Consistency algorithms

Chapter 3

Outline

- Arc-consistency algorithms
- Path-consistency and i-consistency
- Generalized arc-consistency, relational arc-consistency
- Global and bound consistency
- Distributed (generalized) arc-consistency
- Consistency operators: join, resolution, Gausian elimination

Consistency methods

- Approximation of inference:
  - Arc, path and i-consistecy
- Methods that transform the original network into tighter and tighter representations

Arc-consistency

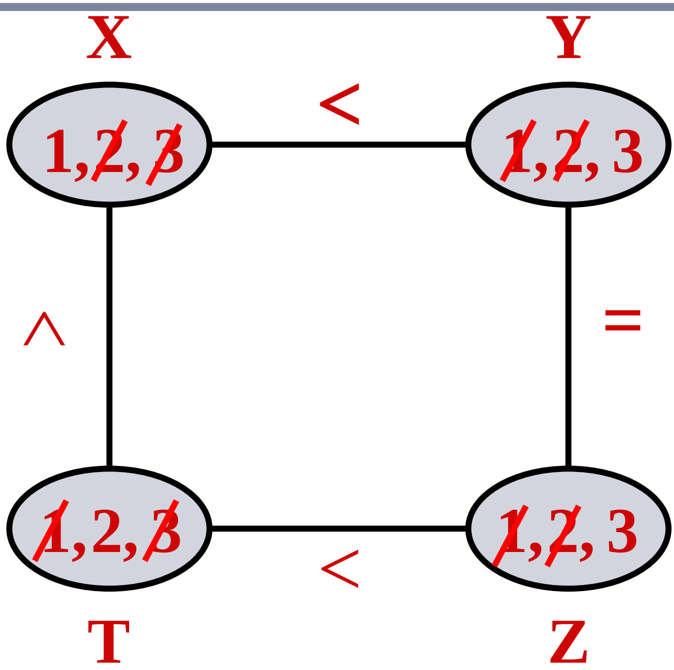
$1 \le X, Y, Z, T \le 3$

$X < Y$

$Y = Z$

$T < Z$

$X \le T$



Arc-consistency

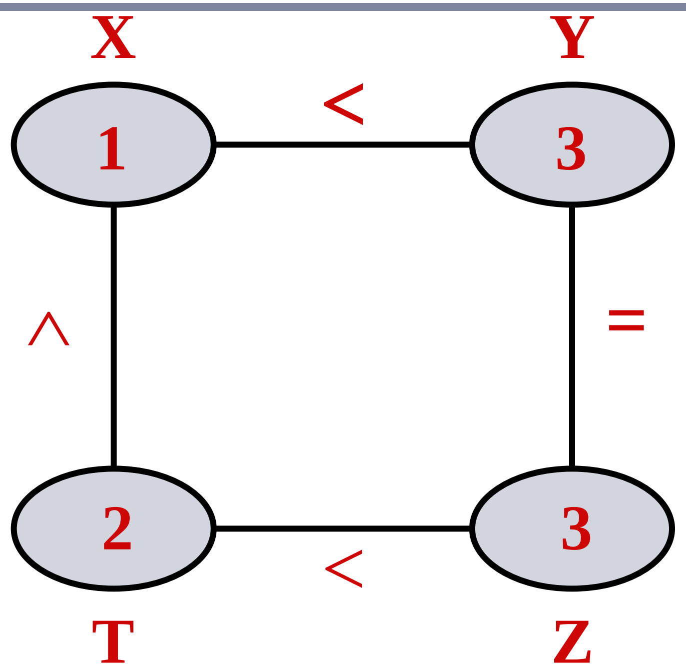
$1 \leq X, Y, Z, T \leq 3$

$X < Y$

$Y = Z$

$T < Z$

$X \leq T$



Arc-consistency

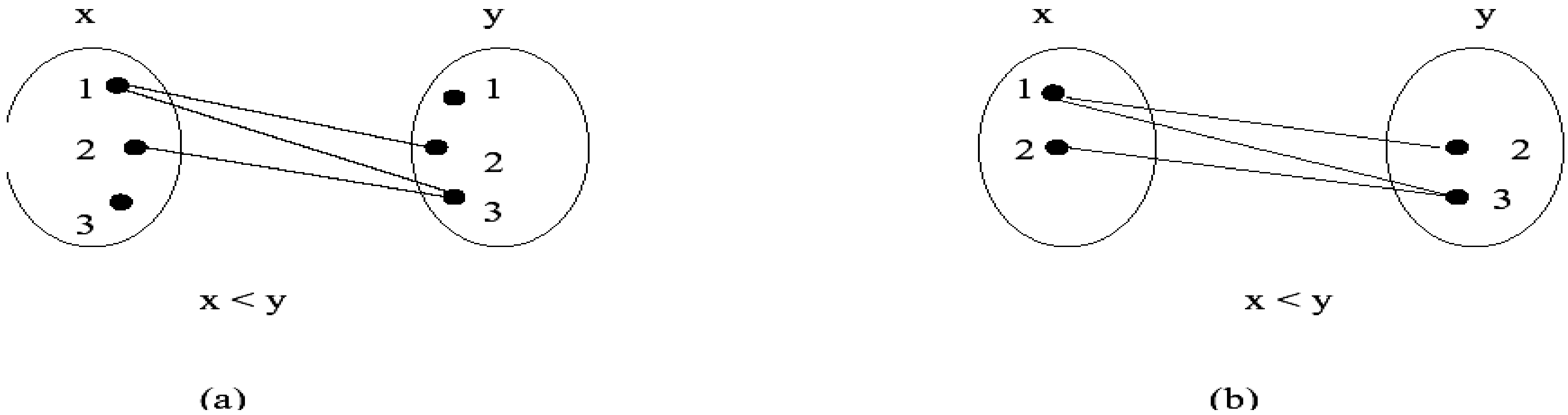


Figure 3.1: A matching diagram describing the arc-consistency of two variables x and y. In (a) the variables are not arc-consistent. In (b) the domains have been reduced, and the variables are now arc-consistent.

Definition 3.2.2 (arc-consistency) *Given a constraint network* $\mathcal{R} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$, *with* $R_{ij} \in C$, *a variable* x_i *is* arc-consistent *relative to* x_j *if and only if for every value* $a_i \in D_i$ *there exists a value* $a_j \in D_j$ *such that* $(a_i, a_j) \in R_{ij}$. *The subnetwork (alternatively, the arc) defined by* $\{x_i, x_j\}$ *is arc-consistent if and only if* x_i *is arc-consistent relative to* x_j *and* x_j *is arc-consistent relative to* x_i. *A network of constraints is called* arc-consistent *iff all of its arcs (e.g., subnetworks of size 2) are arc-consistent.*

Inference: Join and Project

x_1	x_2	x_3
a	b	c
b	b	c
c	b	c
c	b	s

(a) Relation R

x_1	x_2	x_3
b	b	c
c	b	c
c	n	n

(b) Relation R'

x_2	x_3	x_4
a	a	1
b	c	2
b	c	3

(c) Relation R''

project

x_2	x_3
b	c
n	n

(b) $\pi_{\{x_2,x_3\}}(R')$

join

x_1	x_2	x_3	x_4
b	b	c	2
b	b	c	3
c	b	c	2
c	b	c	3

(c) $R' \bowtie R''$

Revise for arc-consistency

$\text{REVISE}((x_i), x_j)$

input: a subnetwork defined by two variables $X = \{x_i, x_j\}$, a distinguished variable x_i, domains: D_i and D_j, and constraint R_{ij}

output: D_i, such that, x_i arc-consistent relative to x_j

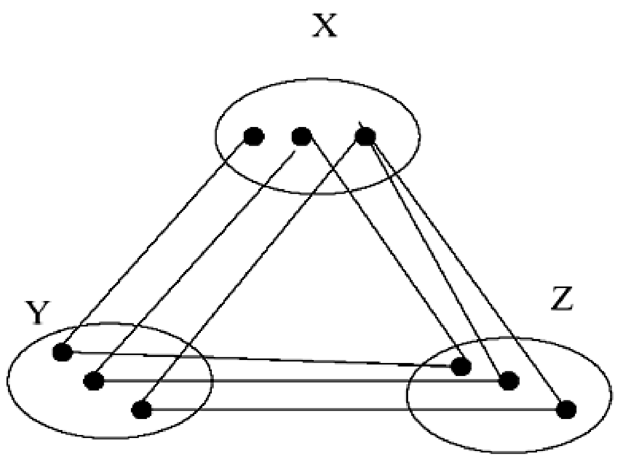
1. **for** each $a_i \in D_i$
2.   **if** there is no $a_j \in D_j$ such that $(a_i, a_j) \in R_{ij}$
3.   **then** delete a_i from D_i
4.   **endif**
5. **endfor**

Figure 3.2: The Revise procedure

$\bowtie = \otimes$

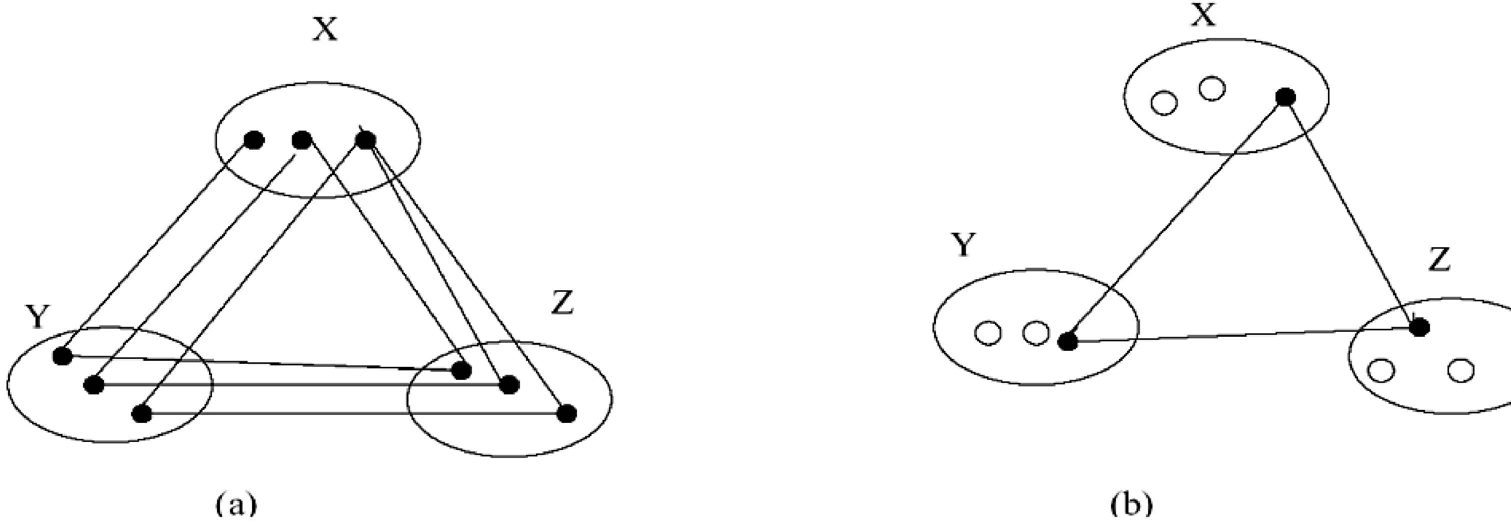
$$D_i \leftarrow D_i \cap \pi_i(R_{ij} \otimes D_j)$$

A matching diagram describing a network of constraints that is not arc-consistent (b) An arc-consistent equivalent network.



(a)

A matching diagram describing a network of constraints that is not arc-consistent (b) An arc-consistent equivalent network.



AC-1

AC-1($\mathcal{R}$)

input: a network of constraints $\mathcal{R} = (X, D, C)$
output: $\mathcal{R}'$ which is the loosest arc-consistent network equivalent to $\mathcal{R}$

1. **repeat**
2. **for** every pair $\{x_i, x_j\}$ that participates in a constraint
3. Revise($(x_i), x_j$) (or $D_i \leftarrow D_i \cap \pi_i(R_{ij} \bowtie D_j)$)
4. Revise($(x_j), x_i$) (or $D_j \leftarrow D_j \cap \pi_j(R_{ij} \bowtie D_i)$)
5. **endfor**
6. **until** no domain is changed

Figure 3.4: Arc-consistency-1 (AC-1)

- Complexity (Mackworth and Freuder, 1986): $O(enk^3)$
- e = number of arcs, n variables, k values
- $(ek^2$, each loop, nk number of loops), best-case = ek,
- Arc-consistency is: $\Omega(ek^2)$

AC-3

AC-3($\mathcal{R}$)

input: a network of constraints $\mathcal{R} = (X, D, C)$

output: $\mathcal{R}'$ which is the largest arc-consistent network equivalent to $\mathcal{R}$

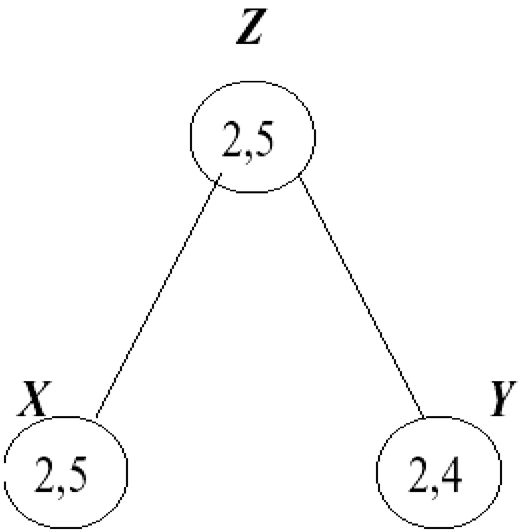
for every pair $\{x_i, x_j\}$ that participates in a constraint $R_{ij} \in \mathcal{R}$
  $queue \leftarrow queue \cup \{(x_i, x_j), (x_j, x_i)\}$
endfor
while $queue \neq \{\}$
  select and delete (x_i, x_j) from $queue$
  $Revise((x_i), x_j)$
  if $Revise((x_i), x_j)$ causes a change in D_i
    then $queue \leftarrow queue \cup \{(x_k, x_i), i \neq k\}$
  endif
endwhile

Figure 3.5: Arc-consistency-3 (AC-3)

- Complexity: $O(ek^3)$ since each arc may be processed in O(2k)
- Best case O(ek),

Example: A 3 variables network with 2 constraints: z divides x and z divides y

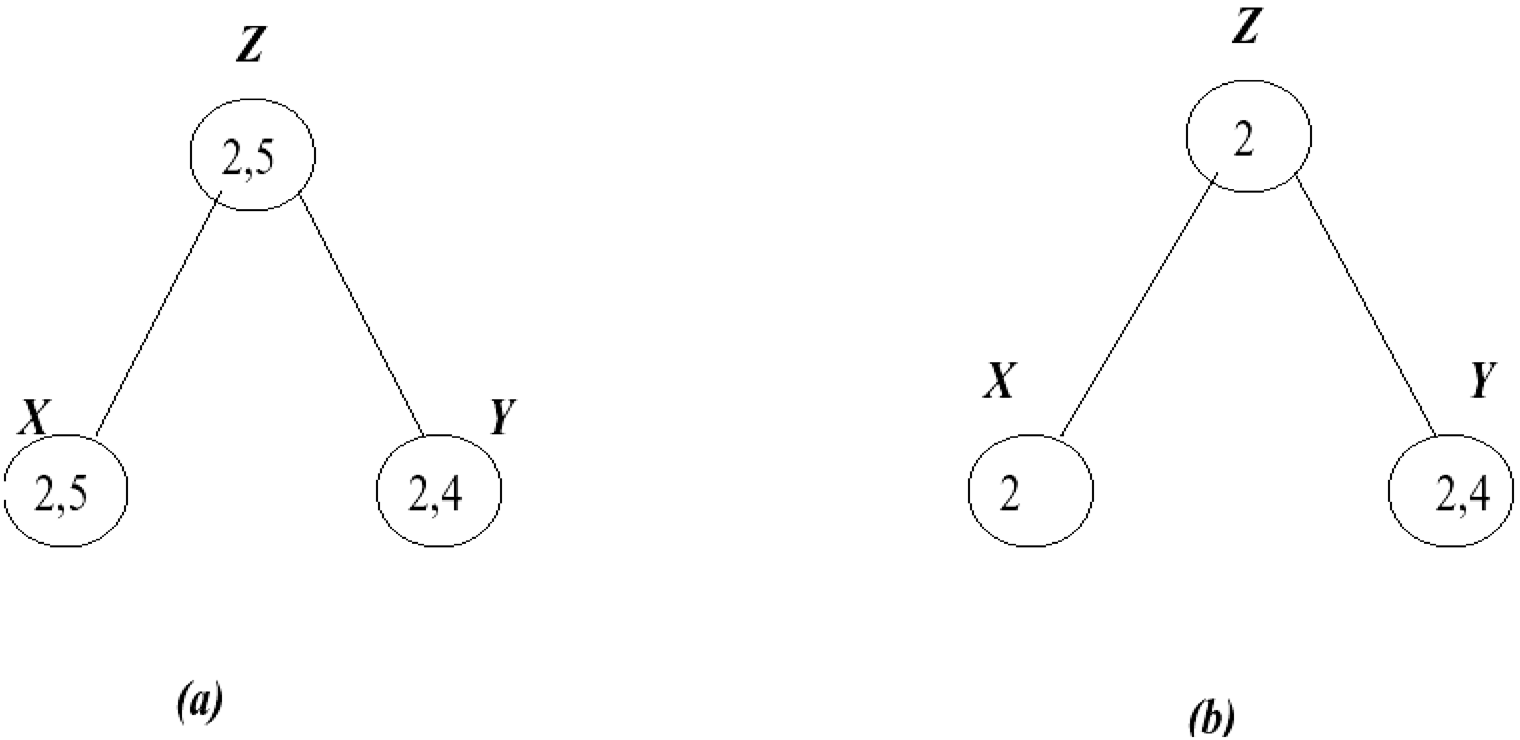
(a) before and (b) after AC-3 is applied.



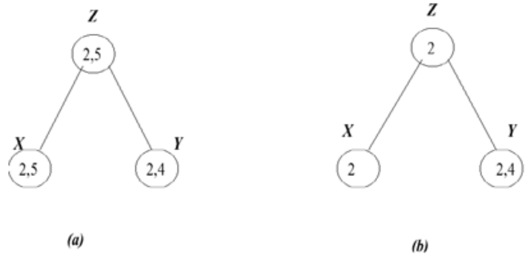
(a)

Example: A 3 variables network with 2 constraints: z divides x and z divides y

(a) before and (b) after AC-3 is applied.



AC-4



AC-4($\mathcal{R}$)
input: a network of constraints $\mathcal{R}$
output: An arc-consistent network equivalent to $\mathcal{R}$

Initialization: $M \leftarrow \emptyset$,
    initialize $S_{(x_i,c_i)}$, $counter(i, a_i, j)$ for all R_{ij}
    for all counters
        if $counter(x_i, a_i, x_j) = 0$ (if $< x_i, a_i >$ is unsupported by x_j)
            then add $< x_i, a_i >$ to $LIST$
        endif
    endfor
while $LIST$ is not empty
    choose $< x_i, a_i >$ from LIST, remove it, and add it to M
    for each $< x_j, a_j >$ in $S_{(x_i,a_i)}$
        decrement $counter(x_j, a_j, x_i)$
        if $counter(x_j, a_j, x_i) = 0$
            then add $< x_j, a_j >$ to $LIST$
        endif
    endfor
endwhile

Figure 3.7: Arc-consistency-4 (AC-4)

- Complexity: $O(ek^2)$
- (Counter is the number of supports to a_i in x_i from x_j. $S_{(xi,ai)}$ is the set of pairs that (x_i, a_i) supports)

Example applying AC-4

Example 3.2.9 Consider the problem in Figure 3.6. Initializing the $S_{(x,a)}$ arrays (indicating all the variable-value pairs that each $< x, a >$ supports), we have : $S_{(z,2)} = \{< x, 2 >, < y, 2 >, < y, 4 >\}$, $S_{(z,5)} = \{< x, 5 >\}$, $S_{(x,2)} = \{< z, 2 >\}$, $S_{(x,5)} = \{< z, 5 >\}$, $S_{(y,2)} = \{< z, 2 >\}$, $S_{(y,4)} = \{< z, 2 >\}$.
For counters we have: $counter(x, 2, z) = 1$, $counter(x, 5, z) = 1$, $counter(z, 2, x) = 1$, $counter(z, 5, x) = 1$, $counter(z, 2, y) = 2$, $counter(z, 5, y) = 0$, $counter(y, 2, z) = 1$, $counter(y, 4, z) = 1$. (Note that we do not need to add counters between variables that are not directly constrained, such as x and y.) Finally, $List = \{< z, 5 >\}$, $M = \emptyset$. Once $< z, 5 >$ is removed from $List$ and placed in M, the counter of $< x, 5 >$ is updated to $counter(x, 5, z) = 0$, and $< x, 5 >$ is placed in $List$. Then, $< x, 5 >$ is removed from $List$ and placed in M. Since the only value it supports is $< z, 5 >$ and since $< z, 5 >$ is already in M, the $List$ remains empty and the process stops. □

Distributed arc-consistency (Constraint propagation)

- Implement AC-1 distributedly.
- $h_{j \to i}$ node x_j sends the message to node x_i
- Node x_i updates its domain:
- Messages can be sent asynchronously or scheduled in a topological order

$$D_i \leftarrow D_i \cap \pi_i(R_{ij} \otimes D_j)$$

$$h_i^j \leftarrow \pi_i(R_{ij} \otimes D_j)$$

$$D_i \leftarrow D_i \cap \pi_i(R_{ij} \otimes D_j) =$$

$$D_i \leftarrow D_i \cap h_i^j$$

Exercise: make the following network arc-consistent

- Draw the network's primal and dual constraint graph
- Network =
  - Domains {1,2,3,4}
  - Constraints: $y < x$, $z < y$, $t < z$, $f<t$, $x<=t+1$, $Y<f+2$

Arc-consistency Algorithms

- **AC-1**: brute-force, distributed $O(nek^3)$
- **AC-3**, queue-based $O(ek^3)$
- **AC-4**, context-based, optimal $O(ek^2)$
- **AC-5,6,7**,…. Good in special cases
- **Important:** applied at every node of search
- (*n* number of variables, *e*=#constraints, *k*=domain size)
- Mackworth and Freuder (1977,1983), Mohr and Anderson, (1985)…

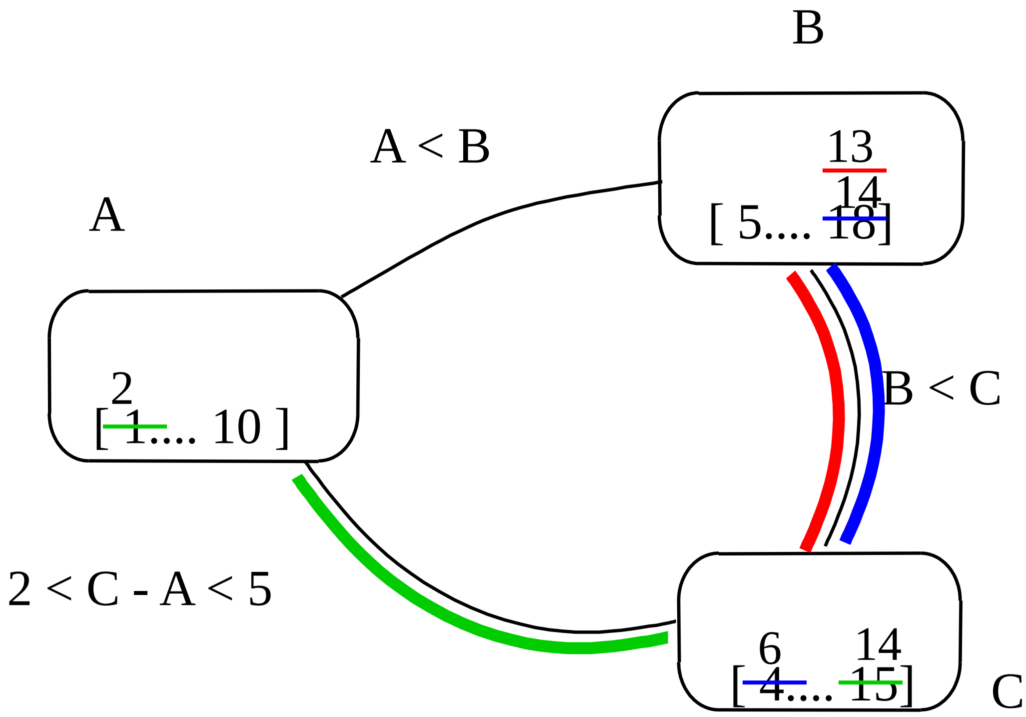
Using constraint tightness in analysis

t = number of tuples bounding a constraint

- **AC-1**: brute-force, $O(nek^3)$ $O(nekt)$
- **AC-3**, queue-based $O(ek^3)$ $O(ekt)$
- **AC-4**, context-based, optimal $O(et)$
- **AC-5,6,7**,…. Good in special cases
- **Important:** applied at every node of search
- (n number of variables, e=#constraints, k=domain size)
- Mackworth and Freuder (1977,1983), Mohr and Anderson, (1985)…

Constraint checking

→ Arc-consistency



1- B: [5 .. 14]

C: [6 .. 15]

2- A: [2 .. 10]

C: [6 .. 14]

3- B: [5 .. 13]

Is arc-consistency enough?

- Example: a triangle graph-coloring with 2 values.
  - Is it arc-consistent?
  - Is it consistent?
- It is not path, or 3-consistent.

Outline

- Arc-consistency algorithms
- Path-consistency and i-consistency
- Arc-consistency, Generalized arc-consistency, relation arc-consistency
- Global and bound consistency
- Distributed (generalized) arc-consistency
- Consistency operators: join, resolution, Gausian elimination

Path-consistency

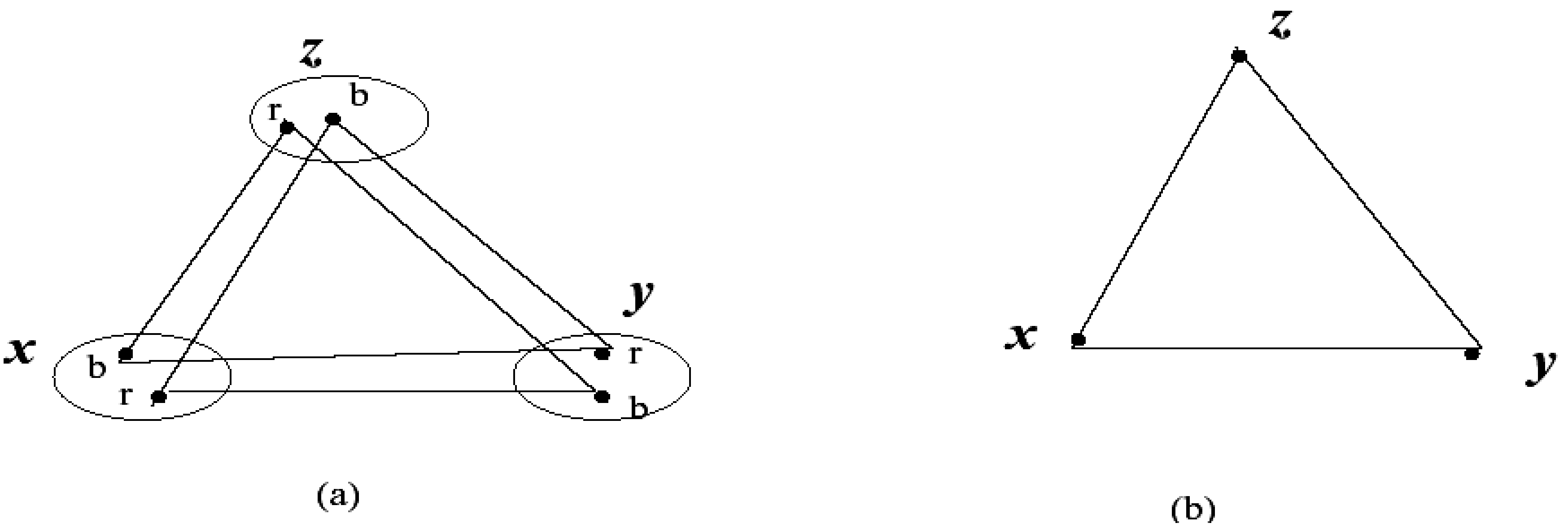


Figure 3.8: (a) The matching diagram of a 2-value graph coloring problem. (b) Graphical picture of path-consistency using the matching diagram.

Path-consistency (3-consistency)

Definition 3.3.2 (Path-consistency) *Given a constraint network* $\mathcal{R} = (X, D, C)$, *a*

Alternatively, a binary constraint R_{ij} *is path-consistent relative to* x_k *iff for every pair* $(a_i, a_j), \in R_{ij}$, *where* a_i *and* a_j *are from their respective domains, there is a value* $a_k \in D_k$ *s.t.* $(a_i, a_k) \in R_{ik}$ *and* $(a_k, a_j) \in R_{kj}$. *A subnetwork over three variables* $\{x_i, x_j, x_k\}$ *is path-consistent iff for any permutation of* (i, j, k), R_{ij} *is path consistent relative to* x_k. *A network is path-consistent iff for every* R_{ij} *(including universal binary relations) and for every* $k \neq i, j$ R_{ij} *is path-consistent relative to* x_k.

Revise-3

REVISE-3$((x, y), z)$

input: a three-variable subnetwork over (x, y, z), R_{xy}, R_{yz}, R_{xz}.
output: revised R_{xy} path-consistent with z.

1. **for** each pair $(a, b) \in R_{xy}$
2.   **if** no value $c \in D_z$ exists such that $(a, c) \in R_{xz}$ and $(b, c) \in R_{yz}$
3.   **then** delete (a, b) from R_{xy}.
4.   **endif**
5. **endfor**

Figure 3.9: Revise-3

$$R_{ij} \leftarrow R_{ij} \cap \pi_{ij}(R_{ik} \otimes D_k \otimes R_{kj})$$

- Complexity: $O(k^3)$
- Best-case: O(t)
- Worst-case O(tk)

PC-1

PC-1($\mathcal{R}$)

input: a network $\mathcal{R} = (X, D, C)$.
output: a path consistent network equivalent to $\mathcal{R}$.

1. **repeat**
2.   **for** $k \leftarrow 1$ to n
3.   **for** $i, j \leftarrow 1$ to n
4. $R_{ij} \leftarrow R_{ij} \cap \pi_{ij}(R_{ik} \bowtie D_k \bowtie R_{kj}) /^* \ (Revise - 3((i,j),k))$
5.   **endfor**
6.   **endfor**
7. **until** no constraint is changed.

Figure 3.10: Path-consistency-1 (PC-1)

- Complexity: $O(n^5 k^5)$
- O(n^3) triplets, each take O(k^3) steps → O($n^3 k^3$)
- Max number of loops: O(n^2 k^2) .

PC-2

PC-3($\mathcal{R}$)

input: a network $\mathcal{R} = (X, D, C)$.
output: $\mathcal{R}'$ a path consistent network equivalent to $\mathcal{R}$.

1. $Q \leftarrow \{(i,k,j) \mid 1 \le i < j \le n, 1 \le k \le n, k \ne i, k \ne j\ \}$
2. **while** Q is not empty
3. select and delete a 3-tuple (i,k,j) from Q
4. $R_{ij} \leftarrow R_{ij} \cap \pi_{ij}(R_{ik} \bowtie D_k \bowtie R_{kj})$ /* (Revise-3($(i,j),k$))
5. **if** R_{ij} changed then
6. $Q \leftarrow Q \cup \{(l,i,j)(l,j,i) \mid 1 \le l \le n, l \ne i, l \ne j\}$
7. **endwhile**

Figure 3.11: Path-consistency-3 (PC-3)

- Complexity: $O(n^3k^5)$
- Optimal PC-4: $O(n^3k^3)$
- (each pair deleted may add: 2n-1 triplets, number of pairs: O(n^2 k^2) → size of Q is O(n^3 k^2), processing is O(k^3))

Example: before and after path-consistency

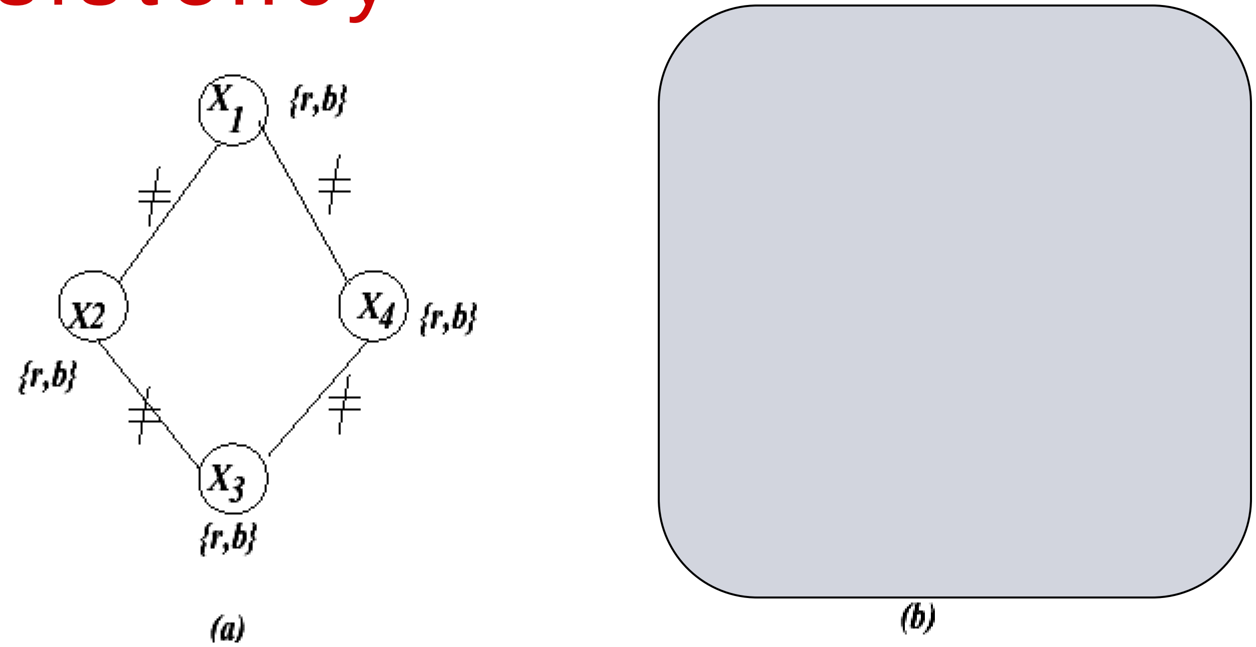


Figure 3.12: A graph-coloring graph (a) before path-consistency (b) after path-consistency

- PC-1 requires 2 processings of each arc while PC-2 may not
- Can we do path-consistency distributedly?

Example: before and after path-consistency

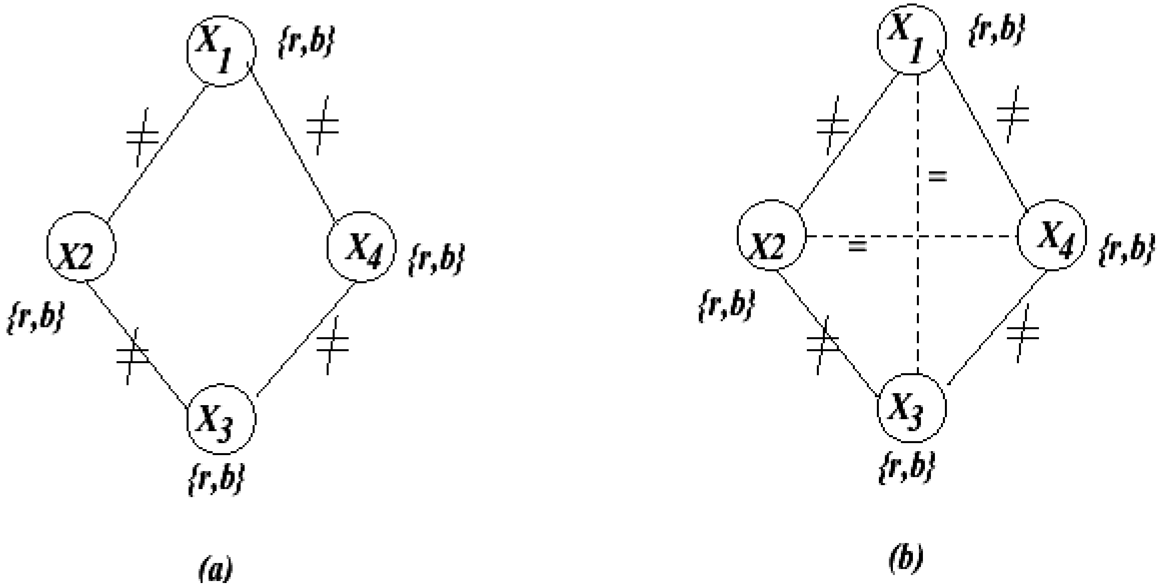


Figure 3.12: A graph-coloring graph (a) before path-consistency (b) after path-consistency

- PC-1 requires 2 processings of each arc while PC-2 may not
- Can we do path-consistency distributedly?

Path-consistency algorithms

- Apply Revise-3 $O(k^3)$ until no change

$$R_{ij} \leftarrow R_{ij} \cap \pi_{ij}(R_{ik} \otimes D_k \otimes R_{kj})$$

- Path-consistency (3-consistency) adds binary constraints.
- PC-1: $O(n^5k^5)$
- PC-2: $O(n^3k^5)$
- PC-4 optimal: $O(n^3k^3)$

I-consistency

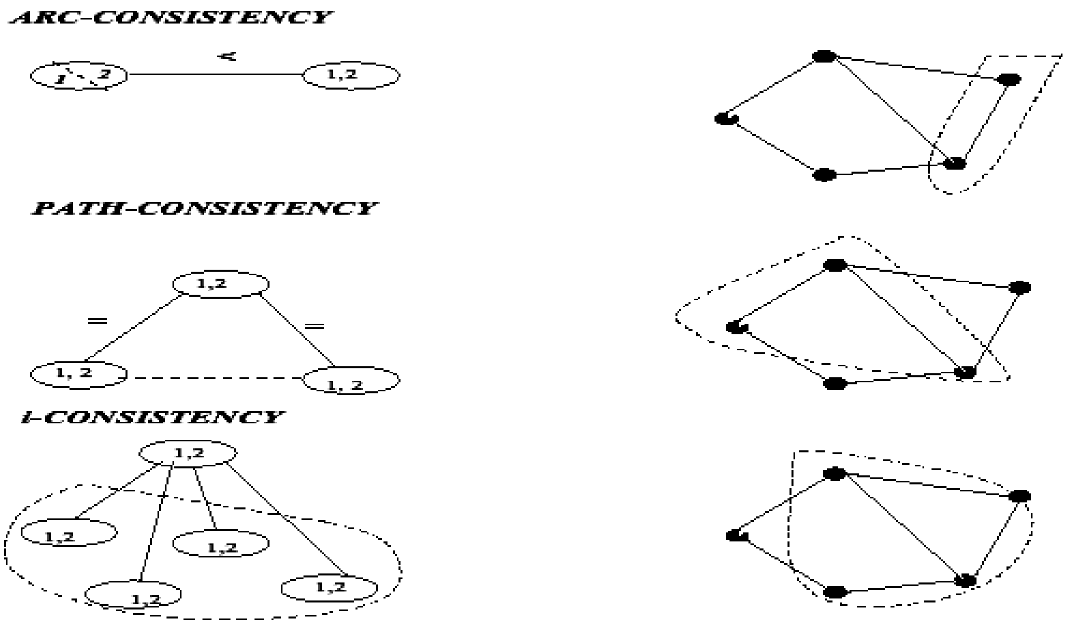


Figure 3.17: The scope of consistency enforcing: (a) arc-consistency, (b) path-consistency, (c) i-consistency

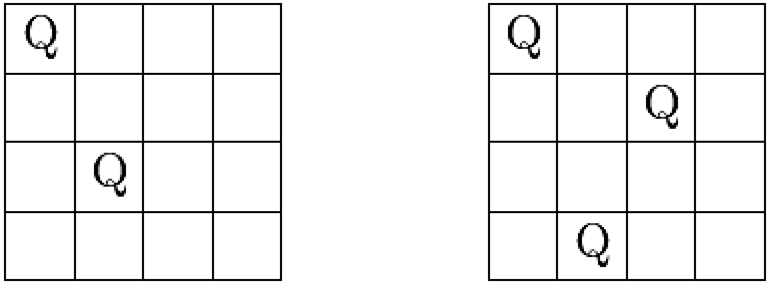
Higher levels of consistency, global-consistency

Definition:

A network is i-consistent *iff given any consistent instantiation of any* $i-1$ *distinct variables, there exists an instantiation of any ith variable such that the* i *values taken together satisfy all of the constraints among the* i *variables. A network is* strongly i-consistent *iff it is* j*-consistent for all* $j \leq i$*. A strongly* n*-consistent network, where* n *is the number of variables in the network, is called* globally consistent.

A Globally consistent network is backtrack-free

4-queen example



(a) (b)

Figure 3.13: (a) Not 3-consistent; (b) Not 4-consistent

Revise-i

REVISE-$i(\{x_1, x_2, ..., x_{i-1}\}, x_i)$

input: a network $\mathcal{R} = (X, D, C)$
output: a constraint R_S, $S = \{x_1, ..., x_{i-1}\}$ i-consistent relative to x_i.
1. **for** each instantiation $\bar{a}_{i-1} = (< x_1, a_1 >, < x_2, a_2 >, ..., < x_{i-1}, a_{i-1} >)$ do,
2. **if** no value of $a_i \in D_i$ exists s.t. $(\bar{a}_{i-1}, a_i)$ is consistent
 then delete $\bar{a}_{i-1}$ from R_S
 (Alternatively, let $\mathcal{S}$ be the set of all subsets of $\{x_1, ..., x_i\}$ that contain x_i
 and appear as scopes of constraints of $\mathcal{R}$, then
 $R_S \leftarrow R_S \cap \pi_S(\bowtie_{S' \subseteq \mathcal{S}} R_{S'}))$
3. **endfor**

Figure 3.14: Revise-i

- Complexity: for binary constraints $O(k^i)$
- For arbitrary constraints: $O((2k)^i)$

I-consistency

I-CONSISTENCY($\mathcal{R}$)

input: a network $\mathcal{R}$.
output: an i-consistent network equivalent to $\mathcal{R}$.

1. **repeat**
2. **for** every subset $S \subseteq X$ of size $i-1$, and for every x_i, do
3. let $\mathcal{S}$ be the set of all subsets in of $\{x_1, ..., x_i\}$ $scheme(\mathcal{R})$ that contain x_i
4. $R_S \leftarrow R_S \cap \pi_S(\bowtie_{S' \in \mathcal{S}} R_{S'})$ (this is Revise-i(S, x_i))
6. **endfor**
7. **until** no constraint is changed.

Figure 3.15: i-consistency-1

Theorem 3.4.3 (complexity of i-consistency) *The time and space complexity of brute-force i-consistency $O(2^i(nk)^{2i})$ and $O(n^i k^i)$, respectively. A lower bound for enforcing i-consistency is $\Omega(n^i k^i)$.* □

I-consistency

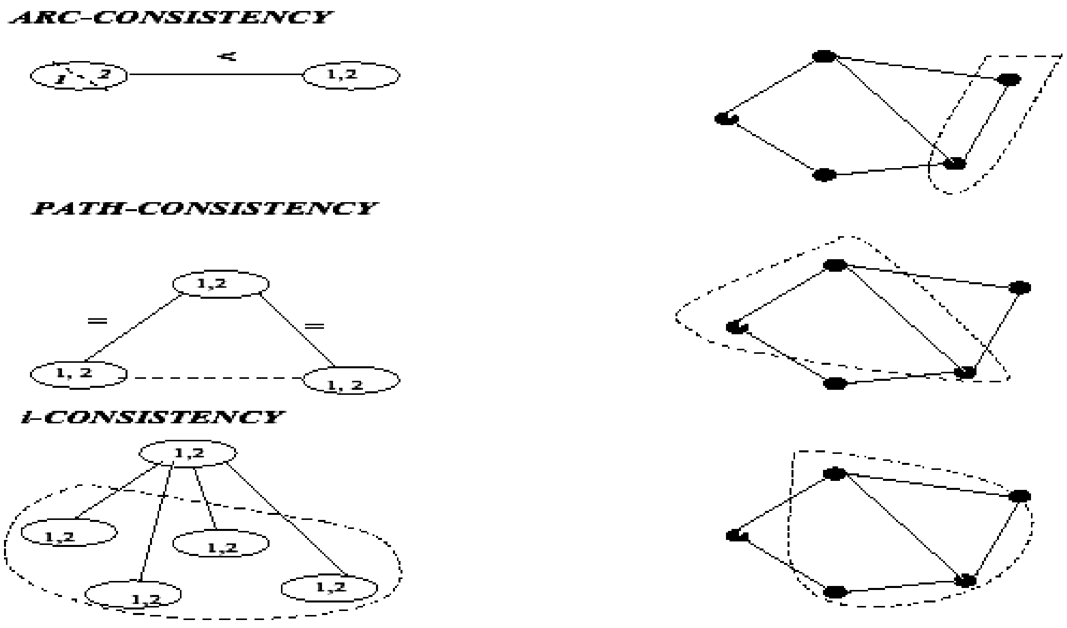


Figure 3.17: The scope of consistency enforcing: (a) arc-consistency, (b) path-consistency, (c) i-consistency

Outline

- Arc-consistency algorithms
- Path-consistency and i-consistency
- Arc-consistency, Generalized arc-consistency, relation arc-consistency
- Global and bound consistency
- Distributed (generalized) arc-consistency
- Consistency operators: join, resolution, Gausian elimination

Arc-consistency for non-binary constraints:

Generalized arc-consistency

Definition 3.5.1 (generalized arc-consistency) *Given a constraint network $\mathcal{R} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$, with $R_S \in C$, a variable x is* arc-consistent *relative to R_S if and only if for every value $a \in D_x$ there exists a tuple $t \in R_S$ such that $t[x] = a$. t can be called a support for a. The constraint R_S is called arc-consistent iff it is arc-consistent relative to each of the variables in its scope and a constraint network is arc-consistent if all its constraints are arc-consistent.*

$$D_x \leftarrow D_x \cap \pi_x(R_S \otimes D_{S-\{x\}})$$

Complexity: O(t k), t bounds number of tuples.

Relational arc-consistency:

$$R_{S-\{x\}} \leftarrow \pi_{S-\{x\}}(R_S \otimes D_x)$$

Algorithm 1: AC3 / GAC3

function `Revise3`(**in** x_i: variable; c: constraint)**: Boolean** ;

begin

  CHANGE $\leftarrow$ **false**;

  foreach $v_i \in D(x_i)$ **do**

    if $\nexists \tau \in c \cap \pi_{X(c)}(D)$ *with* $\tau[x_i] = v_i$ **then**

      remove v_i from $D(x_i)$;

      CHANGE $\leftarrow$ **true**;

  return CHANGE ;

end

function `AC3/GAC3`(**in** X: set)**: Boolean** ;

begin

  /* initalisation */;

  $Q \leftarrow \{(x_i, c) \mid c \in C, x_i \in X(c)\}$;

  /* propagation */;

  while $Q \neq \emptyset$ **do**

    select and remove (x_i, c) from Q;

    if *Revise*(x_i, c) **then**

      if $D(x_i) = \emptyset$ **then** return **false** ;

      else $Q \leftarrow Q \cup \{(x_j, c') \mid c' \in C \wedge c' \neq c \wedge x_i, x_j \in X(c') \wedge j \neq i\}$;

  return **true** ;

end

Generalized arc-consistency

Proposition 27 (GAC3). *GAC3 is a sound and complete algorithm for achieving arc consistency that runs in $O(er^3d^{r+1})$ time and $O(er)$ space, where r is the greatest arity among constraints.*

Examples of generalized arc-consistency

- $x+y+z \leq 15$ and $z \geq 13$ implies

  $x \leq 2$, $y \leq 2$

- Example of relational arc-consistency

$$A \wedge B \rightarrow G,$$

$$\neg G, \Rightarrow$$

$$\neg A \vee \neg B$$

More arc-based consistency

- Global constraints: e.g., all-different constraints
  - Special semantic constraints that appears often in practice and a specialized constraint propagation. Used in constraint programming.
- Bounds-consistency: pruning the boundaries of domains

Sudoku –
Constraint Satisfaction

- **Constraint**
- **Propagation**

- **Inference**

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

- **Variables: empty slots**
- **Domains = {1,2,3,4,5,6,7,8,9}**
- **Constraints:**
  - **27 all-different**

Each row, column and major block must be alldifferent

“Well posed” if it has unique solution: 27 constraints

Outline

- Arc-consistency algorithms
- Path-consistency and i-consistency
- Arc-consistency, Generalized arc-consistency, relation arc-consistency
- Global and bound consistency
- Distributed (generalized) arc-consistency
- Consistency operators: join, resolution, Gausian elimination

Global constraints

Constraints of arbitrary scope length defined by expression, a Boolean function

Global constraints are classes of constraints defined by a formula of arbitrary arity (see Section 9.2).

Example 2. The constraint `alldifferent`$(x_1, x_2, x_3) \equiv (v_i \neq v_j \wedge v_i \neq v_k \wedge v_j \neq v_k)$ allows the infinite set of 3-tuples in $\mathbb{Z}^3$ such that all values are different. The constraint $c(x_1, x_2, x_3) = \{(2,2,3), (2,3,2), (2,3,3), (3,2,2), (3,2,3), (3,3,2)\}$ allows the finite set of 3-tuples containing both values 2 and 3 and only them.

Gloabal Constraints (continued)

Example 86. The `alldifferent`$(x_1, \ldots, x_n)$ global constraint is the class of constraints that are defined on any sequence of n variables, $n \geq 2$, such that $x_i \neq x_j$ for all $i, j, 1 \leq i, j \leq n, i \neq j$. The `NValue`$(y, [x_1, \ldots, x_n])$ global constraint is the class of constraints that are defined on any sequence of $n+1$ variables, $n \geq 1$, such that $|\{x_i \mid 1 \leq i \leq n\}| = y$ [100, 8].

We need specialized procedures for generalize Arc-consistency because it is too expensive to try and apply the general algorithm (see Bessiere, section 9.2)

We can decompose a global constraint, or use various specialized representation

Example for alldiff

- A = {3,4,5,6}
- B = {3,4}
- C= {2,3,4,5}
- D= {2,3,4}
- E = {3,4}
- F= {1,2,3,4,5,6}
- Alldiff (A,B,C,D,E}
- Arc-consistency does nothing
- Apply GAC to sol(A,B,C,D,E,F)?
- → A = {6}, F = {1}….
- Alg: bipartite matching kn^1.5
- (Lopez-Ortiz, et. Al, IJCAI-03 pp 245 (A fast and simple algorithm for bounds consistency of alldifferent constraint)

Global constraints

- Alldifferent
- Sum constraint (variable equal the sum of others)
- Global cardinality constraint (a value can be assigned a bounded number of times to a set of variables)
- The cummulative constraint (related to scheduling tasks)

Bounds consistency

Definition 3.5.4 (bounds consistency) *Given a constraint C over a scope S and domain constraints, a variable $x \in S$ is bounds-consistent relative to C if the value $min\{D_x\}$ (respectively, $max\{D_x\}$) can be extended to a full tuple t of C. We say that t supports $min\{D_x\}$. A constraint C is* bounds-consistent *if each of its variables is bounds-consistent.*

Bounds consistency

Example 3.5.5 Consider the constraint problem with variables $x_1, ...x_6$, each with domains $1, ..., 6$, and constraints:

$$C_1 : x_4 \geq x_1 + 3, \quad C_2 : x_4 \geq x_2 + 3, \quad C_3 : x_5 \geq x_3 + 3, \quad C_4 : x_5 \geq x_4 + 1,$$

$$C_5 : alldifferent\{x_1, x_2, x_3, x_4, x_5\}$$

The constraints are not bounds consistent. For example, the minimum value 1 in the domain of x_4 does not have support in constraint C_1 as there is no corresponding value for x_1 that satisfies the constraint. Enforcing bounds consistency using constraints C_1 through C_4 reduces the domains of the variables as follows: $D_1 = \{1, 2\}$, $D_2 = \{1, 2\}$, $D_3 = \{1, 2, 3\}$ $D_4 = \{4, 5\}$ and $D_5 = \{5, 6\}$. Subsequently, enforcing bounds consistency using constraints C_5 further reduces the domain of C to $D_3 = \{3\}$.Now constraint C_3 is no longer bound consistent. Reestablishing bounds consistency causes the domain of x_5 to be reduced to $\{6\}$. Is the resulting problem already arc-consistent? □

Outline

- Arc-consistency algorithms
- Path-consistency and i-consistency
- Arc-consistency, Generalized arc-consistency, relation arc-consistency
- Global and bound consistency
- Distributed (generalized) arc-consistency
- Consistency operators: join, resolution, Gausian elimination

Boolean constraint propagation

- (A V ¬B) and (B)
  - B is arc-consistent relative to A but not vice-versa
- Arc-consistency by resolution:

  res((A V ¬B),B) = A

Given also (B V C), path-consistency:

res((A V ¬B),(B V C) = (A V C)

Relational arc-consistency rule = unit-resolution

$$A \wedge B \rightarrow G, \neg G, \Rightarrow \neg A \vee \neg B$$

Constraint propagation for Boolean constraints: Unit propagation

Procedure UNIT-PROPAGATION
Input: A cnf theory, φ, $d = Q_1, ..., Q_n$.
Output: An equivalent theory such that every unit clause does not appear in any non-unit clause.

1. queue = all unit clauses.
2. **while** queue is not empty, do.
3. $T \leftarrow$ next unit clause from Queue.
4. **for** every clause β containing T or $\neg T$
5. **if** β contains T delete β (subsumption elimination)
6. **else**, For each clause $\gamma = \mathit{resolve}(\beta, T)$.
   if γ, the resolvent, is empty, the theory is unsatisfiable.
7. **else**, add the resolvent γ to the theory and delete β.
   if γ is a unit clause, add to Queue.
8. **endfor**.
9. **endwhile**.

Theorem 3.6.1 *Algorithm* UNIT-PROPAGATION *has a linear time complexity.*

Consistency for numeric constraints (Gausian elimination)

$x \in [1,10], y \in [5,15],$

$x + y = 10$

$arc - consistency \Rightarrow x \in [1,5], y \in [5,9]$

Gausian elimination of $x + y = 10, -y \leq -5$

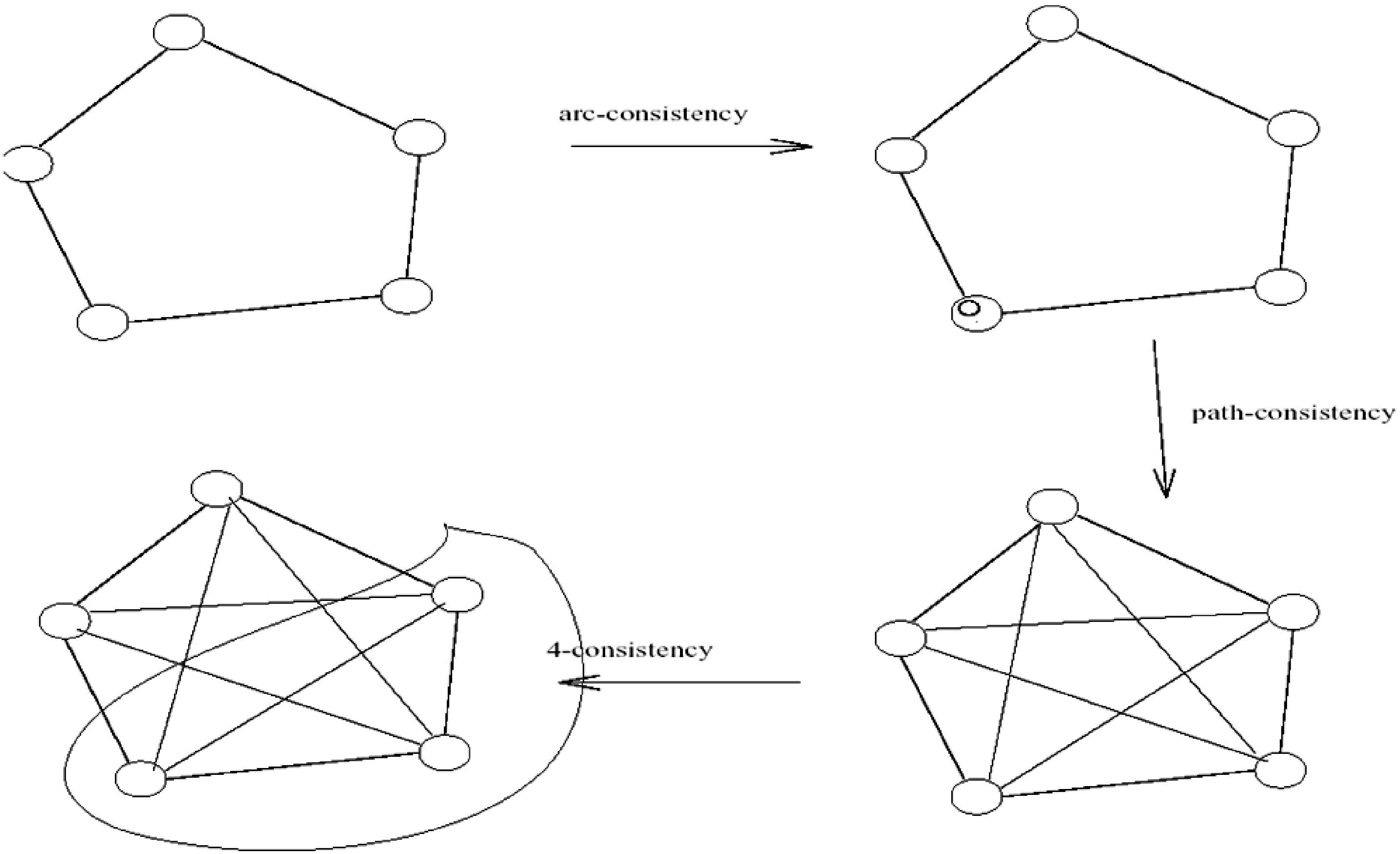
$z \in [-10,10],$

$y + z \leq 3$

$path - consistency \Rightarrow x - z \geq 7$

Gausian Elinination of: $x + y = 10, -y - z \geq -3$

Changes in the network graph as a result of arc-consistency, path-consistency and 4-consistency.



Outline

- Arc-consistency algorithms
- Path-consistency and i-consistency
- Arc-consistency, Generalized arc-consistency, relation arc-consistency
- Global and bound consistency
- Distributed (generalized) arc-consistency
- Consistency operators: join, resolution, Gausian elimination

Distributed arc-consistency (Constraint propagation)

- Implement AC-1 distributedly.

$$D_i \leftarrow D_i \cap \pi_i(R_{ij} \otimes D_j)$$

- Node x_j sends the message to node x_i

$$h_i^j \leftarrow \pi_i(R_{ij} \otimes D_j)$$

- Node x_i updates its domain:

$$D_i \leftarrow D_i \cap h_i^j$$

- Relational and generalized arc-consistency can be implemented distributedly: sending messages between constraints over the dual graph

$$R_{S-\{x\}} \leftarrow \pi_{S-\{x\}}(R_S \otimes D_x)$$

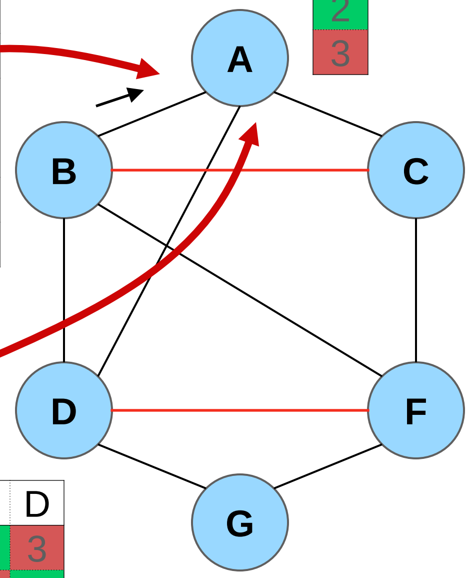
Relational Arc-consistency

The message that R2 sends to R1 is

$$h_i^j \leftarrow \pi_{l_{ij}}(R_i \bowtie (\bowtie_{k \in ne(i)} h_k^i))$$

R1 updates its relation and domains and sends messages to neighbors

$$D_i \leftarrow D_i \cap (\bowtie_{k \in ne(i)} D_k^i)$$



R_1

A
1
2
3

R_2

A	B
1	2
1	3
2	1
2	3
3	1
3	2

R_3

A	C
1	2
3	2

R_4

A	B	D
1	2	3
1	3	2
2	1	3
2	3	1
3	1	2
3	2	1

R_5

B	C	F
1	2	3
3	2	1

R_6

D	F	G
1	2	3
2	1	3

Distributed Relational Arc-Consistency

- DRAC can be applied to the dual problem of any constraint network:

$$h_i^j \leftarrow \pi_{l_{ij}}(R_i \bowtie (\bowtie_{k \in ne(i)} h_k^i)) \qquad (1)$$

$$R_i \leftarrow R_i \cap (\bowtie_{k \in ne(i)} h_k^i) \qquad (2)$$

DRAC on the dual join-graph

R_1

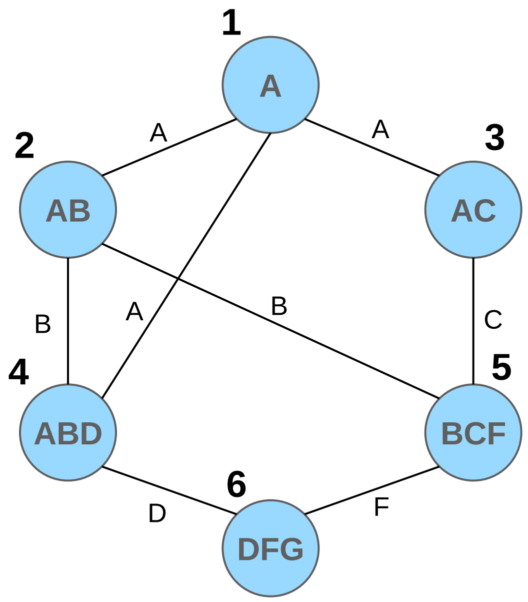
A
1
2
3

R_2

A	B
1	2
1	3
2	1
2	3
3	1
3	2

R_3

A	C
1	2
3	2



R_4

A	B	D
1	2	3
1	3	2
2	1	3
2	3	1
3	1	2
3	2	1

R_5

B	C	F
1	2	3
3	2	1

R_6

D	F	G
1	2	3
2	1	3

Iteration 1

R_1

A
1
2
3

h_2^1

A
1
2
3

h_3^1

A
1
3

h_4^1

A
1
2
3

h_5^2

B
1
3

h_4^2

A	B
1	2
1	3
2	1
2	3
3	1
3	2

h_3^2

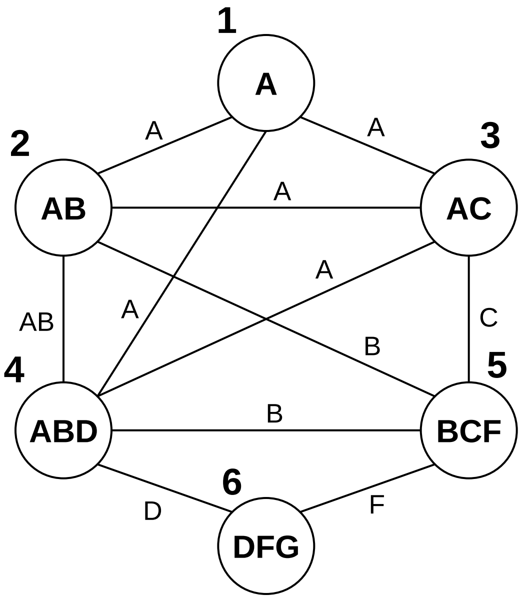
A
1
3

h_1^2

A
1
2
3

R_2

A	B
1	2
1	3
2	1
2	3
3	1
3	2



R_3

A	C
1	2
3	2

h_1^3

A
1
2
3

h_2^3

A
1
2
3

h_4^3

A
1
2
3

h_5^3

C
2

h_6^4

D
1
2

h_5^4

B
1
3

h_3^4

A
1
3

h_2^4

A	B
1	2
1	3
2	1
2	3
3	1
3	2

h_1^4

A
1
2
3

R_4

A	B	D
1	2	3
1	3	2
2	1	3
2	3	1
3	1	2
3	2	1

R_5

B	C	F
1	2	3
3	2	1

h_2^5

B
1
2
3

h_3^5

C
2

h_4^5

B
1
2
3

h_6^5

F
1
3

R_6

D	F	G
1	2	3
2	1	3

h_4^6

D
1
2
3

h_5^6

F
1
3

$$R_i \leftarrow R_i \cap \left(\bowtie_{k \in ne(i)} h^i_k\right) \qquad (2)$$

Iteration 1

R_1

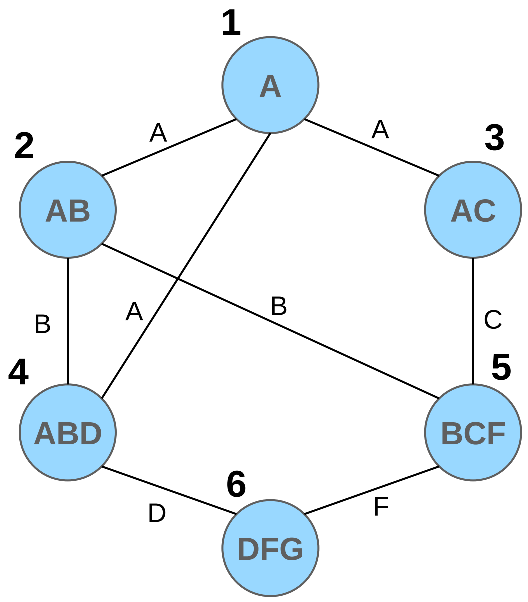
A
1
3

R_2

A	B
1	3
2	1
2	3
3	1

R_3

A	C
1	2
3	2



R_4

A	B	D
1	3	2
2	3	1
3	1	2
3	2	1

R_5

B	C	F
1	2	3
3	2	1

R_6

D	F	G
2	1	3

$$h_i^j \leftarrow \pi_{l_{ij}}(R_i \bowtie (\bowtie_{k \in ne(i)} h_k^i)) \quad (1)$$

Iteration 2

R_1

A
1
3

h_2^1

A
1
2
3

h_3^1

A
1
3

h_4^1

A
1
2
3

h_5^2

B
1
3

h_4^2

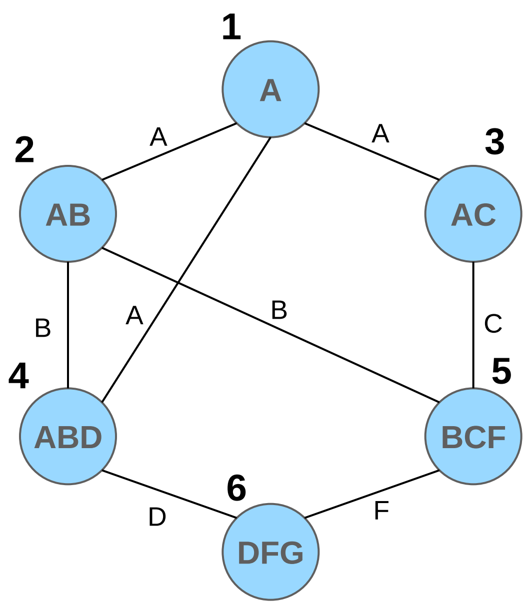
B
1
2
3

h_1^2

A
1
3

R_2

A	B
1	3
2	1
2	3
3	1



R_3

A	C
1	2
3	2

h_1^3

A
1
3

h_5^3

C
2

h_6^4

D
2

h_2^4

B
1
3

h_1^4

A
1
3

R_4

A	B	D
1	3	2
2	3	1
3	1	2
3	2	1

R_5

B	C	F
1	2	3
3	2	1

h_2^5

B
1
3

h_3^5

C
2

h_6^5

F
1

R_6

D	F	G
2	1	3

h_4^6

D
1
2

h_5^6

F
1
3

$$R_i \leftarrow R_i \cap (\bowtie_{k \in ne(i)} h_k^i) \qquad (2)$$

Iteration 2

R_1

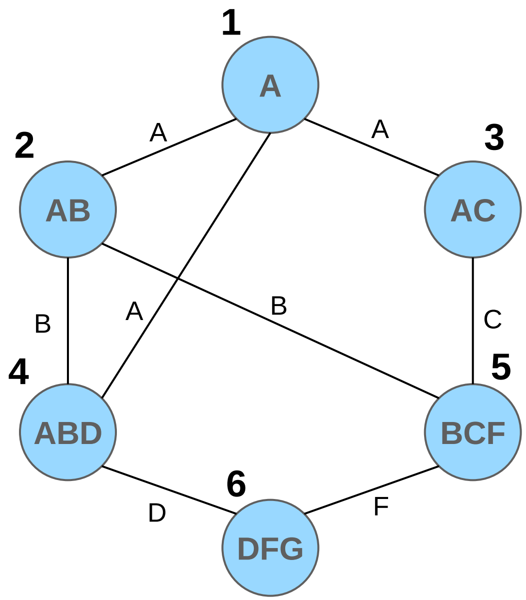
A
1
3

R_2

A	B
1	3
3	1

R_3

A	C
1	2
3	2



R_4

A	B	D
1	3	2
3	1	2

R_5

B	C	F
3	2	1

R_6

D	F	G
2	1	3

$$h_i^j \leftarrow \pi_{l_{ij}}(R_i \bowtie (\bowtie_{k \in ne(i)} h_k^i)) \qquad (1)$$

Iteration 3

R_1

A
1
3

h_2^1

A
1
3

h_3^1

A
1
3

h_4^1

A
1
3

h_5^2

B
3

h_4^2

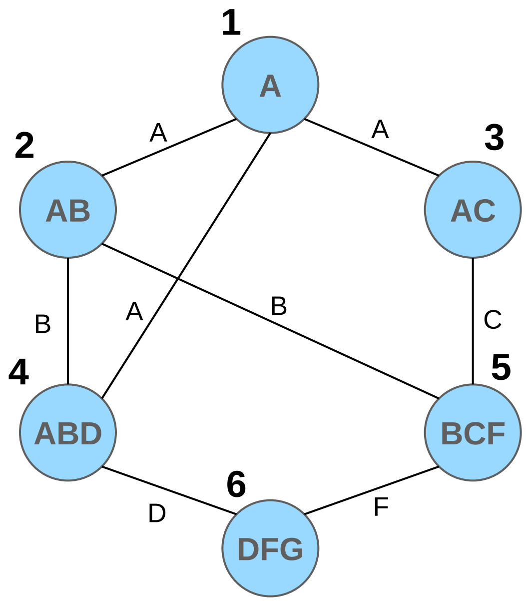
B
1
3

h_1^2

A
1
3

R_2

A	B
1	3
3	1



R_3

A	C
1	2
3	2

h_1^3

A
1
3

h_5^3

C
2

h_6^4

D
2

h_2^4

B
1
3

h_1^4

A
1
3

R_4

A	B	D
1	3	2
3	1	2

R_5

B	C	F
3	2	1

h_2^5

B
1
3

h_3^5

C
2

h_6^5

F
1

R_6

D	F	G
2	1	3

h_4^6

D
2

h_5^6

F
1

$$R_i \leftarrow R_i \cap (\bowtie_{k \in ne(i)} h_k^i) \qquad (2)$$

Iteration 3

R_1

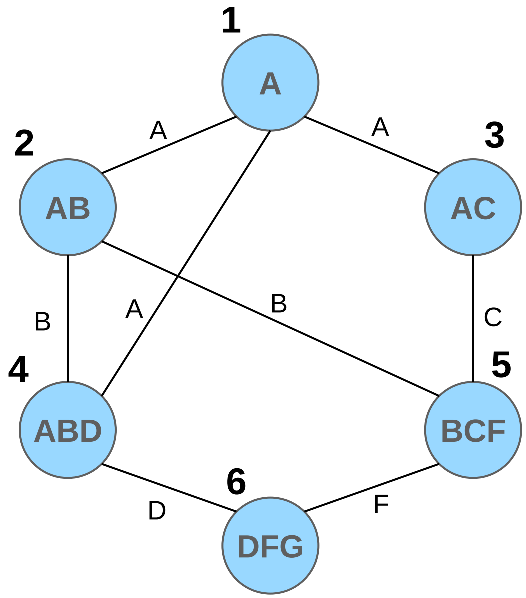
A
1
3

R_2

A	B
1	3

R_3

A	C
1	2
3	2



R_4

A	B	D
1	3	2
3	1	2

R_5

B	C	F
3	2	1

R_6

D	F	G
2	1	3

$$h_i^j \leftarrow \pi_{l_{ij}}(R_i \bowtie (\bowtie_{k \in ne(i)} h_k^i)) \quad (1)$$

Iteration 4

R_1

A
1
3

h_2^1

A
1

h_3^1

A
1
3

h_4^1

A
1
3

h_5^2

B
3

h_4^2

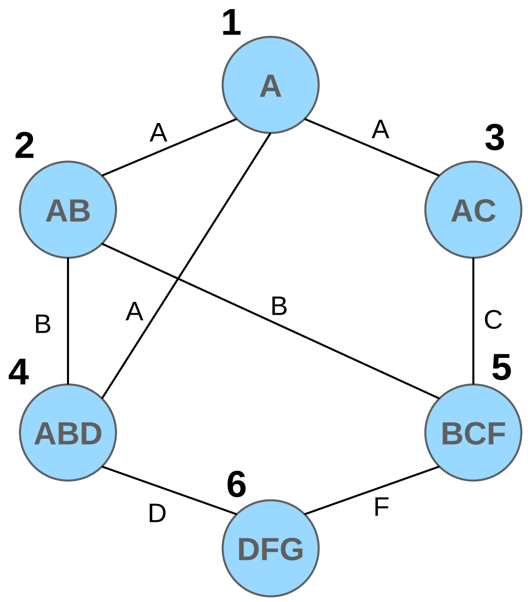
B
1
3

h_1^2

A
1
3

R_2

A	B
1	3



R_3

A	C
1	2
3	2

h_1^3

A
1
3

h_5^3

C
2

h_6^4

D
2

h_2^4

B
1
3

h_1^4

A
1
3

R_4

A	B	D
1	3	2
3	1	2

R_5

B	C	F
3	2	1

h_2^5

B
3

h_3^5

C
2

h_6^5

F
1

R_6

D	F	G
2	1	3

h_4^6

D
2

h_5^6

F
1

$$R_i \leftarrow R_i \cap (\bowtie_{k \in ne(i)} h_k^i) \qquad (2)$$

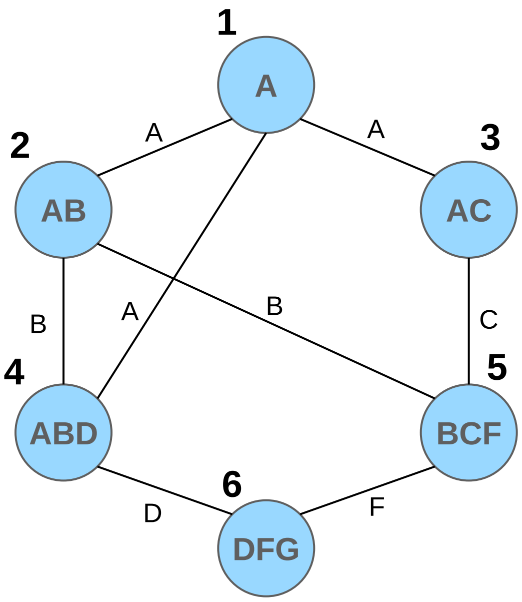
Iteration 4

R_1

A
1

R_2

A	B
1	3



R_3

A	C
1	2
3	2

R_4

A	B	D
1	3	2

R_5

B	C	F
3	2	1

R_6

D	F	G
2	1	3

$$h_i^j \leftarrow \pi_{l_{ij}}(R_i \bowtie (\bowtie_{k \in ne(i)} h_k^i)) \qquad (1)$$

Iteration 5

R_1

A
1

h_2^1

A
1

h_3^1

A
1

h_4^1

A
1

h_5^2

B
3

h_4^2

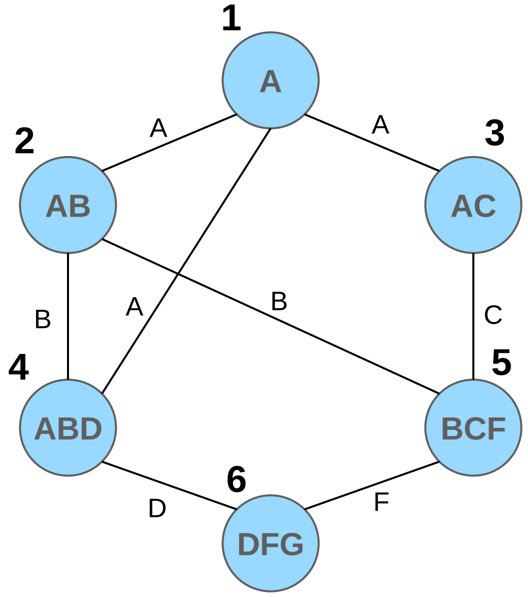
B
3

h_1^2

A
1

R_2

A	B
1	3



R_3

A	C
1	2
3	2

h_1^3

A
1

h_5^3

C
2

h_6^4

D
2

h_2^4

B
3

h_1^4

A
1

R_4

A	B	D
1	3	2

R_5

B	C	F
3	2	1

h_2^5

B
3

h_3^5

C
2

h_6^5

F
1

R_6

D	F	G
2	1	3

h_4^6

D
2

h_5^6

F
1

$$R_i \leftarrow R_i \cap (\bowtie_{k \in ne(i)} h_k^i) \quad (2)$$

Iteration 5

R_1

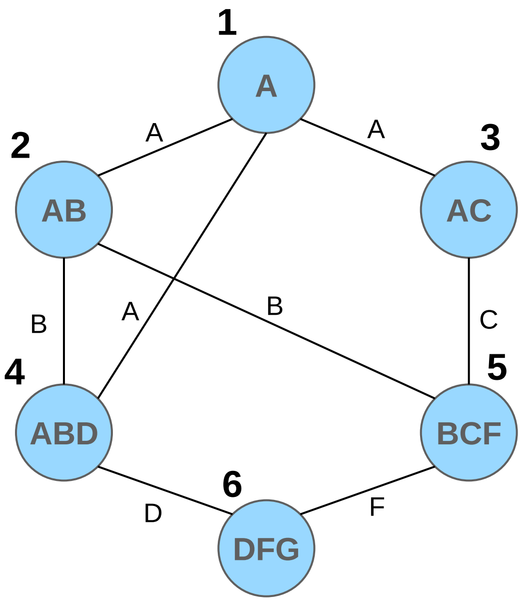
A
1

R_2

A	B
1	3

R_3

A	C
1	2



R_4

A	B	D
1	3	2

R_5

B	C	F
3	2	1

R_6

D	F	G
2	1	3

Tractable classes

Theorem 3.7.1 *1. The consistency binary constraint networks having no cycles can be decided by arc-consistent*

2. The consistency of binary constraint networks with bi-valued domains can be decided by path-consistency,

3. The consistency of Horn cnf theories can be decided by unit propagation.