$V_1 - V_2$	none
V_1-V_3	$V_1(G)$
V_2 -V_3	$V_2(G)$
$V_2>V_1$	none
$V_1>V_2$	$V_1(R)$



Arcs to examine

$V_2>V_1$, $V_3>V_1$

- Delete unmentioned tail values

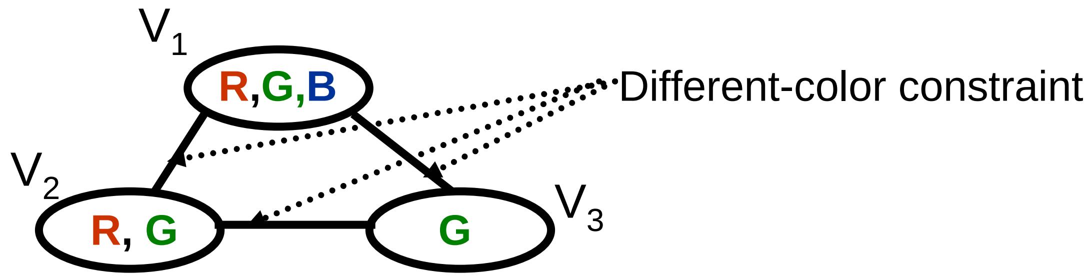
IF An element of a variable's domain is removed,

THEN add all arcs to that variable to the examination queue.

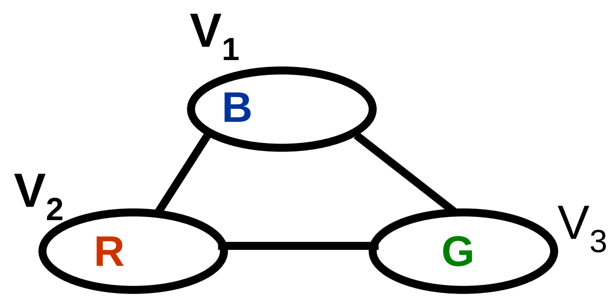
Constraint Propagation Example

Graph Coloring

Initial Domains



Arc examined	Value deleted
$V_1 – V_2$	none
V_1-V_3	V_1(G)
V_2 -V_3	V_2(G)
V_2-V_1	V_1(R)
V_2>V_1	**none**
V_3>V_1	**none**



Arcs to examine

IF **examination queue is empty**

THEN **arc (pairwise) consistent.**

Outline

- Constraint satisfaction problem (CSPS)
- Solving CSPs
  - Arc-consistency and propagation
  - Analysis of constraint propagation
  - Search (next lecture)

What is the Complexity of Constraint Propagation?

Assume:

• domains are of size at most d

•there are e binary constraints.

Which is the correct complexity?

1. $O(d^2)$
2. $O(ed^2)$
3. $O(ed^3)$
4. $O(e^d)$

Is arc consistency sound and complete?

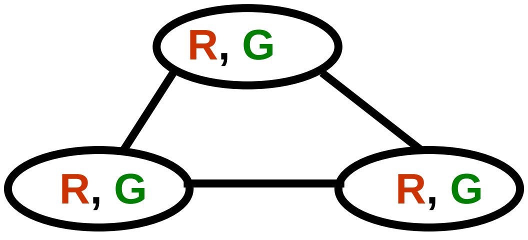
Each *arc consistent solution* selects a value for every variable from the arc consistent domains.

Completeness: Does arc consistency rule out any valid solutions?

- Yes
- No

Soundness: Is every arc-consistent solution a solution to the CSP?

- Yes
- No



Next Lecture: To Solve CSPs we combine arc consistency and search

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 (DV)
- Iterative Repair
- Backjumping (BJ)