Constraint Programming II: Solving CPs using Propagation and Basic Search

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

Brian C. Williams
Enrique Fernandez
16.410/413
October 28th, 2015

Assignments

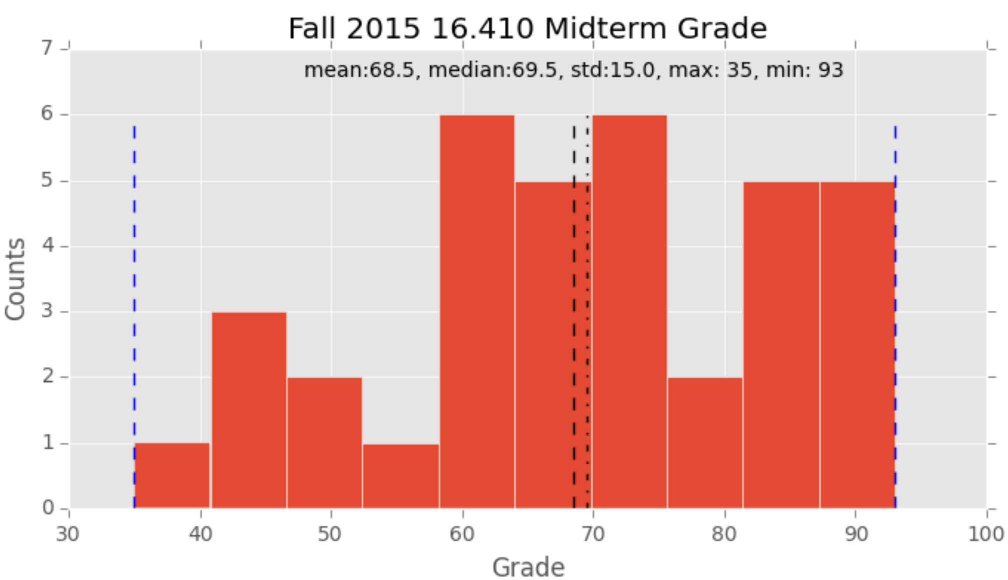
- Remember:
  - Problem Set #6: Out today. Due next Wednesday
  - Project Part 1 (16.413): Due on Nov 6th

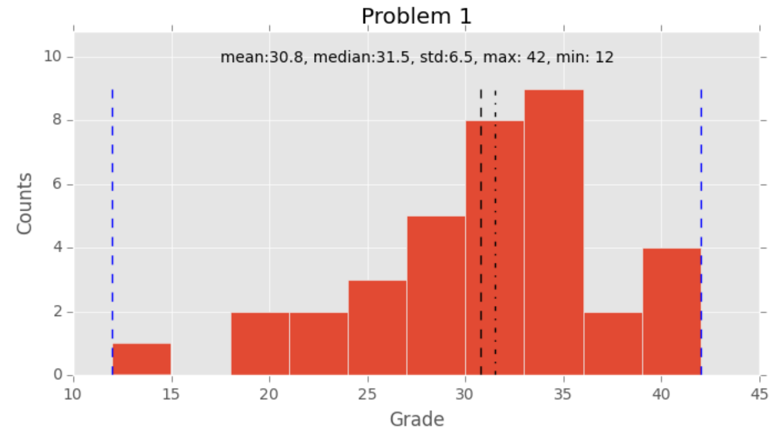
- Reading:
  - Today and Monday:
    [AIMA] Ch. 6.2-5; Constraint Satisfaction.

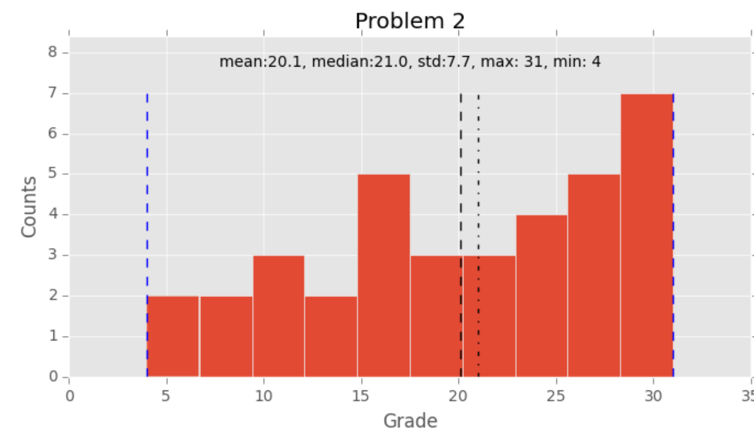
- To Learn More: *Constraint Processing*, by Rina Dechter.
  - Ch. 5: General Search Strategies: Look-Ahead.
  - Ch. 6: General Search Strategies: Look-Back.
  - Ch. 7: Stochastic Greedy Local Search.

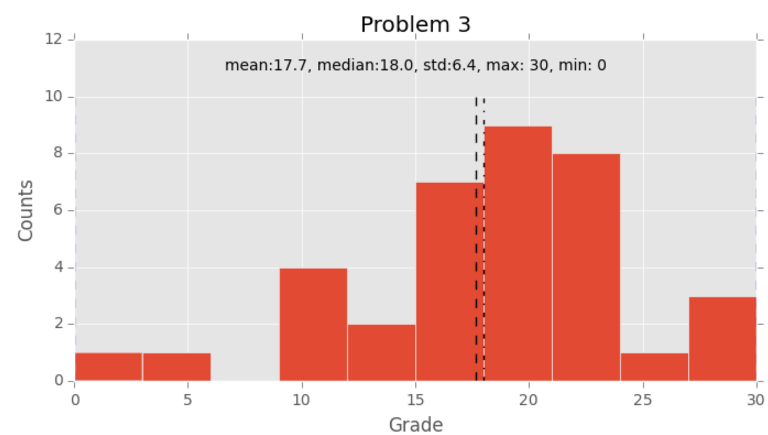
Midterm results

	P1	P2	P3	Total
Max	42	31	30	**93**
Min	12	4	0	**35**
Avg	31	20	18	**69**
Median	32	21	18	**70**
Std	6.45	7.73	6.36	**14.98**
# 0	0	0	1	**0**

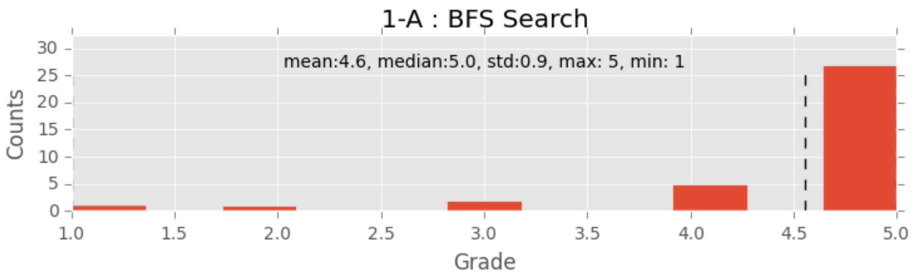


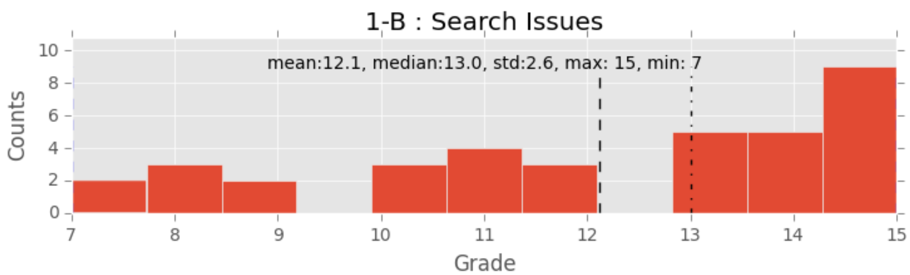


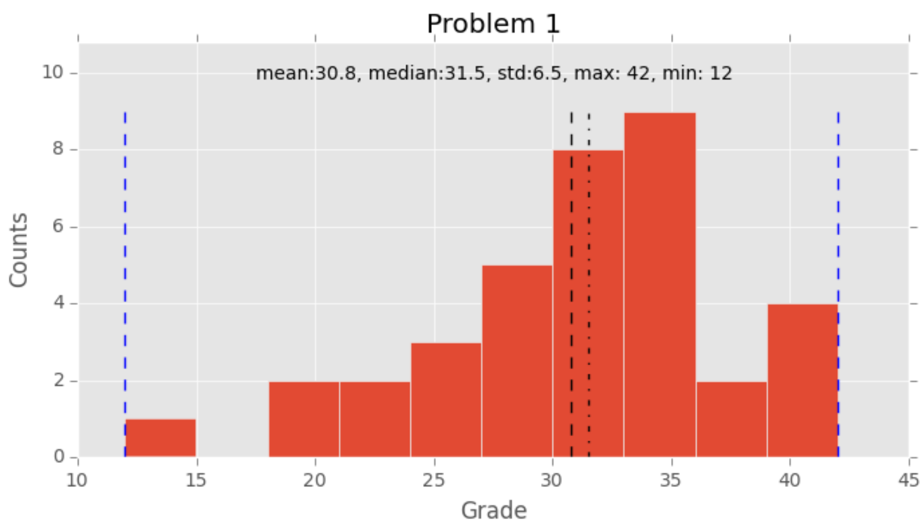


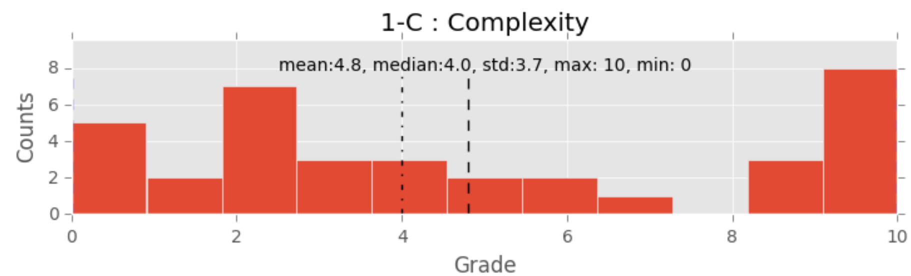


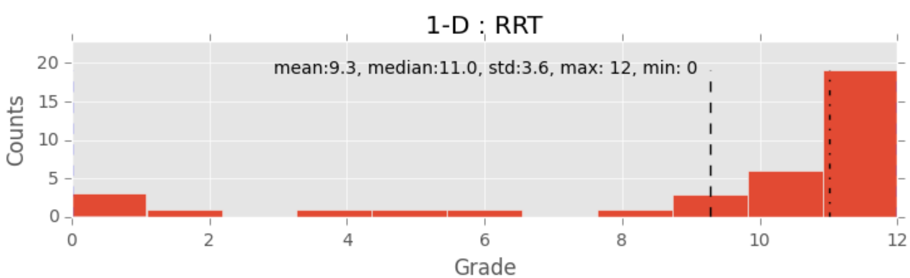
	1-A	1-B	1-C	1-D	P1 (Total)
Max	5	15	10	12	42
Min	1	7	0	0	12
Avg	5	12	5	9	31
Median	5	13	4	11	32
Std	0.94	2.61	3.73	3.56	6.45
# 0s	0	0	5	1	0



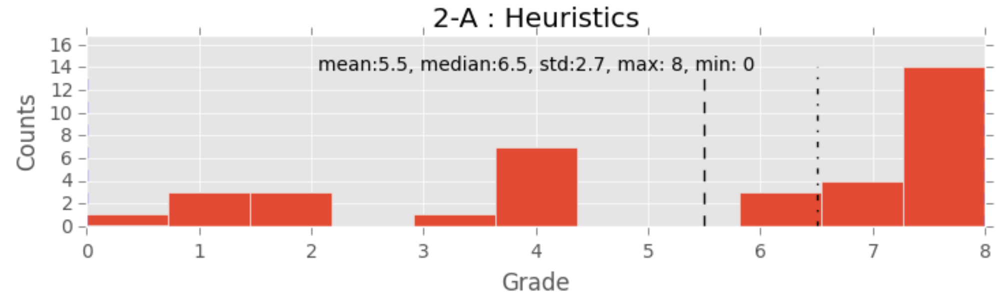


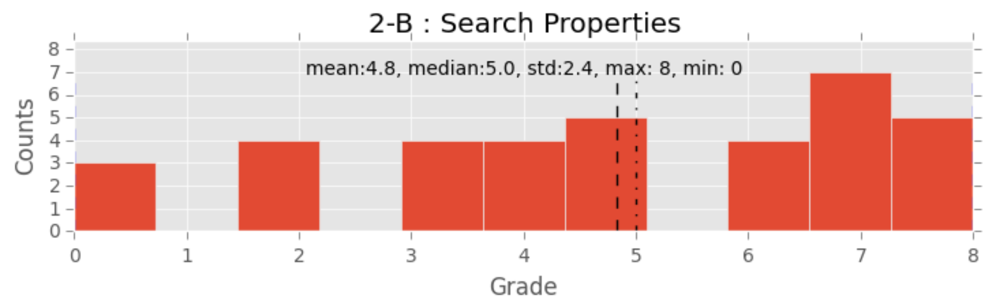


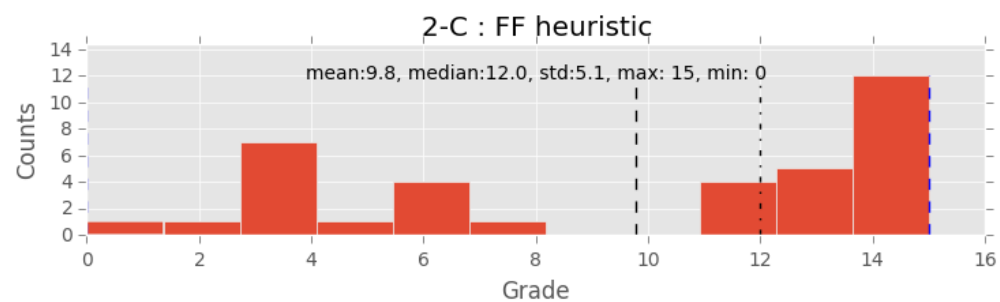


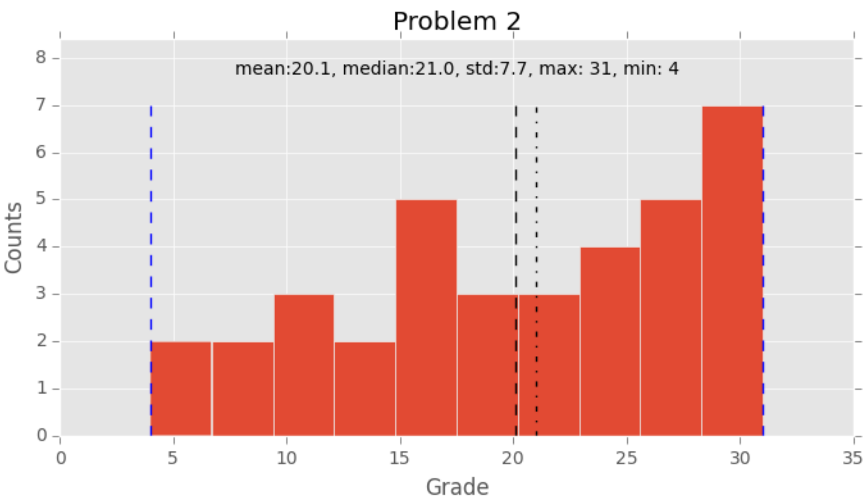


	2-A	2-B	2-C	P2 (Total)
Max	8	8	15	31
Min	0	0	0	4
Avg	6	5	10	20
Median	7	5	12	21
Std	2.66	2.43	5.14	7.73
# 0s	1	3	1	0

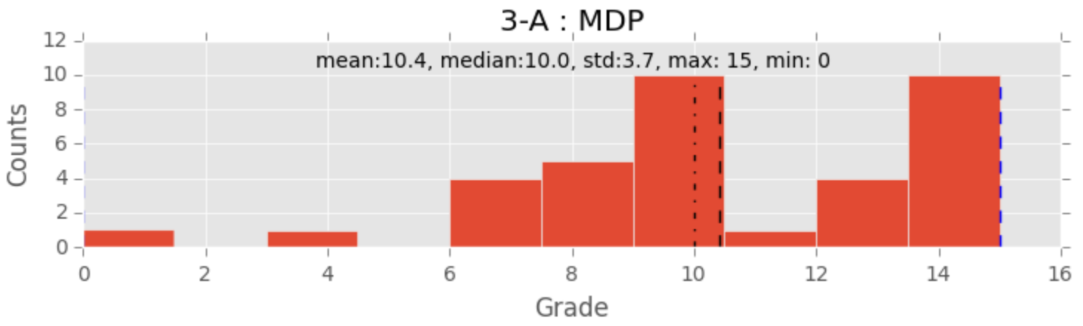


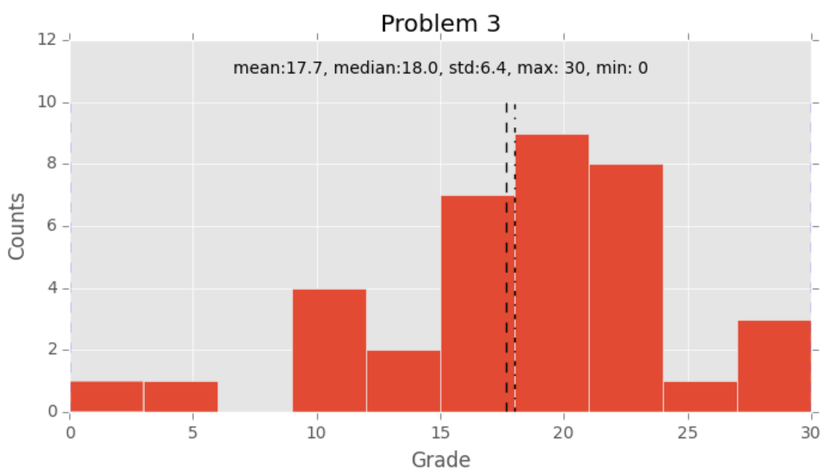


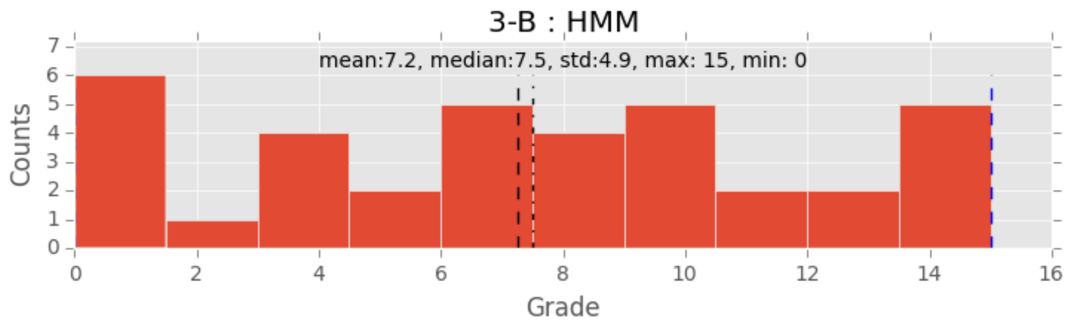




	3-A	3-B	P3 (Total)
Max	15	15	30
Min	0	0	0
Avg	10	7	18
Median	10	8	18
Std	3.68	4.92	6.36
# 0s	1	6	1







Constraint Problems are Everywhere

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

(a) Sudoku Puzzle

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

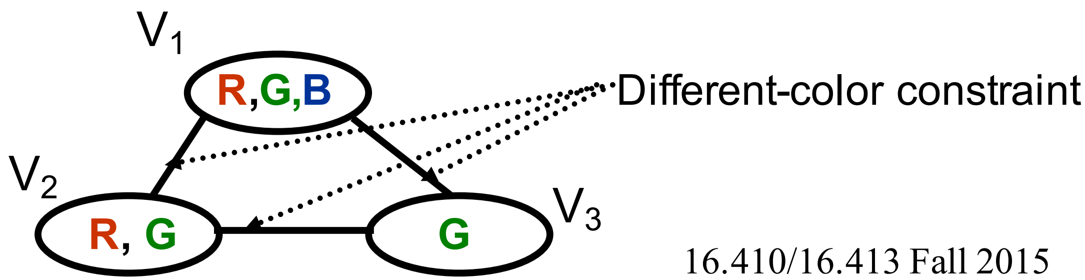
(b) The Solution

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.



V? V = {V1, V2, V3}

D_1? D_1 = {R,G,B}

C_{12}? C_{12} = {<R,G>, <G, R>, <B, R>, <B, G>}

Constraint Modeling (Programming) Languages

Features Declarative specification of the problem that separates the formulation and the search strategy.

Example: Constraint Model of the Sudoku Puzzle in

Number Jack (http://4c110.ucc.ie/numberjack/home).

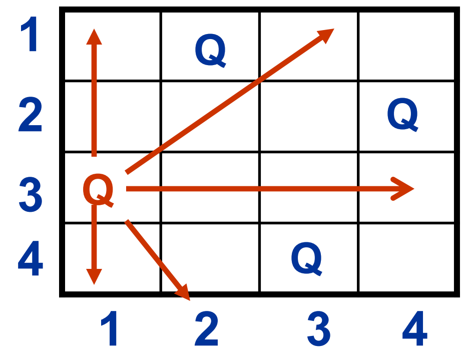
```
matrix = Matrix(N*N,N*N,1,N*N)
sudoku = Model( [AllDiff(row) for row in matrix.row],
                [AllDiff(col) for col in matrix.col],
                [AllDiff(matrix[x:x+N, y:y+N].flat)
                    for x in range(0,N*N,N)
                    for y in range(0,N*N,N)] )
```

Constraint Problems are Everywhere

(a) Sudoku Puzzle

N-Queens

Place queens so that no queen can attack another.



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

Outline

- Arc-consistency and constraint propagation.
- Analysis of constraint propagation.
- Solving CSPs using search.

Good News / Bad News

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

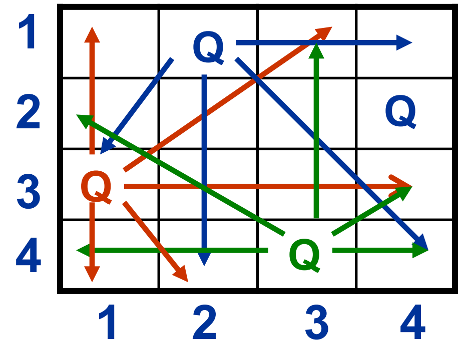
Bad News
Includes NP-Hard (intractable ?) problems.

Algorithmic Design Paradigm

Solving CSPs involves a combination of:

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

N-Queens

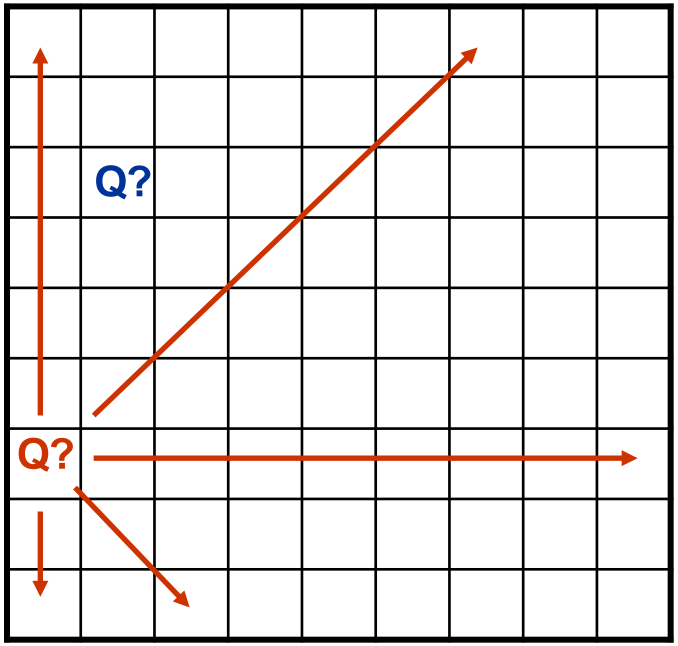


Inference

Eliminate values that can't be part of any solution

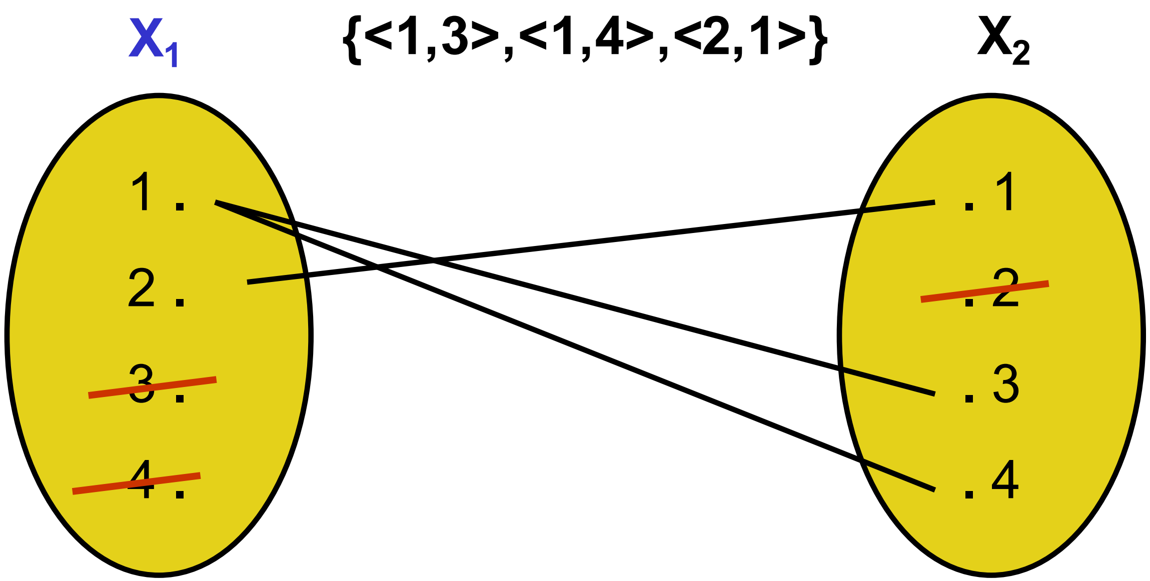
Search

Try alternative assignments against constraints



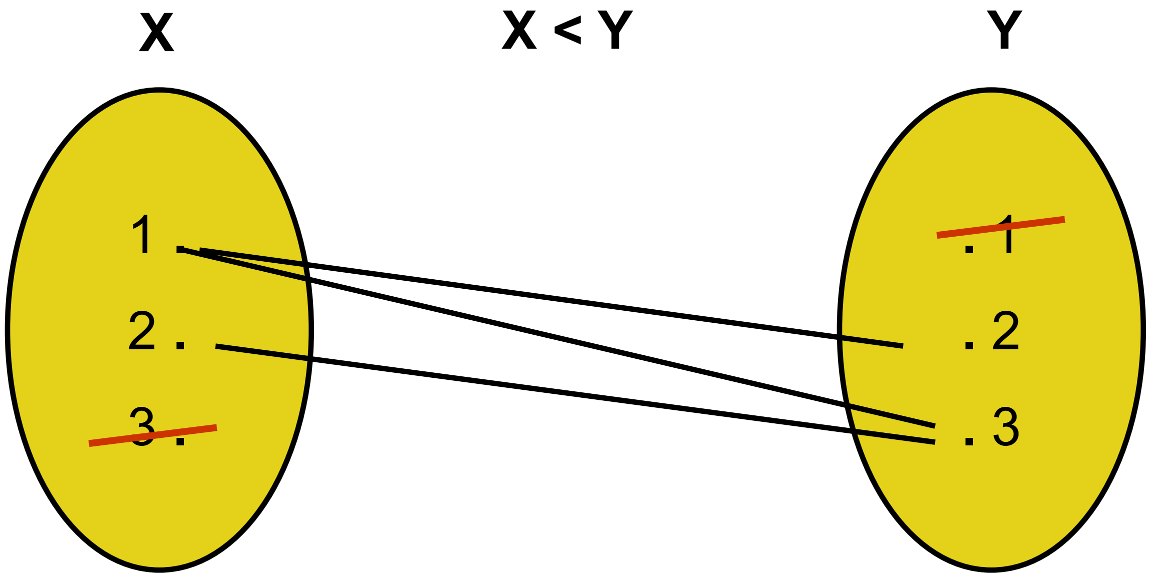
Arc Consistency

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

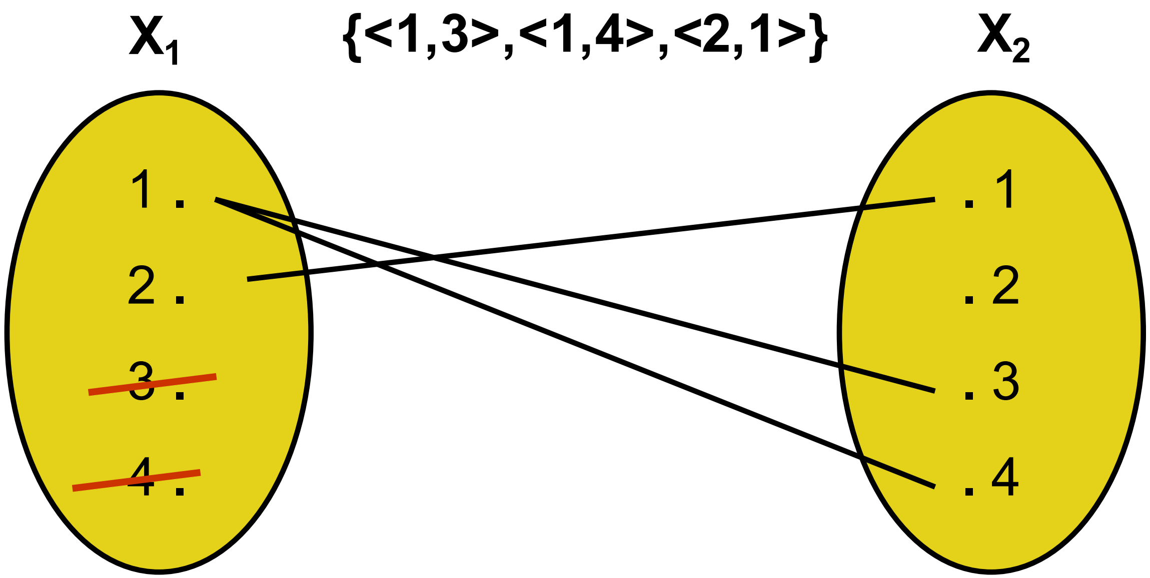


Definition: arc $<x_i, x_j>$ is arc consistent if $<x_i, x_j>$ and $<x_j, x_i>$ are directed arc consistent.

Arc Consistency



Directed Arc Consistency



Definition: arc $\langle x_i, x_j \rangle$ is directed arc consistent if

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

Revise: A directed arc consistency procedure

Definition: arc $\langle x_i, x_j \rangle$ is directed arc consistent if

$\forall a_i \in D_i, \exists a_j \in D_j$ such that $\langle a_i, a_j \rangle \in C_{ij}$.

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

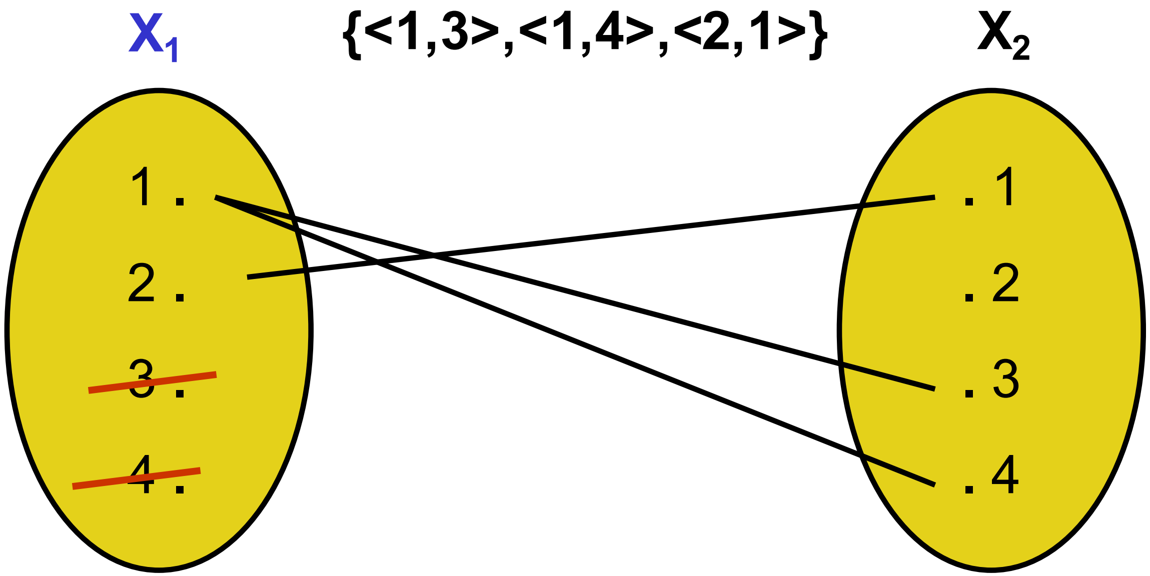
Constraint Processing,

by R. Dechter

pgs 54-56.

Directed Arc Consistency

Revise(x_1, x_2):



Now arc <x_1, x_2> is directed arc consistent.

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

Definition: Problem is arc consistent if all pairs of variables are arc consistent.

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.

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, and remove duplicates).

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

AC-3(CSP)

Input: A constraint satisfaction problem $CSP = \langle X, D, C \rangle$.

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

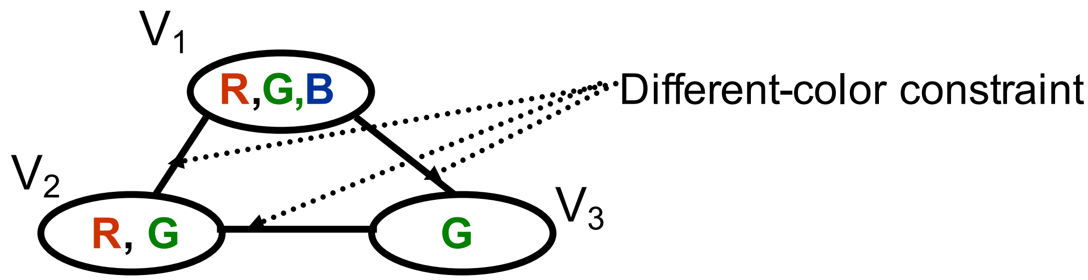
Constraint Processing,
by R. Dechter
pgs 58-59.

for every $c_{ij} \in C$,
  queue ← *queue* $\cup \{\langle x_i, x_j \rangle, \langle x_j, x_i \rangle\}$
endfor
while *queue* $\neq \{\}$
  select and delete arc $\langle x_i, x_j \rangle$ from *queue*
    Revise(x_i, x_j)
    if Revise(x_i, x_j) caused a change in D_i
      then *queue* ← *queue* $\cup \{\langle x_k, x_i \rangle \mid k \neq i, k \neq j\}$
    endif
endwhile

Constraint Propagation Example AC-3

Graph Coloring

Initial Domains

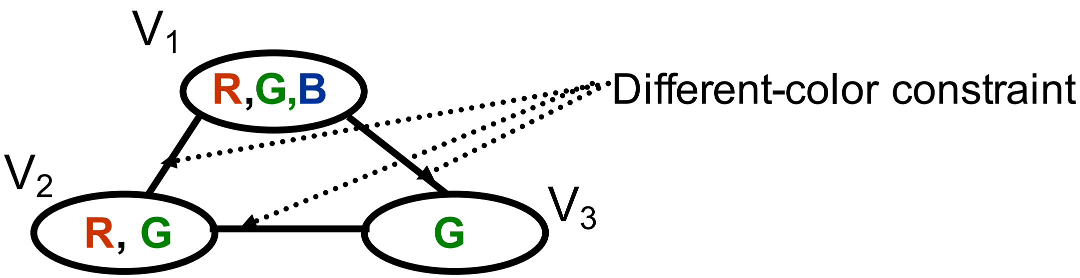


Each undirected arc denotes two directed arcs.

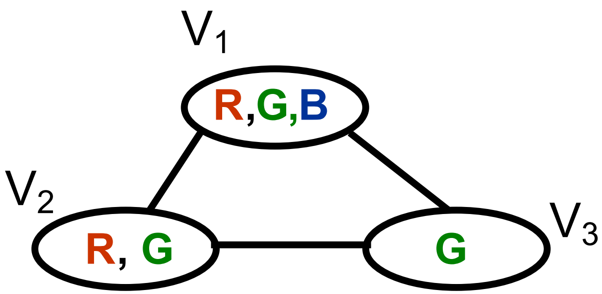
Constraint Propagation Example AC-3

Graph Coloring

Initial Domains



Arc examined	Value deleted



Arcs to examine

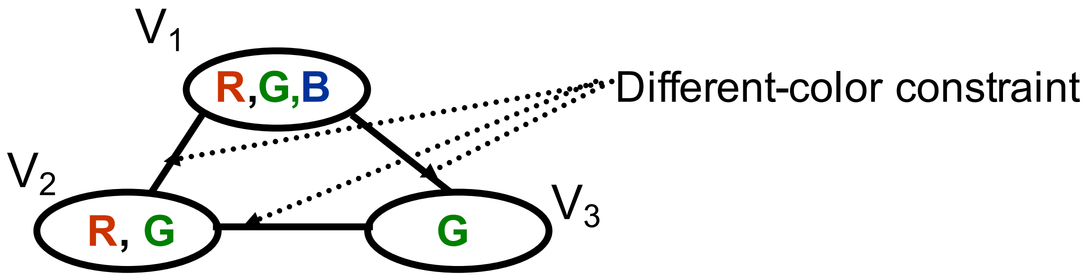
$V_1 - V_2$, $V_1 - V_3$, $V_2 - V_3$

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

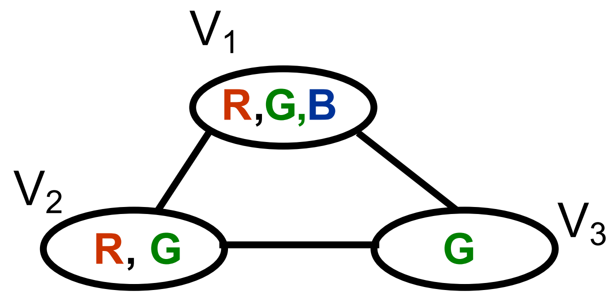
Constraint Propagation Example AC-3

Graph Coloring

Initial Domains



Arc examined	Value deleted



Arcs to examine

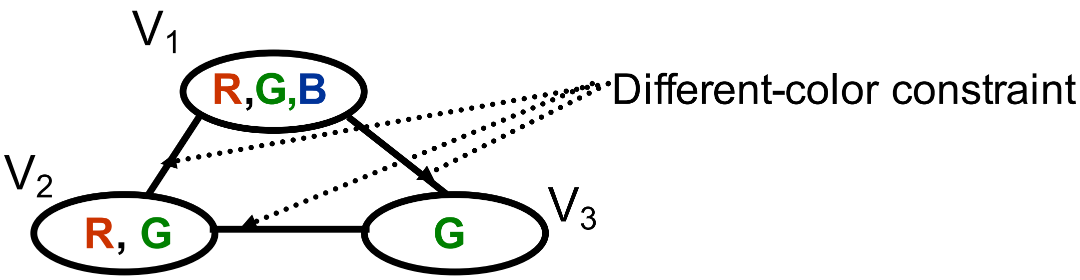
$V_1 - V_2$, $V_1 - V_3$, $V_2 - V_3$

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

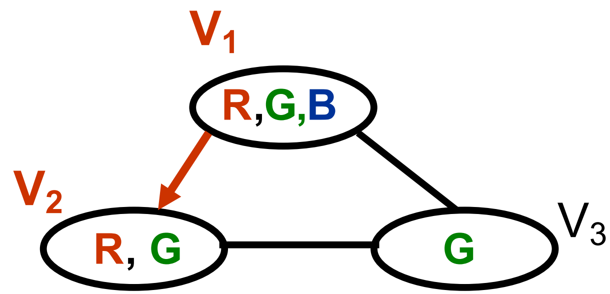
Constraint Propagation Example AC-3

Graph Coloring

Initial Domains



Arc examined	Value deleted
$V_1 > V_2$	



Arcs to examine

$V_2 > V_1, V_1 – V_3, V_2 – V_3$

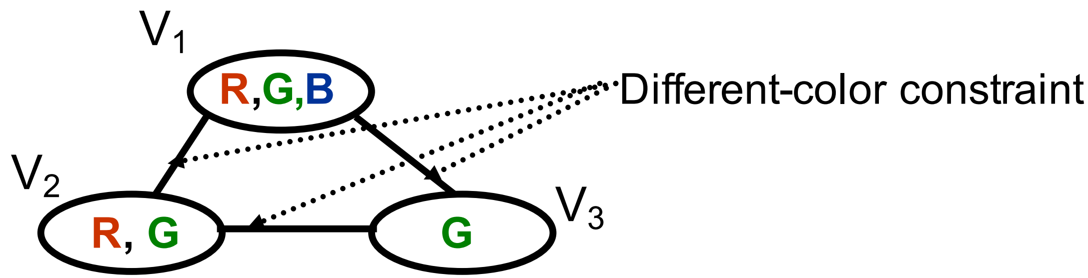
- Delete disallowed tail values.

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

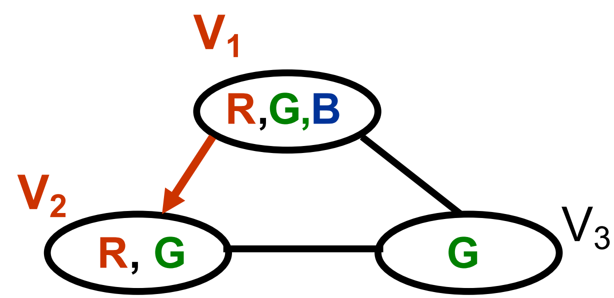
Constraint Propagation Example AC-3

Graph Coloring

Initial Domains



Arc examined	Value deleted
$V_1 > V_2$	none



Arcs to examine

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

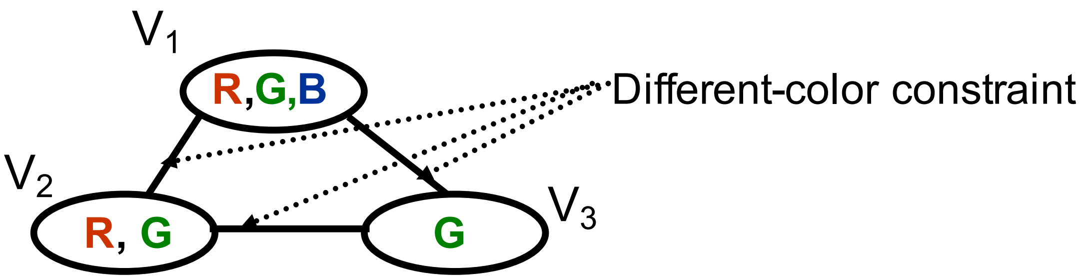
- Delete disallowed tail values.

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

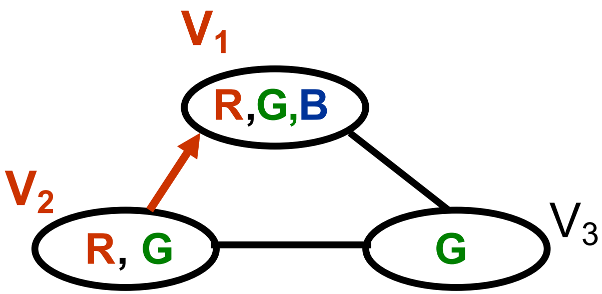
Constraint Propagation Example AC-3

Graph Coloring

Initial Domains



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



Arcs to examine

$V_1 - V_3$, $V_2 - V_3$

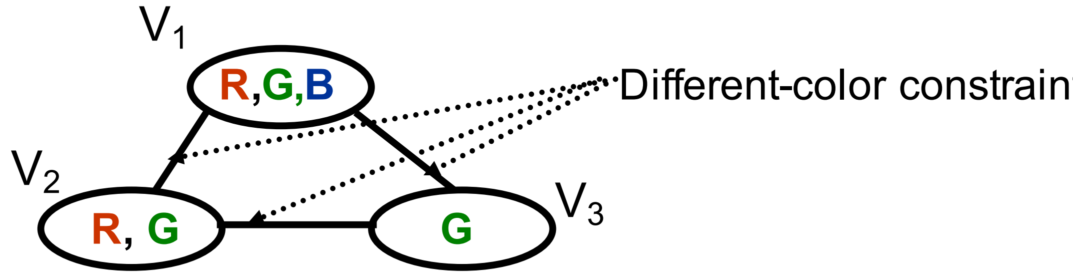
- Delete disallowed tail values.

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

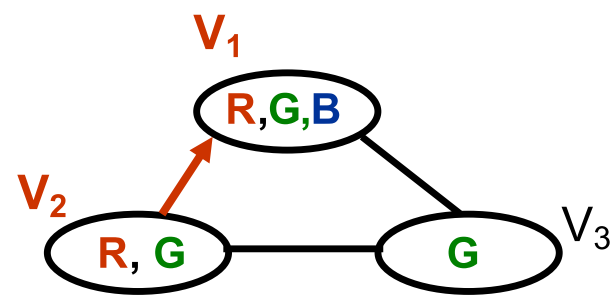
Constraint Propagation Example AC-3

Graph Coloring

Initial Domains



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



Arcs to examine

$V_1 - V_3$, $V_2 - V_3$

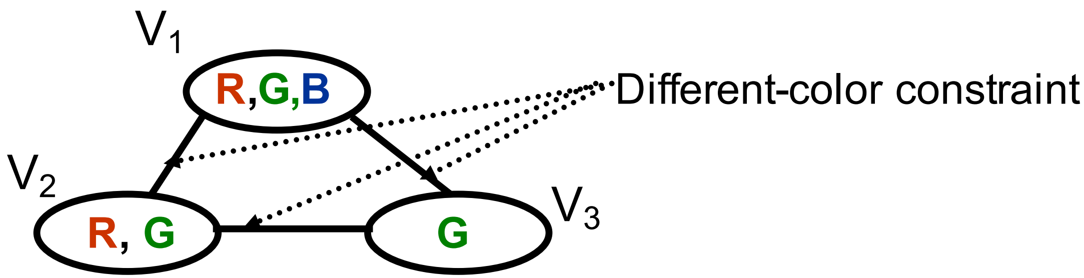
- Delete disallowed tail values.

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

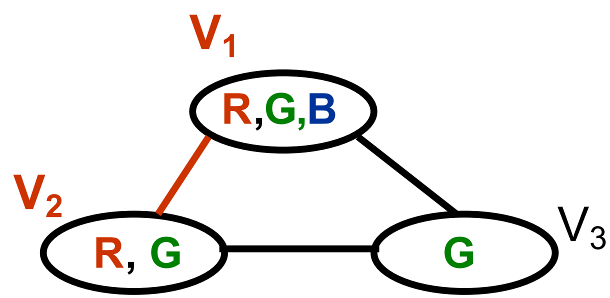
Constraint Propagation Example AC-3

Graph Coloring

Initial Domains



Arc examined	Value deleted
$V_1 - V_2$	none



Arcs to examine

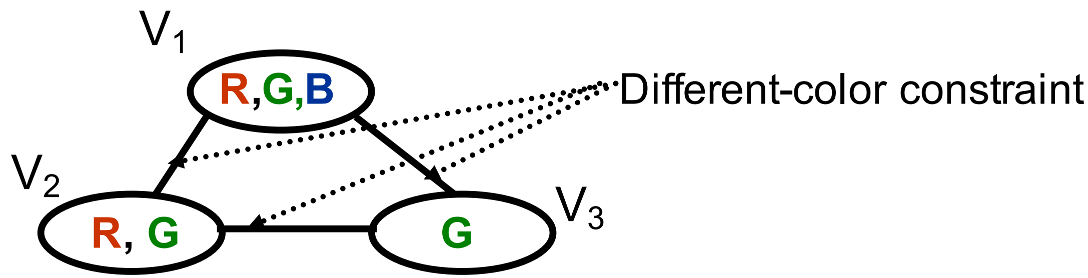
$V_1 - V_3$, $V_2 - V_3$

- Delete disallowed 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.

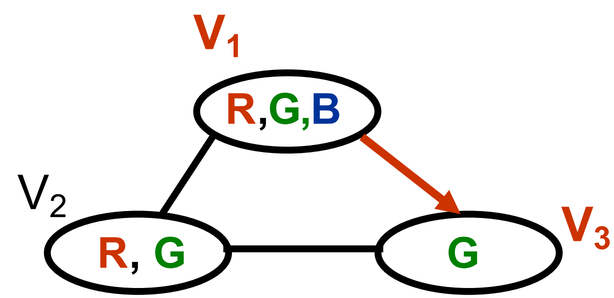
Constraint Propagation Example AC-3

Graph Coloring

Initial Domains



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



Arcs to examine

$V_3 > V_1$, $V_2 – V_3$

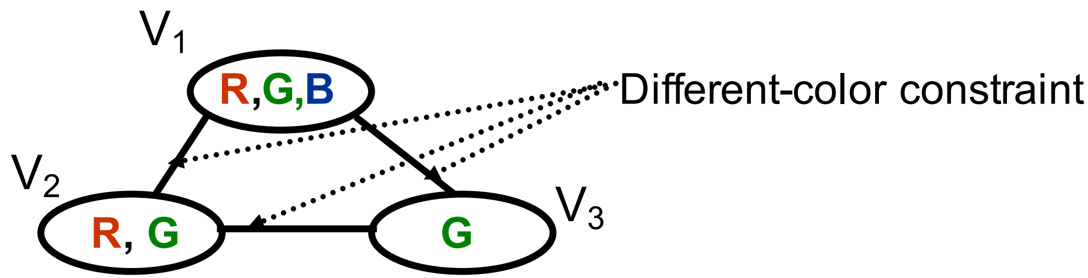
- Delete disallowed tail values.

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

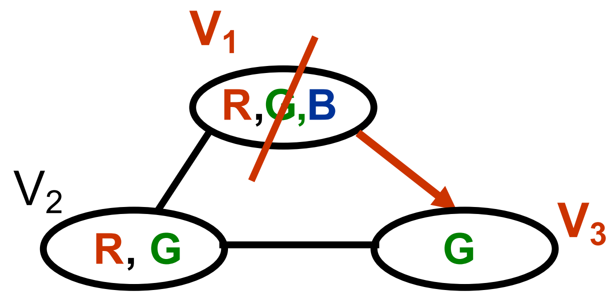
Constraint Propagation Example AC-3

Graph Coloring

Initial Domains



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



Arcs to examine

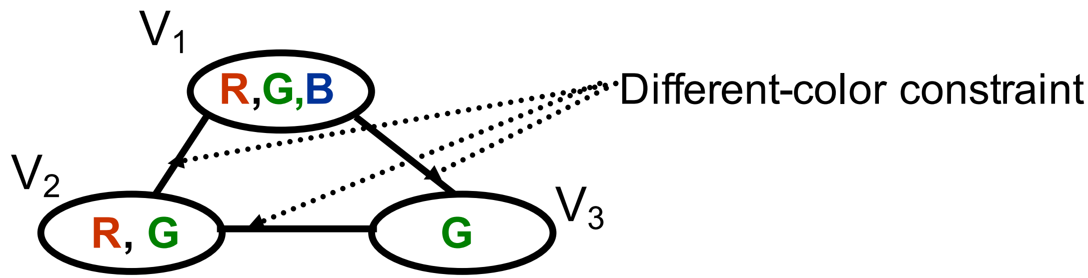
$V_3 > V_1$, $V_2 - V_3$

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

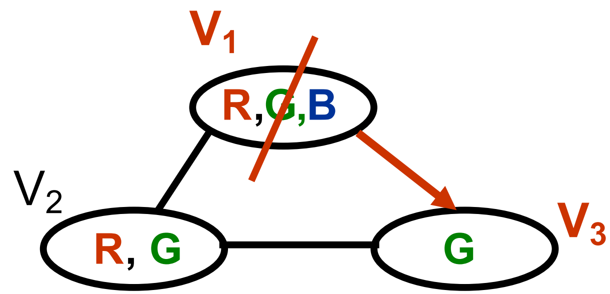
Constraint Propagation Example AC-3

Graph Coloring

Initial Domains



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



Arcs to examine

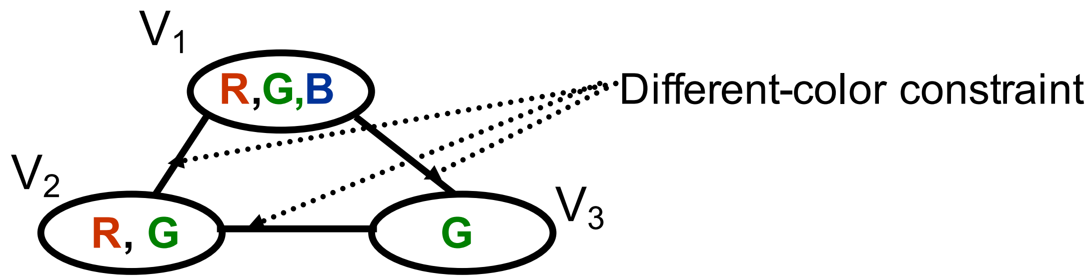
$V_3 > V_1$, $V_2 - V_3$, $V_2 > V_1$, ~~$V_3 > V_1$~~

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

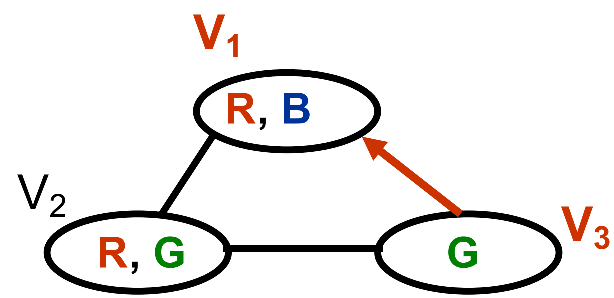
Constraint Propagation Example AC-3

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

$V_2 - V_3$, $V_2 > V_1$

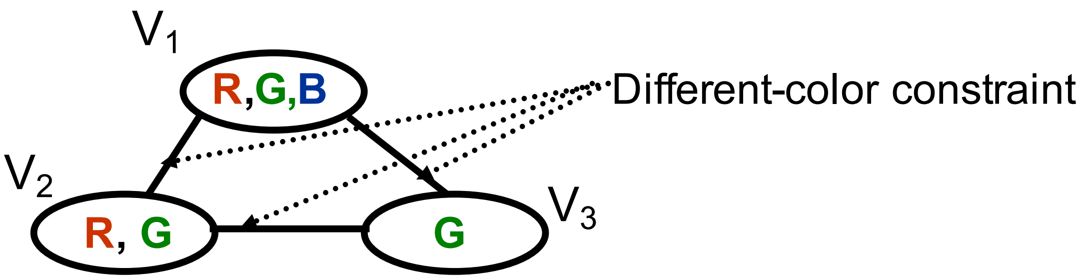
- Delete unmentioned tail values.

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

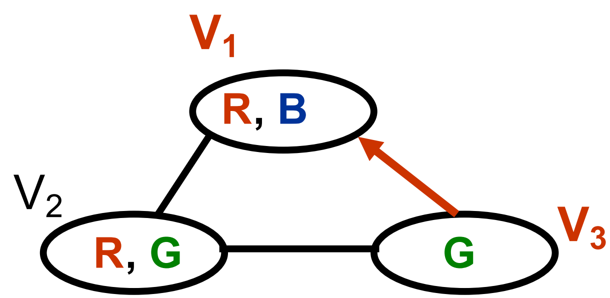
Constraint Propagation Example AC-3

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

$V_2 - V_3$, $V_2 > V_1$

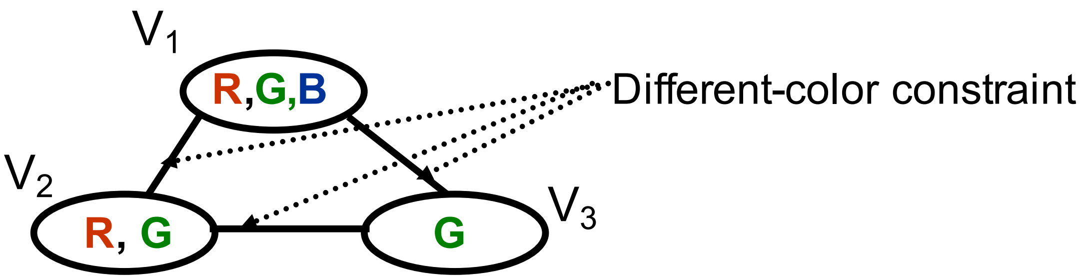
- Delete unmentioned tail values.

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

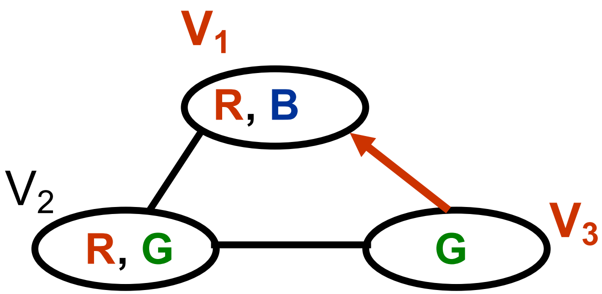
Constraint Propagation Example AC-3

Graph Coloring

Initial Domains



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



Arcs to examine

$V_2 - V_3$, $V_2 > V_1$

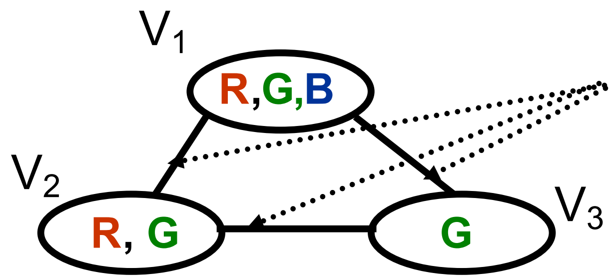
- Delete unmentioned tail values.

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

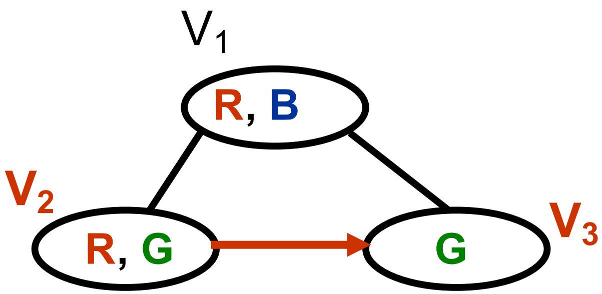
Constraint Propagation Example AC-3

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

$V_3 > V_2$, $V_2 > V_1$

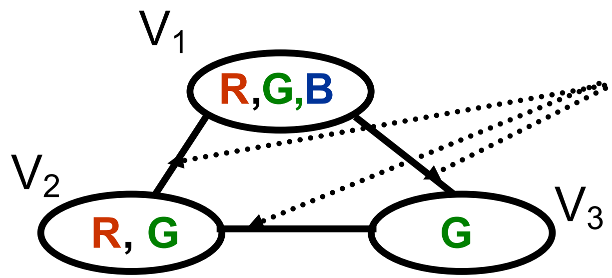
- Delete unmentioned tail values.

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

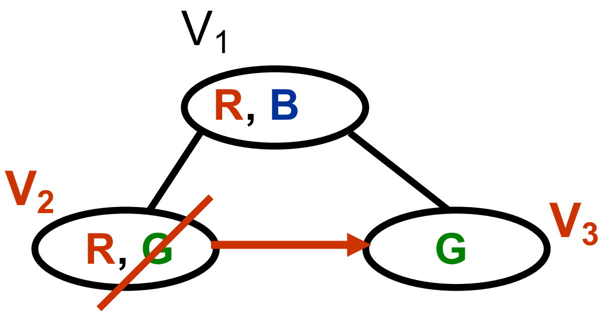
Constraint Propagation Example AC-3

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.

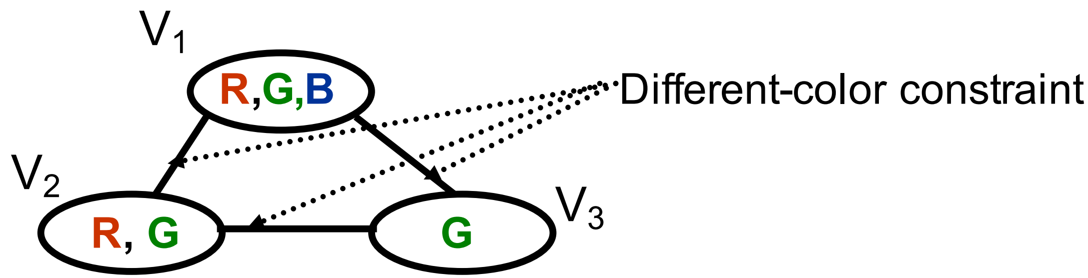
IF An element of a variable's domain is removed,

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

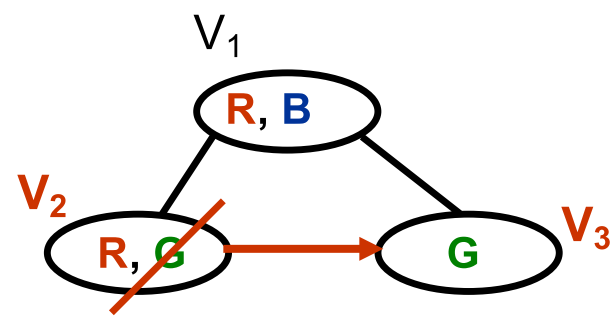
Constraint Propagation Example AC-3

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.

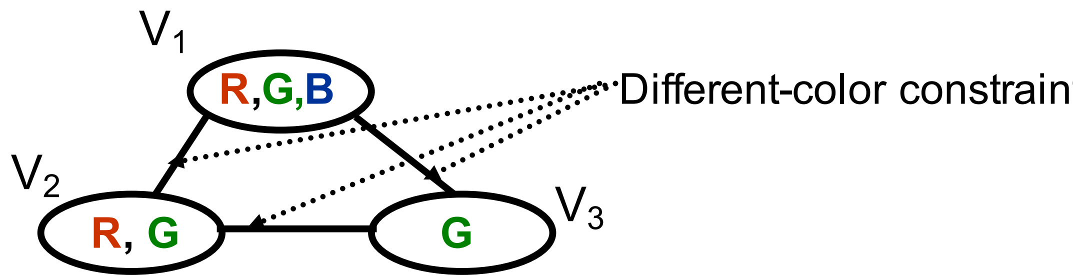
IF **An element of a variable's domain is removed,**

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

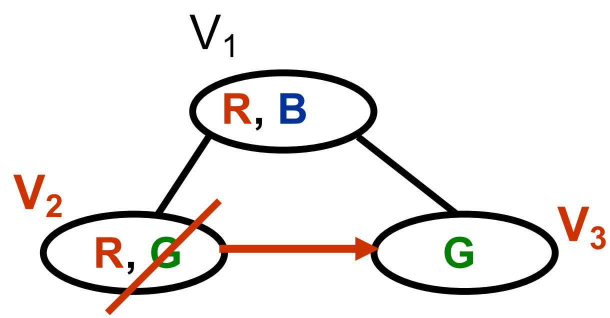
Constraint Propagation Example AC-3

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

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

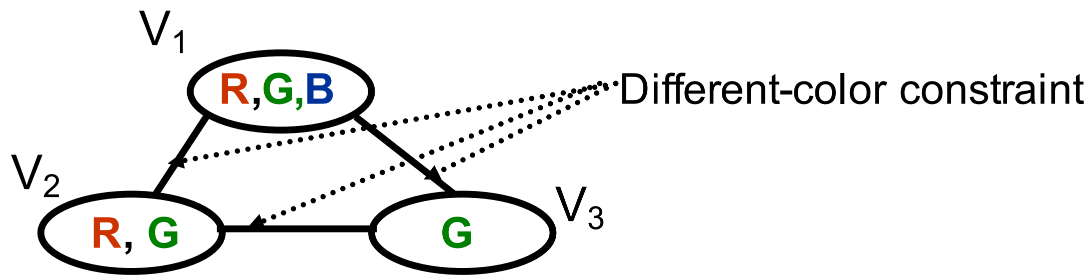
- Delete unmentioned tail values.

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

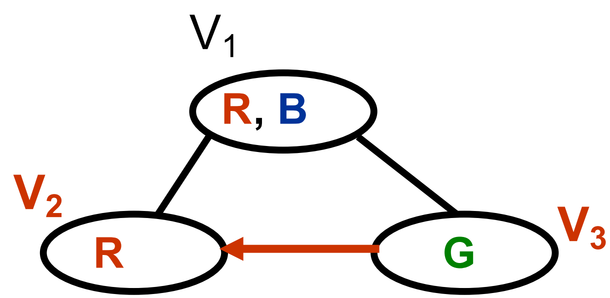
Constraint Propagation Example AC-3

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.

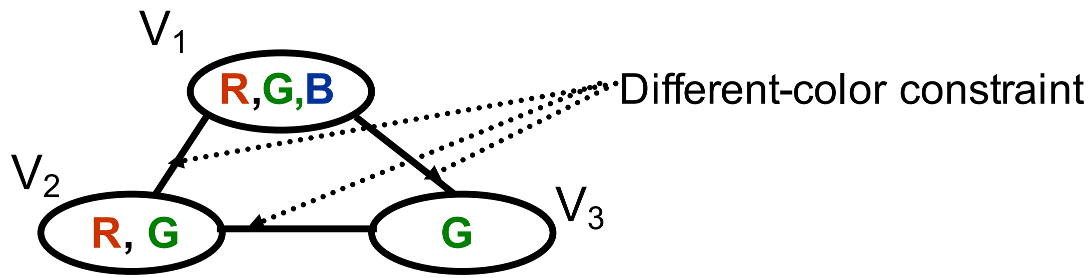
IF **An element of a variable's domain is removed,**

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

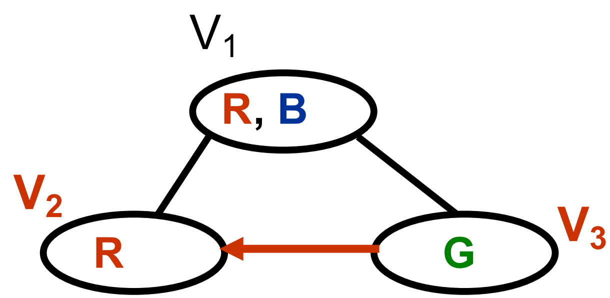
Constraint Propagation Example AC-3

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

$V_2 > V_1$, $V_1 > V_2$

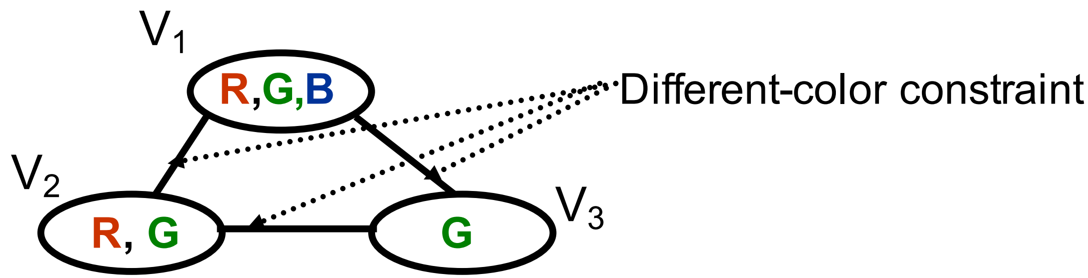
- Delete unmentioned tail values.

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

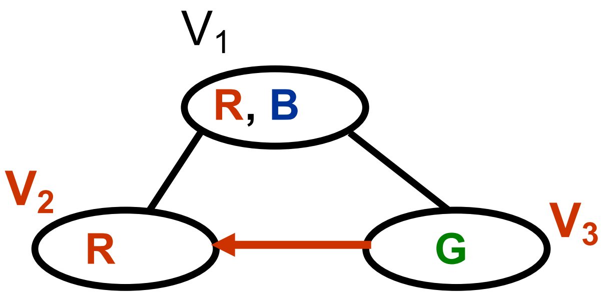
Constraint Propagation Example AC-3

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.

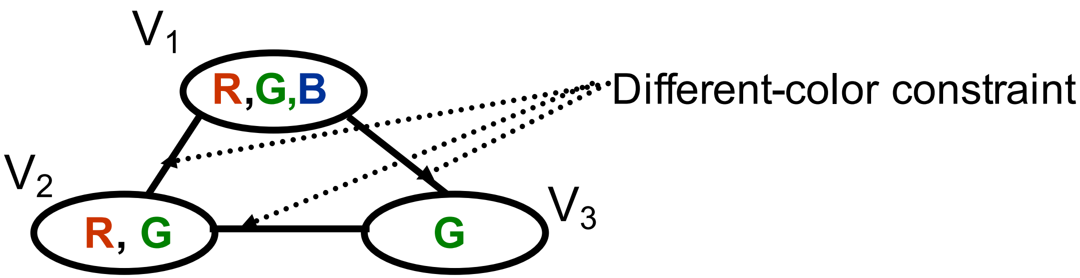
IF An element of a variable's domain is removed,

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

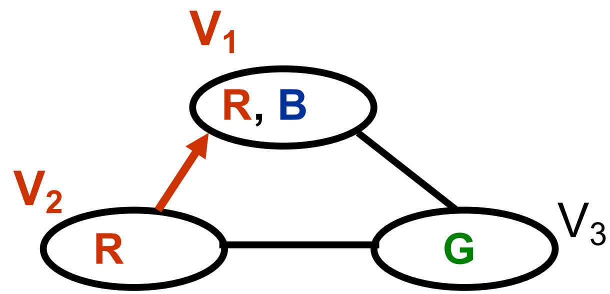
Constraint Propagation Example AC-3

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

$V_1 > V_2$

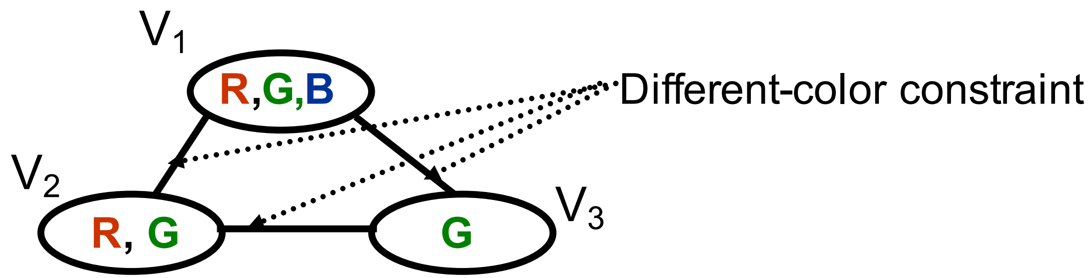
- Delete unmentioned tail values.

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

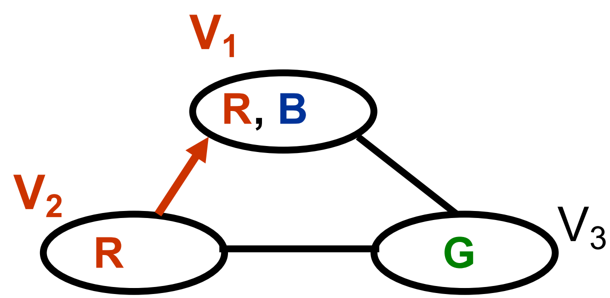
Constraint Propagation Example AC-3

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

$V_1 > V_2$

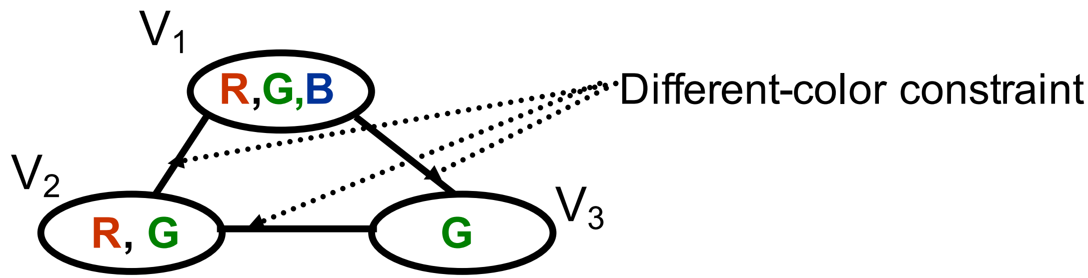
- Delete unmentioned tail values.

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

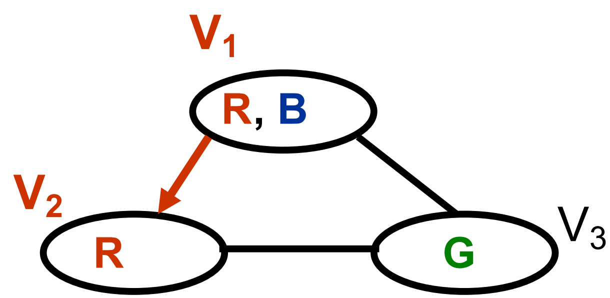
Constraint Propagation Example AC-3

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
$V_1 > V_2$	



Arcs to examine

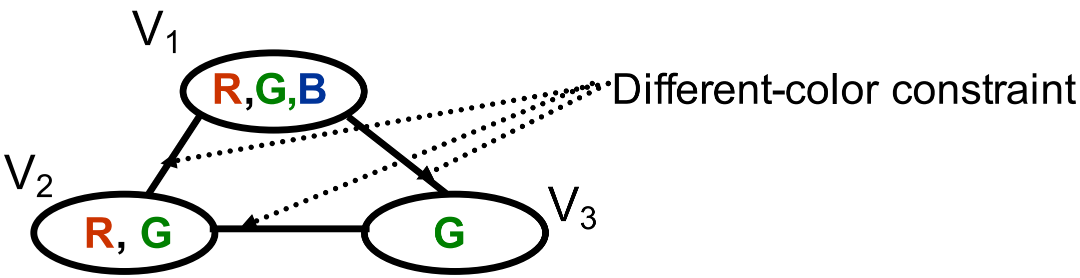
- Delete unmentioned tail values.

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

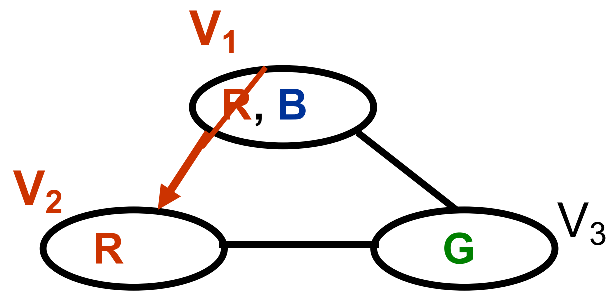
Constraint Propagation Example AC-3

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

- Delete unmentioned tail values.

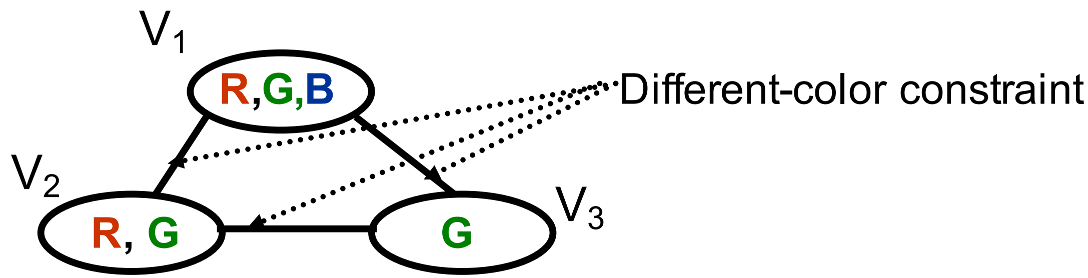
IF **An element of a variable's domain is removed,**

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

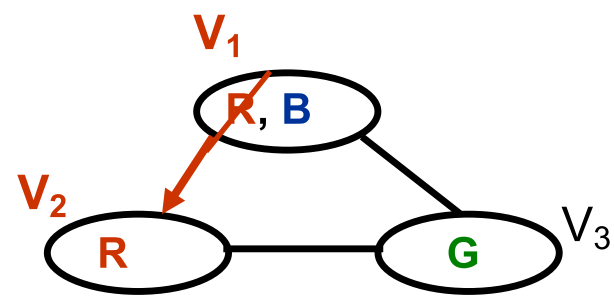
Constraint Propagation Example AC-3

Graph Coloring

Initial Domains



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



Arcs to examine

$V_2 > V_1$, $V_3 > V_1$

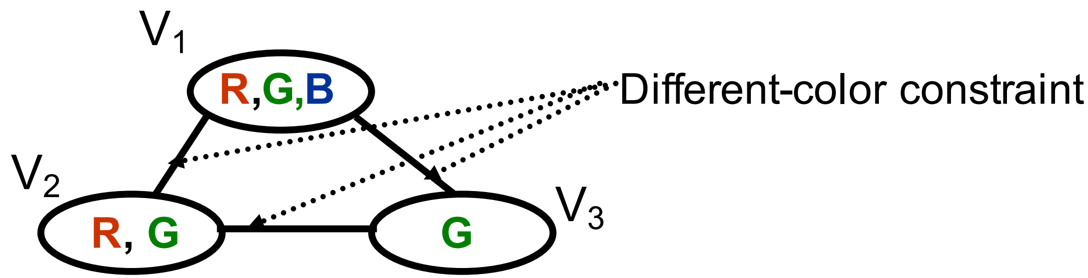
- Delete unmentioned tail values.

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

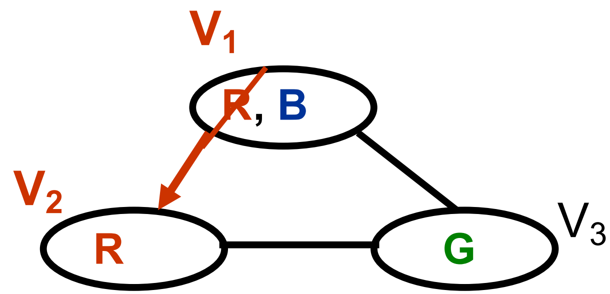
Constraint Propagation Example AC-3

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

$V_2 > V_1$, $V_3 > V_1$

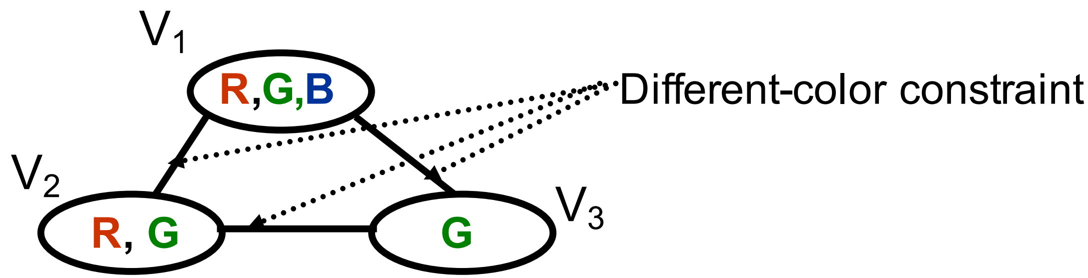
- Delete unmentioned tail values.

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

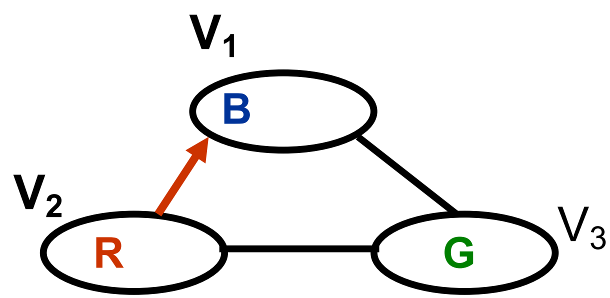
Constraint Propagation Example AC-3

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

$V_3 > V_1$

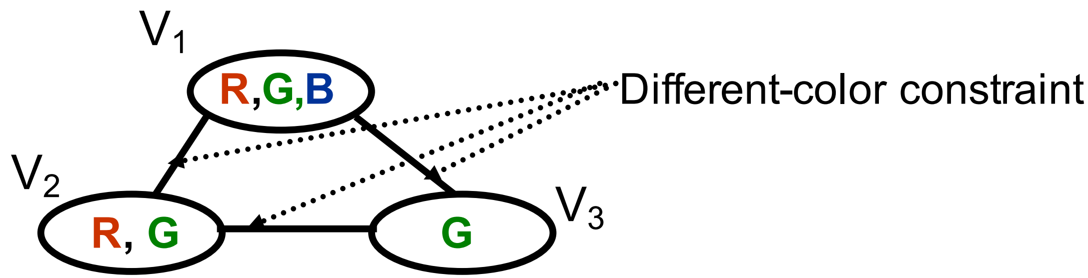
- Delete unmentioned tail values.

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

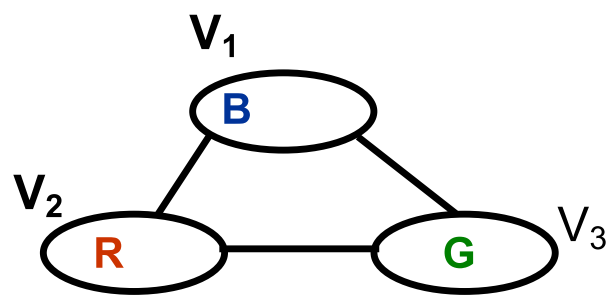
Constraint Propagation Example AC-3

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.

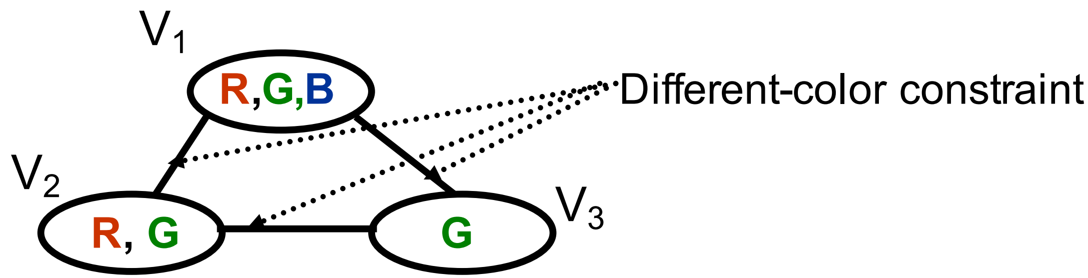
IF An element of a variable's domain is removed,

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

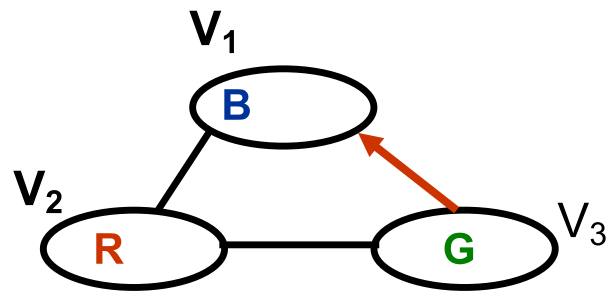
Constraint Propagation Example AC-3

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

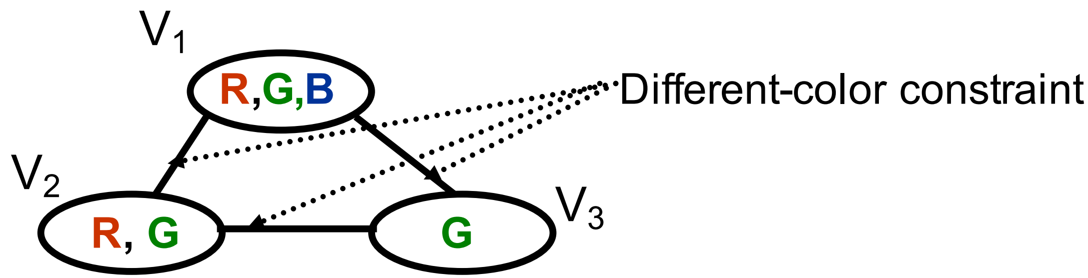
- Delete unmentioned tail values.

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

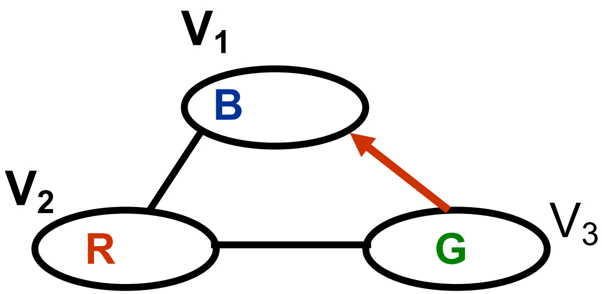
Constraint Propagation Example AC-3

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.

Outline

- Arc-consistency and constraint propagation.
- Analysis of constraint propagation.
- Solving CSPs using search.

What is the Complexity of AC-1?

AC-1(CSP)

Input: A network of constraints 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.**

Assume:

- **There are n variables.**
- **Domains are of size at most k.**
- **There are e binary constraints.**

What is the Complexity of AC-1?

Assume:

- There are n variables.
- Domains are of size at most k.
- There are e binary constraints.

Which is the correct complexity?

1. $O(k^2)$,
2. $O(enk^2)$,
3. $O(enk^3)$,
4. $O(nek)$.

Revise: A directed arc consistency procedure

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.

O(k)

1. **for** each $a_i \in D_i$
2.   **if** there is no $a_j \in D_j$ such that $\langle a_i, a_j \rangle \in R_{ij}$ *** O(k)**
3.   then delete a_i from D_i.
4.   **endif**
5. **endfor**

Complexity of Revise?

= $O(k^2)$.

where k = $\max_i |D_i|$

Full Arc-Consistency via AC-1

AC-1(CSP)

Input: A network of constraints CSP = <X, D, C>.

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

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

Complexity of AC-1?

= O(nk*e*revise),

= O(enk^3),

where $k = \max_i |D_i|$,

$n = |X|$, $e = |C|$.

What is the Complexity of Constraint Propagation using AC-3?

Assume:

- There are n variables.
- Domains are of size at most k.
- There are e binary constraints.

Which is the correct complexity?

1. $O(k^2)$,
2. $O(ek^2)$,
3. $O(ek^3)$,
4. $O(ek)$.

Full Arc-Consistency via AC-3

AC-3(CSP)

Input: A network of constraints CSP = $<X, D, C>$.

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

for every $c_{ij} \in C$, — O(e) +

  queue ← *queue* $\cup \{<x_i,x_j>, <x_i,x_j>\}$

endfor

while *queue* ≠ {}

  select and delete arc $<x_i, x_j>$ from *queue*

   Revise(x_i, x_j) — O(k²)

   if Revise(x_I, x_J) caused a change in D_i. — * O(ek)

    then *queue* ← *queue* $\cup \{<x_k,x_l> \mid k \neq i, k \neq j\}$

   endif

endwhile

Complexity of AC-3?

$= O(e+ek*k^2) = O(ek^3)$, where $k = \max_i |D_i|$, $n = |X|$, $e = |C|$.

Is arc consistency sound and complete?

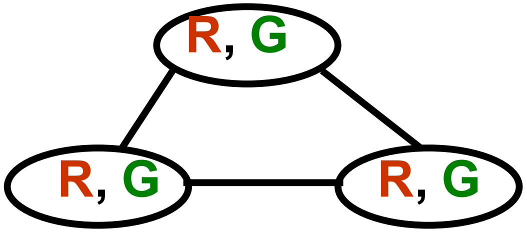
An *arc consistent solution* selects a value for every variable from its arc consistent domain.

Soundness: All solutions to the CSP are arc consistent solutions?

- Yes,
- No.

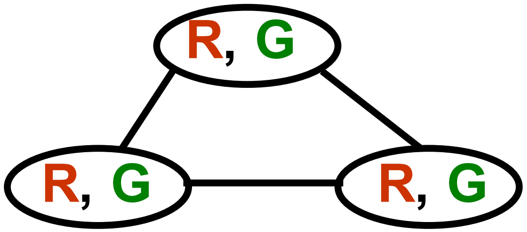
Completeness: All arc-consistent solutions are solutions to the CSP?

- Yes,
- No.

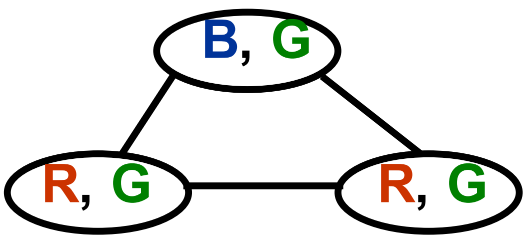


Incomplete: Arc consistency doesn't rule out all infeasible solutions

Graph Coloring



Arc consistent, but no solutions.



Arc consistent, but 2 solutions, not 8.

B,R,G
B,G,R

To Solve CSPs We Combine

1. Arc consistency (via 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 that Incorporate Search:

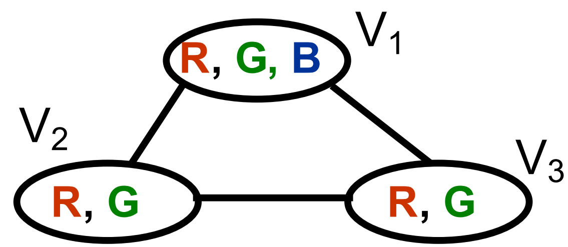
- Standard Search,
- Back Track Search (BT),
- BT with Forward Checking (FC),
- Dynamic Variable Ordering (DV),
- Iterative Repair (IR),
- Conflict-directed Back Jumping (CBJ).

Solving CSPs using Generic Search

- State
  - Partial assignment to variables, made thus far.
- Initial State
  - No assignment.
- Operator
  - Creates new assignment ≡ $(X_i = v_{ij})$.
    - Select any unassigned variable X_i .
    - Select any one of its domain values v_{ij} .
  - Child extends parent assignments with new.
- Goal Test
  - All variables are assigned.
  - All constraints are satisfied.

- Branching factor?

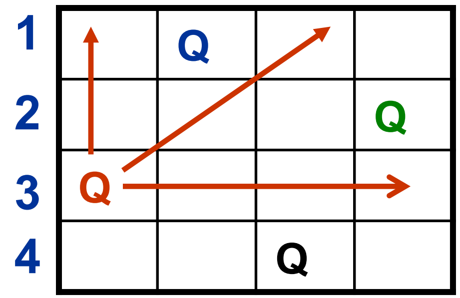
➔ **Sum of domain size of all variables** $O(|v|*|d|)$.



- Performance?

➔ **Exponential in the branching factor** $O([|v|*|d|]^{|v|})$.

Search Performance on N Queens



- Standard Search,
- Backtracking.

- A handful of queens.

Solving CSPs with Standard Search

Standard Search:

- Children select any value for any variable [O(|v|*|d|)].
- Test complete assignments for consistency against CSP.

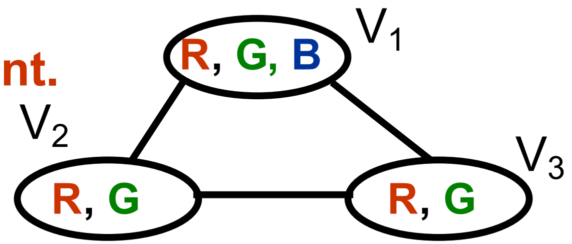
Observations:

1. The order in which variables are assigned does not change the solution.
   - **Many paths denote the same solution,**
     - **(|v|!).**

   ➔ **Expand only one path (i.e., use one variable ordering).**

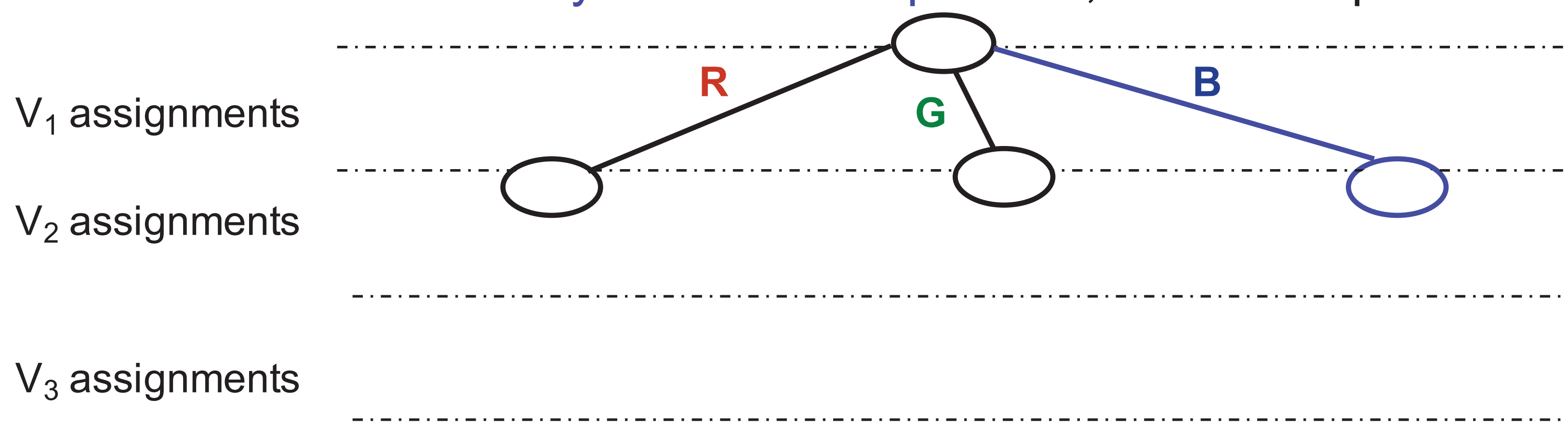
1. We can identify and prune a dead end before we assign all variables.
   - **Extensions to inconsistent partial assignments are always inconsistent.**

   ➔ **Check consistency after each assignment.**



Next: Back Track Search (BT)

1. Expand assignments of one variable at each step.
2. Pursue depth first.
3. Check consistency after each expansion, and backup.



Preselect order of variables to assign.

Assign designated variable.