Constraint Programming: Modeling, Arc Consistency and Propagation

Brian C. Williams
16.410-13
September 22nd, 2010

Slides draw material from:
6.034 notes, by Tomas Lozano Perez
AIMA, by Stuart Russell & Peter Norvig
Constraint Processing, by Rina Dechter

Assignments

- Assignment:
  - Problem Set #2 due today, Wed. Sept. 22nd, 2010.
  - Problem Set #3: Analysis, Path Planning and Constraint Programming, out today, due Wed., Sept. 29th, 2010.
- Reading:
  - Today: *[AIMA] Ch. 6.1, 24.3-5; Constraint Modeling.*
  - Monday: [AIMA] Ch. 6.2-5; Constraint Satisfaction.
  - To Learn More: *Constraint Processing*, by Rina Dechter
    - Ch. 2: Constraint Networks
    - Ch. 3: Consistency Enforcing and Propagation

Outline

- Interpreting line diagrams
- Constraint satisfaction problems (CSP) [aka constraint programs (CP)].
- Solving CSPs
- Case study: Scheduling (Appendix)

Outline

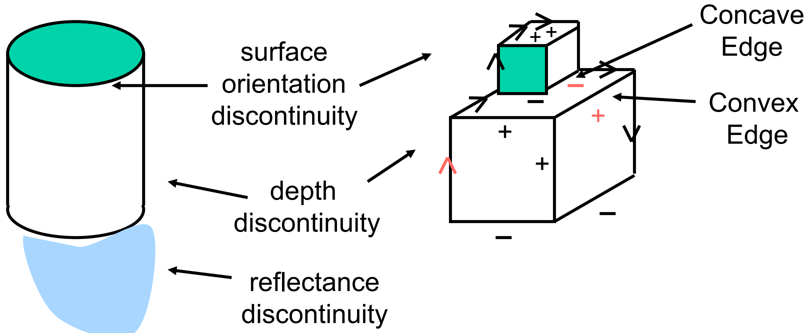
- Interpreting line diagrams
  - Constraint modeling
  - Constraint propagation
- Constraint satisfaction problems (CSP) aka constraint programs (CP)
- Solving CSPs
- Case study: Scheduling (Appendix)

Labeling Line Diagrams for Visual Interpretation

Input: Line drawing (a graph)

Physical constraints

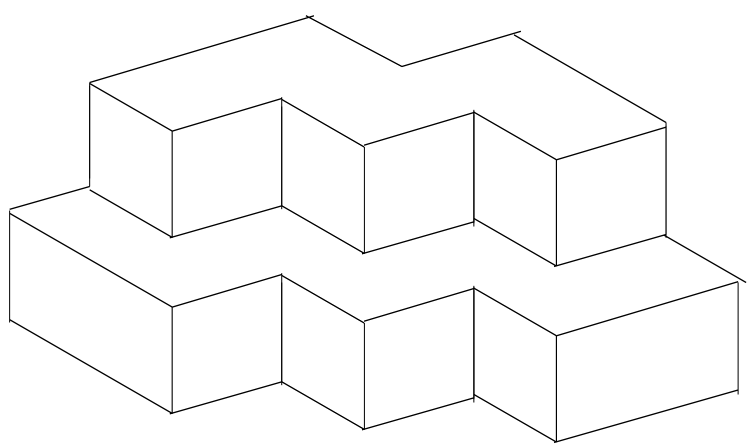
Output: Consistent assignment of line (edge) types



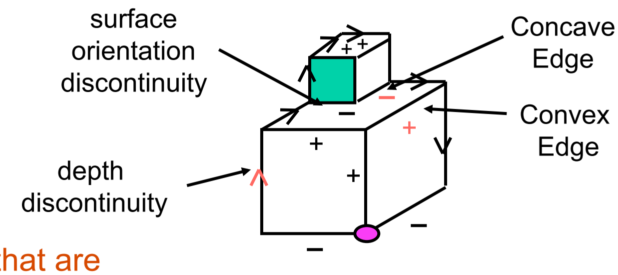
Huffman Clowes (1971): Interpret opaque, trihedral solids

Step 1: Label line types.

Requirement: Labeling must extend to complex objects



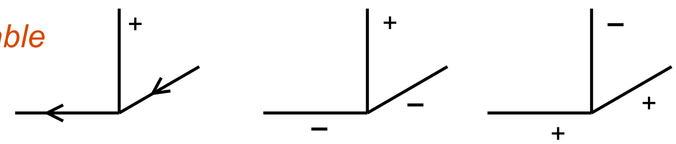
Line Labeling as Constraint Programming



18 vertex labelings that are

physically realizable

Constraints



Huffman Clowes (1971):
Interpretation of opaque, trihedral solids with no surface marks.

Waltz (1972): Compute labeling through local propagation.

Outline

- Interpreting line diagrams
  - Constraint modeling
  - Constraint propagation
- Constraint satisfaction problems (CSP)
  aka constraint programs (CP).
- Solving CSPs
- Case study: Scheduling (Appendix)

Modeling: Make Simplifying Assumptions

1. Limited line interpretations:
   No shadows or cracks.

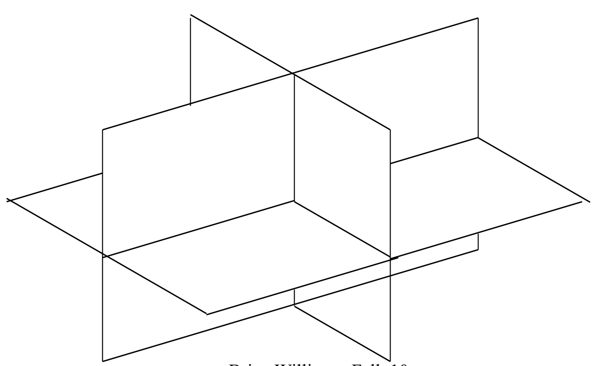
2. Three-faced vertices:
   Intersection of exactly three object faces
   (e.g., no pyramid tops).

3. General position:
   Small perturbations of selected viewing points can not lead to a change in junction type.

Modeling: Systematically derive all realizable junction types

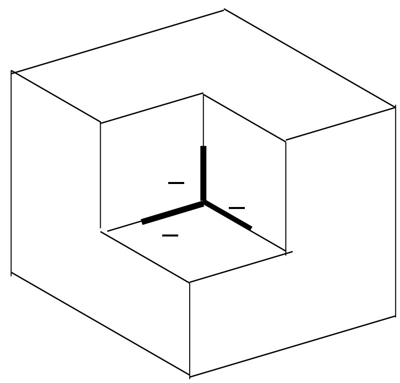
Consider:

- a three face vertex, which divides space into octants,
  - (not guaranteed to be at right angles), and
- all possible fillings of octants,
  viewed from all empty octants.



Modeling: Systematically derive all realizable junction types

- Case 1: View seven filled octants from the only empty octant.

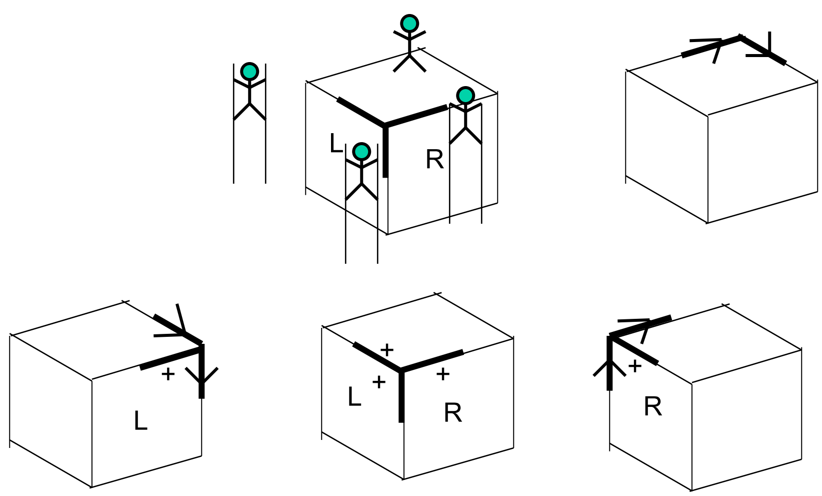


Brian Williams, Fall 10

11

Modeling: Systematically derive all realizable junction types

- Case 2a: View one filled octant from all empty upper octants….

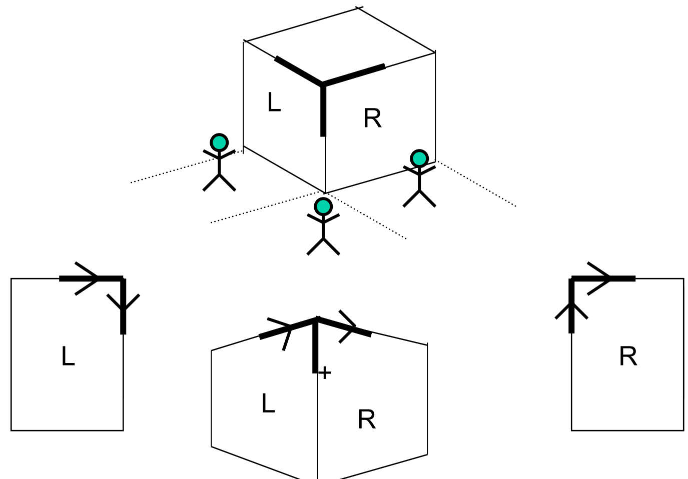


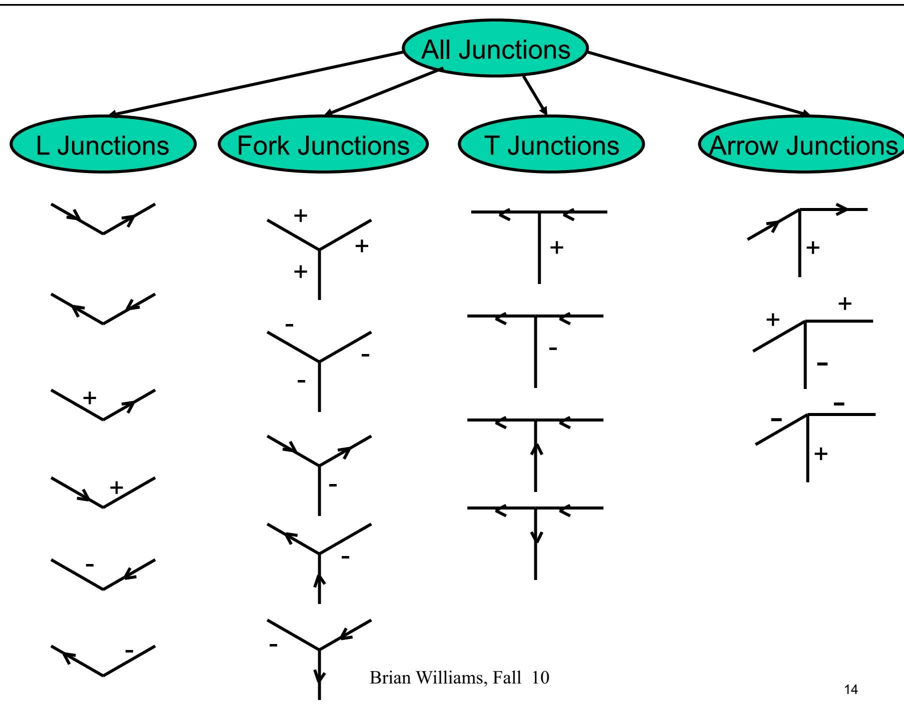
Brian Williams, Fall 10

12

Modeling: Systematically derive all realizable junction types

- Case 2b: View one filled octant from all empty lower octants.

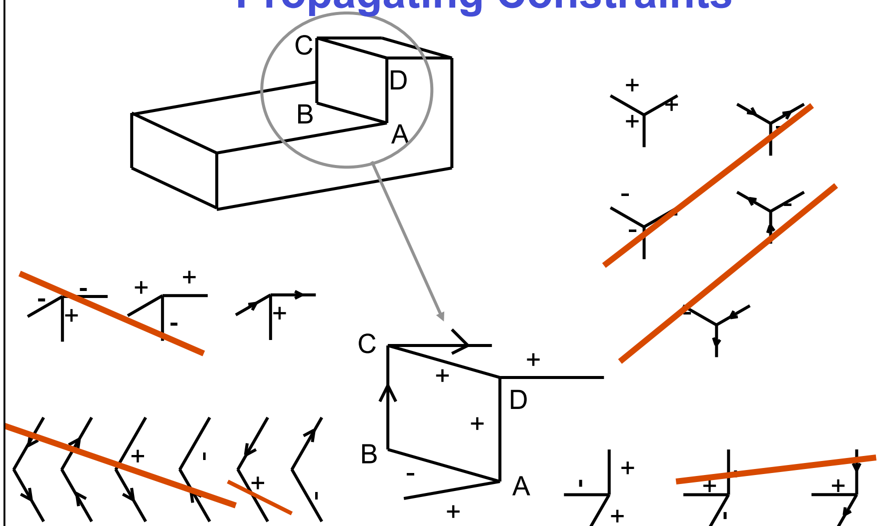




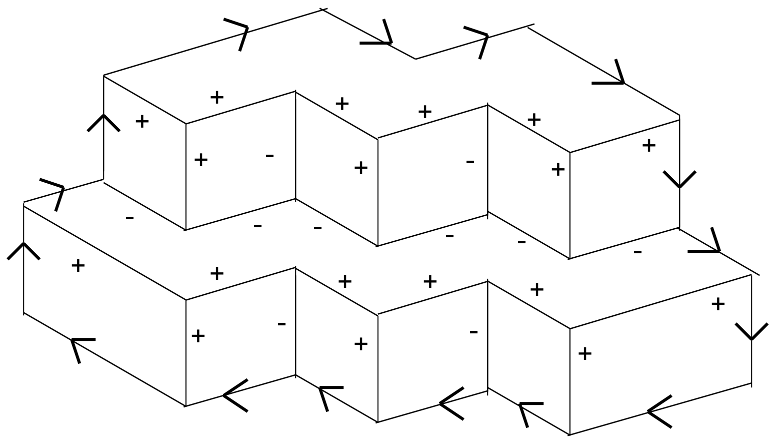
Outline

- Interpreting line diagrams
  - Constraint modeling
  - Constraint propagation
- Constraint satisfaction problems (CSP) aka constraint programs (CP).
- Solving CSPs
- Case study: Scheduling (Appendix)

Solution: Label Lines by Propagating Constraints



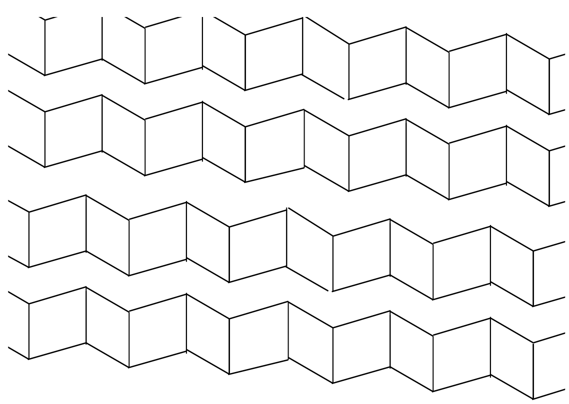
Propagate starting with the background borders



Brian Williams, Fall 10

17

Without background borders, interpretations become unstable.



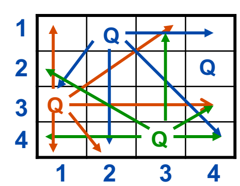
Brian Williams, Fall 10

Outline

- Interpreting line diagrams
- Constraint satisfaction problems (CSP) aka constraint programs (CP).
- Solving CSPs
- Case study: Scheduling (appendix)

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

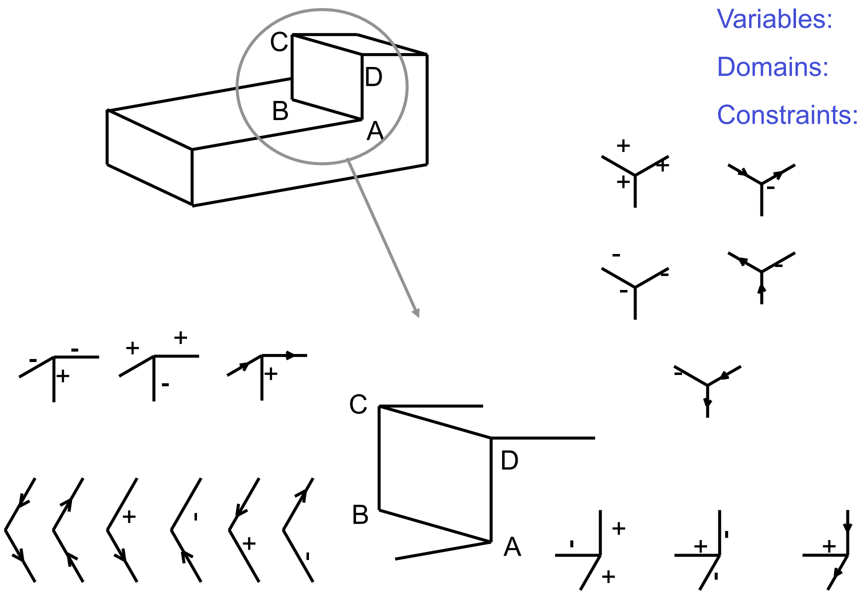
Domains Queen 1-4 or blank

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

Variables: ?

Domains: ?

Constraints: ?



Constraint Satisfaction Problems (CSP)

Input: 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 $C_i = \langle S_i,R_i \rangle$ specifies allowed variable assignments.
  - S_i the constraint's scope, is a subset of variables V.
  - R_i the constraint's relation, is a set of assignments to S_i.

Output: A full assignment to V, from elements of V's domain, such that all constraints in C are satisfied.

Example: "Provide one A and two B's."

- V = {A,B}, each with domain D_i = {1,2}
- C = {<{A,B}, {<1,2>, <1,1>}> "one A"
  <{A,B}, {<1,2>, <2,2>}>} "two Bs"
- Output: <1,2> (for example)

Conventions

- List scope in subscript.
- Specify one constraint per scope.

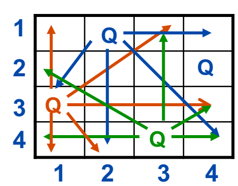
Example: "Provide one A and two B's."

- $C = \{C_{AB}\}$
  $C_{AB} = \{<1,2>\}$

- $C = \{C_A, C_B\}$
  $C_A = \{<1>\}$
  $C_B = \{<2>\}$

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?

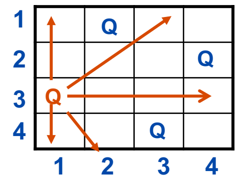
Variables Chessboard positions

Domains Queen 1-4 or blank

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

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}

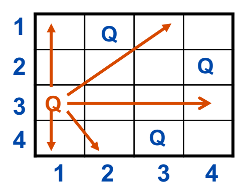
Constraints $Q_i <> Q_j$ ”On different rows”

$|Q_i - Q_j| <> |i-j|$ ”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 $Q_i <> Q_j$ ”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)}

What is C_{13}?

A general class of CSPs

Finite Domain, Binary CSPs

• each constraint relates at most two variables.

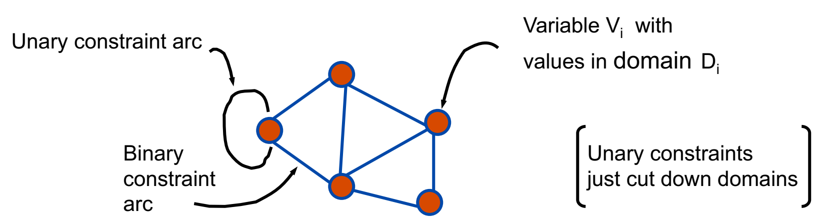
• each variable domain is finite.

Property: all n-ary CSPs are reducible to binary CSPs.

Depict as a Constraint Graph

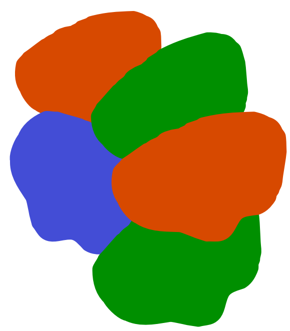
• Nodes (vertices) are variables.

• Arcs (edges) are binary constraints.



Example: Graph Coloring

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



Variables regions

Domains allowed colors

Constraints adjacent regions must have different colors

Outline

- Interpreting line problems
- Constraint satisfaction problems (CSP)
  aka constraint programs (CP).
- Solving CSPs
  - Arc-consistency and propagation
  - Analysis of constraint propagation (next lecture)
  - Search (next lecture)
- Case study: Scheduling (appendix)

Good News / Bad News

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

Bad News
includes NP-Hard (intractable ?) problems

Algorithmic Design Paradigm

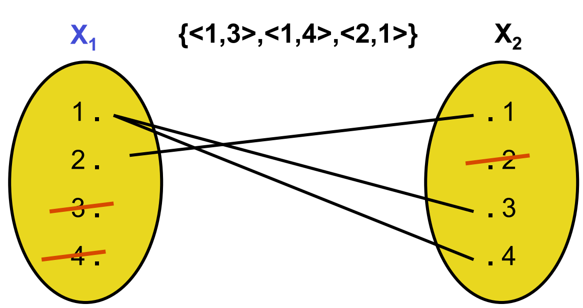
Solving CSPs involves a combination of:

1. Inference
   - Solve partially by eliminating values that can't be part of any solution (constraint propagation).
   - Make implicit constraints explicit.
2. Search
   - Try alternative assignments against constraints.

Inference: Waltz constraint propagation for visual interpretation generalizes to arc-consistency and the AC-3 algorithm.

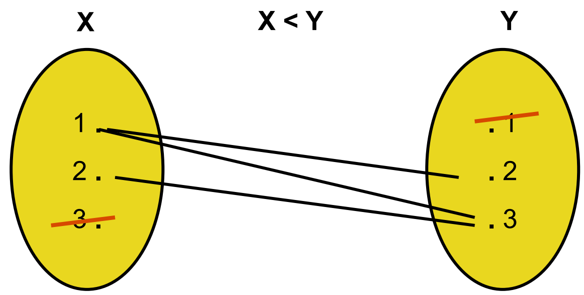
Directed Arc Consistency

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

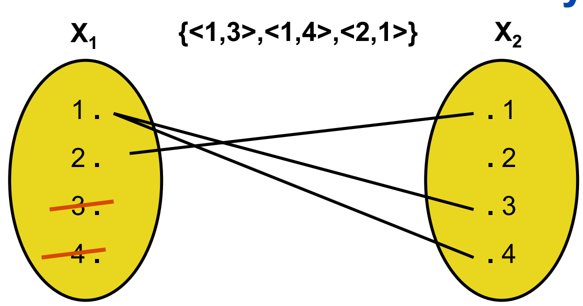


Definition: arc $<x_i, x_j>$ is arc consistent if $<x_i, x_j>$ and $<x_j, x_i>$ are directed arc consistent.

Arc Consistency



Directed Arc Consistency



Definition: arc $<x_i, x_j>$ is directed arc consistent if

- for every a_i in D_i,
  - there exists some a_j in D_j such that
    - assignment $<a_i,a_j>$ satisfies constraint C_{ij},
- $\forall a_i \in D_i$, $\exists a_j \in D_j$ such that $<a_i, a_j> \in C_{ij}$
  - $\forall$ denotes "for all," $\exists$ denotes "there exists" and $\in$ denotes "in."

Revise: A directed arc consistency procedure

Definition: arc $<x_i, x_j>$ is directed arc consistent if
$\forall a_i \in D_i, \exists a_j \in D_j$ such that $<a_i, a_j> \in C_{ij}$

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 $<a_i, a_j> \in R_{ij}$
3. then delete a_i from D_i.
4. **endif**
5. **endfor**

Constraint Processing,
by R. Dechter
pgs 54-6

Full Arc Consistency over All Constraints via Constraint Propagation

Definition: arc $<x_i, x_j>$ is directed arc consistent if
$\forall a_i \in D_i, \exists a_j \in D_j$ such that $<a_i, a_j> \in C_{ij}$

Constraint Propagation:

To achieve (directed) arc consistency over CSP:

1. For every arc C_{ij} in CSP, with tail domain D_i, call Revise.
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: A constraint satisfaction problem CSP = <X, D, C>.

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

1. **repeat**
2. **for** every $c_{ij} \in C$,
3. Revise(x_i, x_j)
4. Revise(x_j, x_i)
5. **endfor**
6. **until no domain is changed.**

For every arc, prune head and tail domains.

Constraint Processing, by R. Dechter

pgs 57

Brian Williams, Fall 10

37

Full Arc Consistency via Constraint Propagation

Definition: arc $<x_i, x_j>$ is directed arc consistent if

$\forall\, a_i \in D_i, \exists\, a_j \in D_j$ such that $<a_i, a_j> \in C_{ij}$

Constraint Propagation:

To achieve (directed) arc consistency over CSP:

1. For every arc C_{ij} in CSP, with tail domain D_i, call Revise.
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)

Brian Williams, Fall 10

38

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

AC-3(CSP)

Input: A constraint satisfaction problem CSP = <X, D, C>.

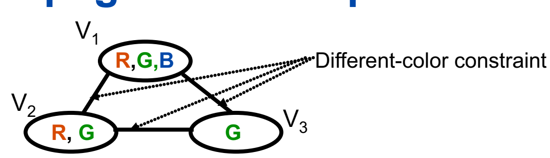
Output: CSP', the largest arc-consistent subset of CSP.

Constraint Processing, by R. Dechter
pgs 58-9

1. **for** every $c_{ij} \in C$,
2. *queue* ← *queue* $\cup$ $\{<x_i, x_j>, <x_j, x_i>\}$
3. **endfor**
4. **while** *queue* $\neq$ {}
5. select and delete arc $<x_i, x_j>$ from *queue*
6. Revise(x_i, x_j)
7. **if** Revise(x_i, x_j) caused a change in D_i
8. **then** *queue* ← *queue* $\cup$ $\{<x_k, x_i> \mid k \neq i, k \neq j\}$
9. **endif**
10. **endwhile**

Constraint Propagation Example AC-3

Graph Coloring
Initial Domains

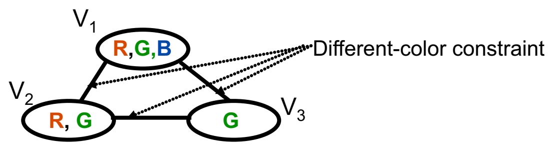


Each undirected arc denotes two directed arcs.

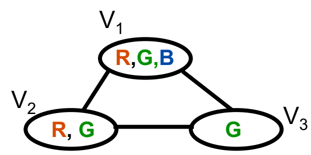
Constraint Propagation Example AC-3

Graph Coloring

Initial Domains



Arc examined	Value deleted



Arcs to examine

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

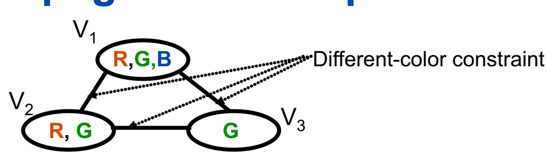
- Introduce queue of arcs to be examined.
- Start by adding all arcs to the queue.

41

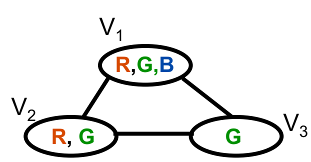
Constraint Propagation Example AC-3

Graph Coloring

Initial Domains



Arc examined	Value deleted



Arcs to examine

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

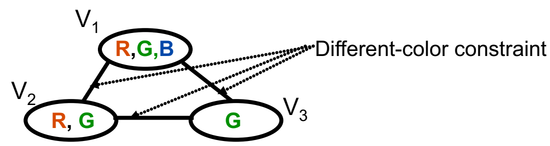
- $V_i - V_j$ denotes two arcs, between V_i and V_j.
- $V_i > V_j$ denotes an arc from V_i to V_j.

42

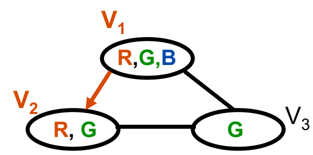
Constraint Propagation Example AC-3

Graph Coloring

Initial Domains



Arc examined	Value deleted
$V_1 > V_2$	



Arcs to examine

$V_2 > V_1$, $V_1 – V_3$, $V_2 – V_3$

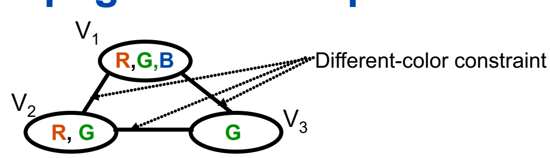
- Delete unmentioned tail values
- $V_i – V_j$ denotes two arcs, between V_i and V_j.
- $V_i > V_j$ denotes an arc from V_i to V_j.

43

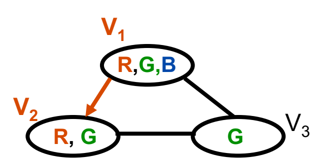
Constraint Propagation Example AC-3

Graph Coloring

Initial Domains



Arc examined	Value deleted
$V_1 > V_2$	none



Arcs to examine

$V_2 > V_1$, $V_1 – V_3$, $V_2 – V_3$

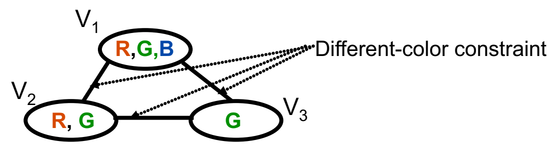
- Delete unmentioned tail values
- $V_i – V_j$ denotes two arcs, between V_i and V_j.
- $V_i > V_j$ denotes an arc from V_i to V_j.

44

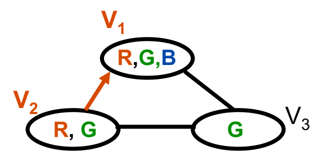
Constraint Propagation Example AC-3

Graph Coloring

Initial Domains



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



Arcs to examine

$V_1 - V_3$, $V_2 - V_3$

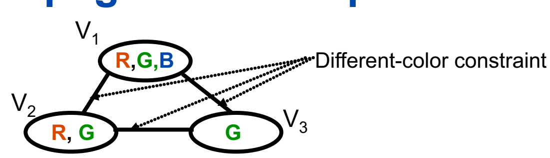
- Delete unmentioned tail values
- $V_i - V_j$ denotes two arcs, between V_i and V_j.
- $V_i > V_j$ denotes an arc from V_i to V_j.

45

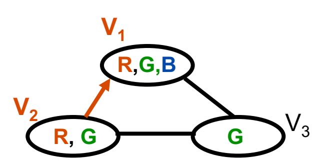
Constraint Propagation Example AC-3

Graph Coloring

Initial Domains



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



Arcs to examine

$V_1 - V_3$, $V_2 - V_3$

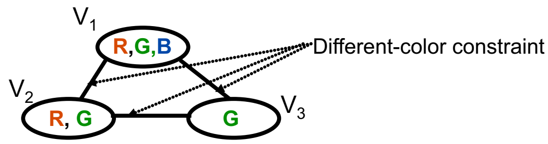
- Delete unmentioned tail values
- $V_i - V_j$ denotes two arcs, between V_i and V_j.
- $V_i > V_j$ denotes an arc from V_i to V_j.

46

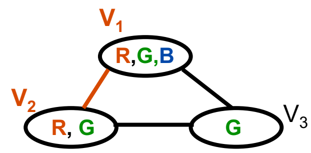
Constraint Propagation Example AC-3

Graph Coloring

Initial Domains



Arc examined	Value deleted
$V_1 – V_2$	none



Arcs to examine

$V_1 – V_3$, $V_2 – V_3$

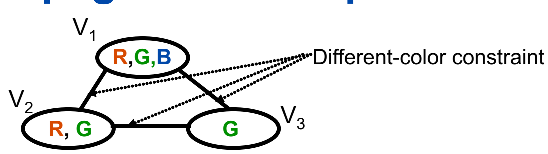
- Delete unmentioned tail values
- $V_i – V_j$ denotes two arcs, between V_i and V_j.
- $V_i > V_j$ denotes an arc from V_i to V_j.

47

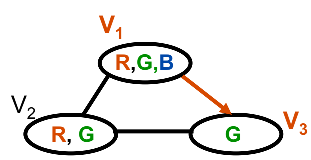
Constraint Propagation Example AC-3

Graph Coloring

Initial Domains



Arc examined	Value deleted
$V_1 – V_2$	none
$V_1 > V_3$	



Arcs to examine

$V_3 > V_1$, $V_2 – V_3$

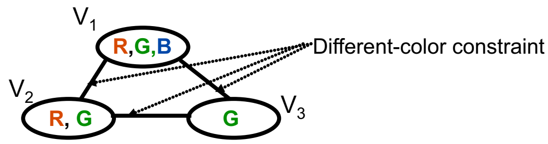
- Delete unmentioned tail values
- $V_i – V_j$ denotes two arcs, between V_i and V_j.
- $V_i > V_j$ denotes an arc from V_i to V_j.

48

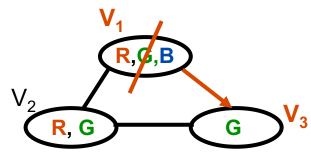
Constraint Propagation Example AC-3

Graph Coloring

Initial Domains



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



Arcs to examine

$V_3 > V_1,\ V_2 - V_3$

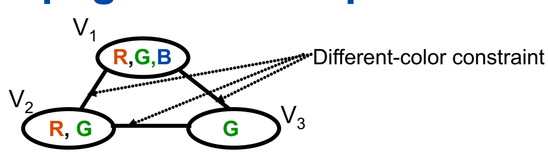
IF An element of a variable's domain is removed,
THEN add all arcs to that variable to the examination queue.

49

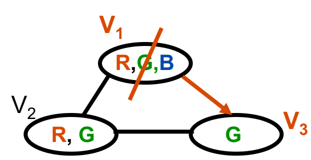
Constraint Propagation Example AC-3

Graph Coloring

Initial Domains



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



Arcs to examine

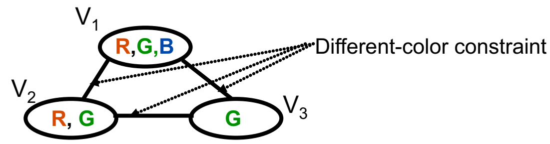
$V_3 > V_1,\ V_2 - V_3,\ V_2 > V_1,\ \cancel{V_3 > V_1}$

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

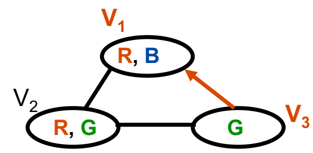
50

Constraint Propagation Example AC-3

Graph Coloring
Initial Domains



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



Arcs to examine

$V_2 – V_3$, $V_2 > V_1$

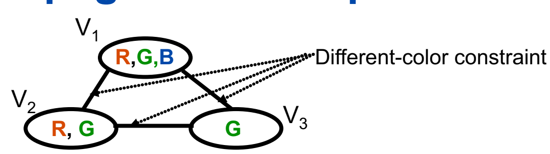
- Delete unmentioned tail values

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

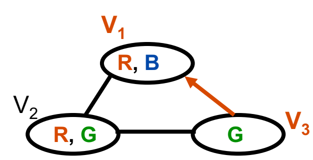
51

Constraint Propagation Example AC-3

Graph Coloring
Initial Domains



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



Arcs to examine

$V_2 – V_3$, $V_2 > V_1$

- Delete unmentioned tail values

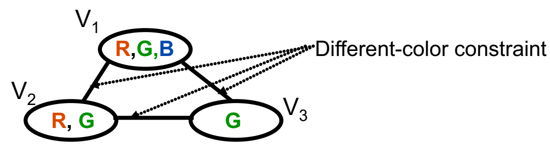
IF An element of a variable's domain is removed,
THEN add all arcs to that variable to the examination queue.

52

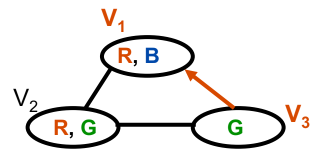
Constraint Propagation Example AC-3

Graph Coloring

Initial Domains



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



Arcs to examine

$V_2 - V_3$, $V_2 > V_1$

- Delete unmentioned tail values

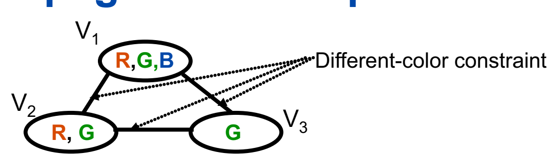
IF An element of a variable's domain is removed,
THEN add all arcs to that variable to the examination queue.

53

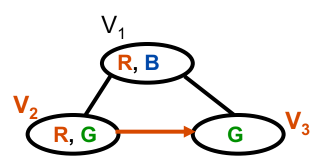
Constraint Propagation Example AC-3

Graph Coloring

Initial Domains



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



Arcs to examine

$V_3 > V_2$, $V_2 > V_1$

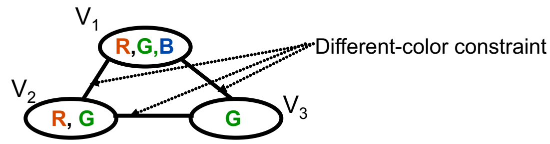
- Delete unmentioned tail values

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

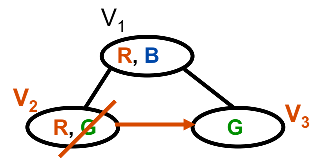
54

Constraint Propagation Example AC-3

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

$V_3 > V_3$, $V_2 > V_1$

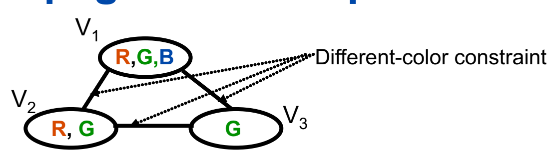
- Delete unmentioned tail values

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

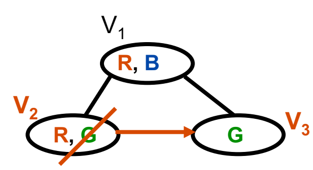
55

Constraint Propagation Example AC-3

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

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

- Delete unmentioned tail values

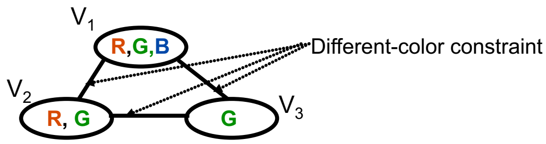
IF An element of a variable's domain is removed,
THEN add all arcs to that variable to the examination queue.

56

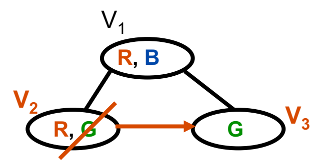
Constraint Propagation Example AC-3

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

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

- Delete unmentioned tail values

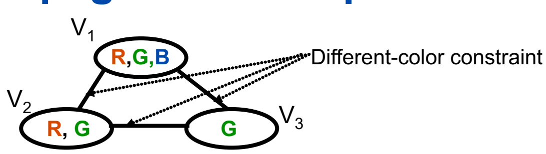
IF An element of a variable's domain is removed,
THEN add all arcs to that variable to the examination queue.

57

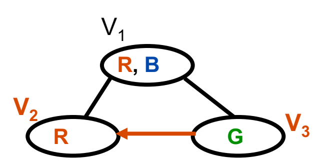
Constraint Propagation Example AC-3

Graph Coloring

Initial Domains



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



Arcs to examine

$V_2 > V_1$, $V_1 > V_2$

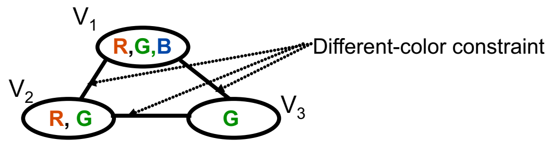
- Delete unmentioned tail values

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

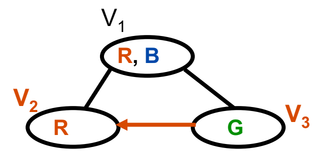
58

Constraint Propagation Example AC-3

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

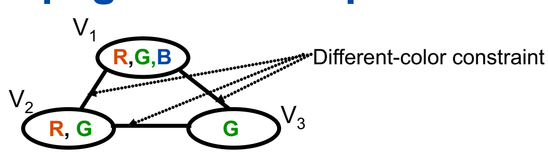
$V_2 > V_1$, $V_1 > V_2$

- Delete unmentioned tail values

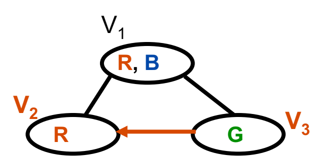
IF An element of a variable's domain is removed,
THEN add all arcs to that variable to the examination queue.

Constraint Propagation Example AC-3

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

$V_2 > V_1$, $V_1 > V_2$

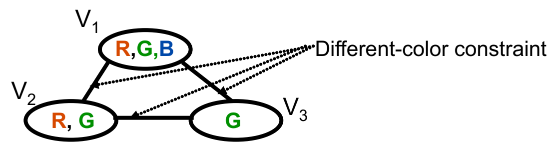
- Delete unmentioned tail values

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

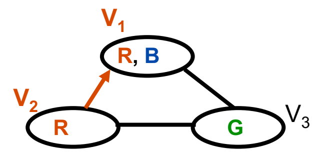
Constraint Propagation Example AC-3

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)
$V_2 > V_1$	



Arcs to examine

$V_1 > V_2$

- Delete unmentioned tail values

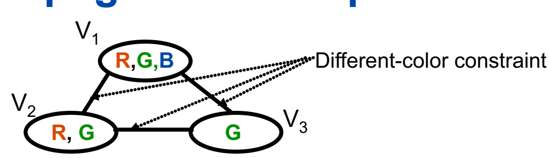
IF An element of a variable's domain is removed,
THEN add all arcs to that variable to the examination queue.

61

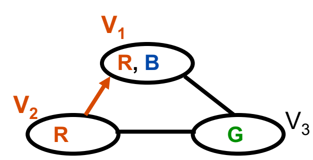
Constraint Propagation Example AC-3

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

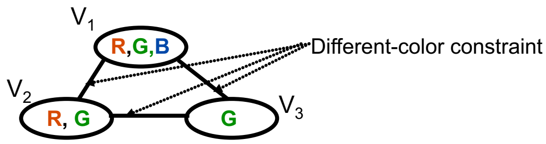
IF An element of a variable's domain is removed,
THEN add all arcs to that variable to the examination queue.

62

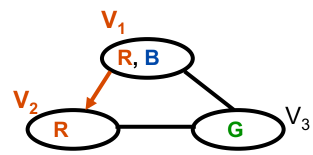
Constraint Propagation Example AC-3

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
$\mathbf{V_1 > V_2}$	



Arcs to examine

• Delete unmentioned tail values

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

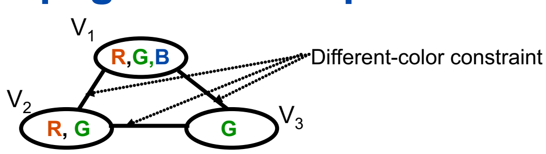
THEN **add all arcs to that variable to the examination queue.**

63

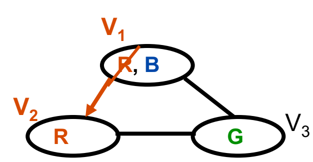
Constraint Propagation Example AC-3

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
$\mathbf{V_1 > V_2}$	$\mathbf{V_1(R)}$



Arcs to examine

• Delete unmentioned tail values

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

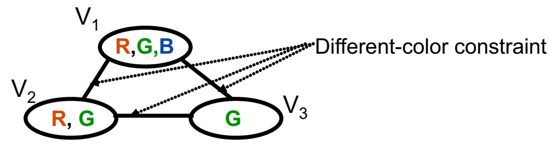
THEN **add all arcs to that variable to the examination queue.**

64

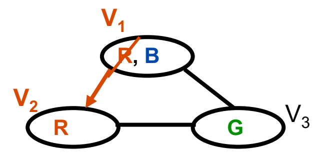
Constraint Propagation Example AC-3

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,**

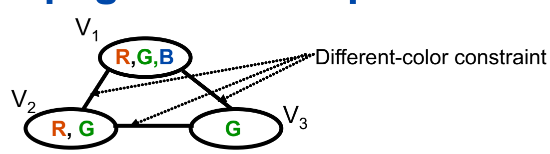
THEN **add all arcs to that variable to the examination queue.**

65

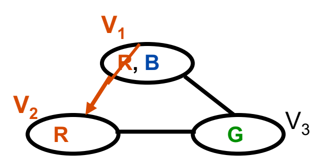
Constraint Propagation Example AC-3

Graph Coloring

Initial Domains



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



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,**

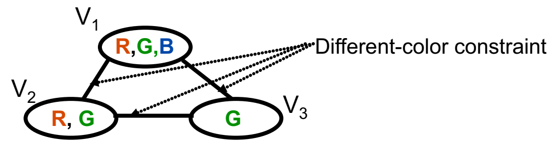
THEN **add all arcs to that variable to the examination queue.**

66

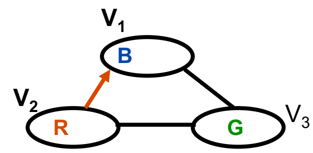
Constraint Propagation Example AC-3

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)
$V_2 > V_1$	



Arcs to examine

$V_3 > V_1$

- Delete unmentioned tail values

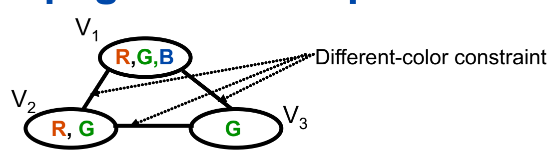
IF An element of a variable's domain is removed,
THEN add all arcs to that variable to the examination queue.

67

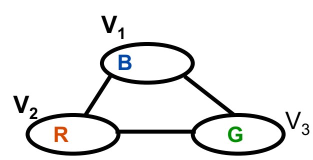
Constraint Propagation Example AC-3

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



Arcs to examine

$V_3 > V_1$

- Delete unmentioned tail values

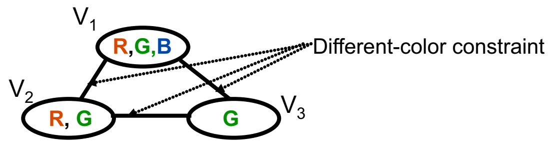
IF An element of a variable's domain is removed,
THEN add all arcs to that variable to the examination queue.

68

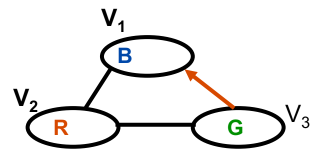
Constraint Propagation Example AC-3

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
$V_3 > V_1$	



Arcs to examine

- Delete unmentioned tail values

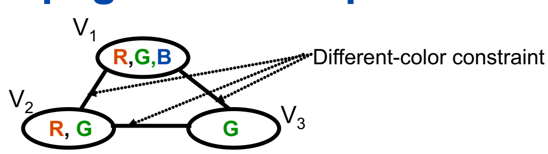
IF **An element of a variable's domain is removed,**
THEN **add all arcs to that variable to the examination queue.**

69

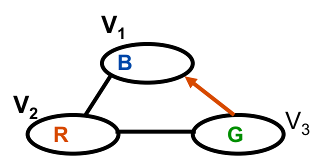
Constraint Propagation Example AC-3

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.**

70

Next: 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 (DVO)
- Iterative Repair
- Backjumping (BJ)

Outline

- Interpreting line diagrams
- Constraint satisfaction problem (CSPS) aka constraint programs (CP).
- Solving CSPs
- Case study: Scheduling (appendix)

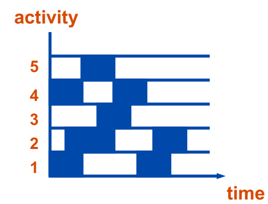
Real World Example: Scheduling as a CSP

Choose time of activities:

• Observations by the Hubble telescope.

• Jobs performed on machine tools.

• Classes taken for degree.



Variables are activities

Domains Are possible start times (or "chunks" of time)

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

Brian Williams, Fall 10

73

Case Study: Course Scheduling

Given:

• 32 required courses (8.01, 8.02, ... 16.410), and

• 8 terms (Fall 1, Spring 1, ... , Spring 4).

Find: a legal schedule.

Constraints
• Pre-requisites satisfied,
• Courses offered only during certain terms,
• A limited number of courses can be taken per term (say 4), and
• Avoid time conflicts between courses.

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

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

But see recent research on valued CSPs!

74

Alternative formulations for variables and values

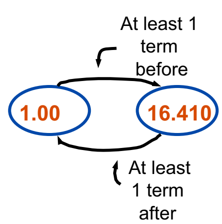
VARIABLES	DOMAINS
A. **1 var per Term** (Fall 1) (Spring 1) (Fall 2) (Spring 2) . . .	All legal combinations of 4 courses, all offered during that term.
B. **1 var per Term-Slot** subdivide each term into 4 course slots: (Fall 1,1) (Fall 1, 2) (Fall1, 3) (Fall 1, 4)	All courses offered during that term.
C. **1 var 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:

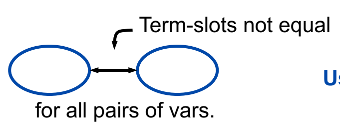
Prerequisite ➡



For pairs of courses that must be ordered.

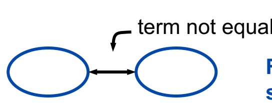
Courses offered only during certain terms ➡ **Filter domain**

Limit # courses ➡



Use term-slots only once

Avoid time conflicts ➡



For course pairs offered at same or overlapping times