Title: Consistency Algorithms and operations on constraints

Material: Davis AIJ'87, Mackworth AIJ'77,
Mackworth and Freuder AIJ'85,
Chapter 1 of Dechter's manuscript

Foundations of Constraint Satisfaction
CSCE 990-005, Fall 2000

Berthe Y. Choueiry (Shu-we-ri)
Ferguson Hall, Room 104
choueiry@cse.unl.edu, Tel: (402)472-5444

Outline

- Motivation and material
- Davis: Waltz algorithm, quick discussion
- Node and arc consistency and their complexity
- *Constraints as relation, operations on relations*
- Path consistency and its complexity
- Some recent developments

Consistency checking

Motivation

CSP are solved with search (conditioning)

Search performance is affected by:

- problem size
- amount of backtracking

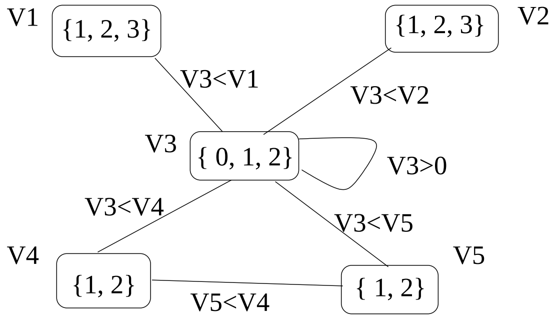
Backtracking may yield **thrashing**

Thrashing

exploring non-promising subtrees
and rediscovering the same inconsistent
combinations over and over again

$\longrightarrow$ Malady of backtrack search

(Some) causes of thrashing



Search order: V1, V2, V3, V4, V5

What happens in search if we:

- do not check the constraint on V3 (node inconsistency)
- do not check constraint between V3 and V5 (arc inconsistency)
- do not check constraints between V3–V4, V3–V5 and V4–V5 (path inconsistency)

Consistency checking

Goal

eliminates inconsistent combinations prior to, or during, search

Minimal network *a.k.a. central problem*
Given two values for two variables, if they are consistent, then they appear in at least one solution.

Algorithms

Binary constraints:

- Node consistency
- Arc consistency (AC-1, AC-2, etc.)
- Path consistency (PC-1, PC-2, etc.)

Constraints of arbitrary arity:

- Waltz algorithm ancestor of AC's

Material

- Davis, AIJ'77: concentrates on the Waltz algorithm and studies its performance and quiescence depending on:
  - constraint types (order relations, bounded diff, algebraic, etc.)
  - domains types (continuous or finite)
- Mackworth, AIJ'77: presents NC, AC-1, AC-2, PC-1, PC-2
- Mackworth and Freuder, AIJ'85: studies their complexity
- Section 1.2 of Dechter's manuscript: operations on relations

→ Mackworth and Freuder concentrate on finite domains
→ More people work on finite domains than continuous ones
→ Continuous domains can be quite tough

Constraint Propagation with Interval Labels

Ernest Davis

- 'Old' paper (1987): terminology slightly different
- Interval labels: continuous domains
  Section 8: sign labels (discrete)
- Concerned with Waltz algorithm
  (e.g., quiescence, completeness)
  Constraint types vs. domain types (Table 3)
- Addresses applications from reasoning
  about Physical Systems (circuit, `SPAM`)
  about time relations (`TMM`)

⟶ Please read Section 3 (at least, more if you wish)

Terminology (I)

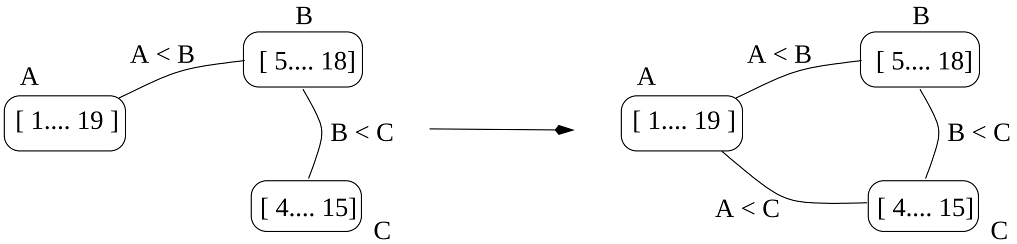
Effective: An effective procedure is a procedure with finitely-long and exact instructions that avoids random devices, and answers in a finite number of steps. [Enderton 72]

Efficient: An efficient procedure is a procedure that returns an answer in a time **polynomial** in the size of the input.

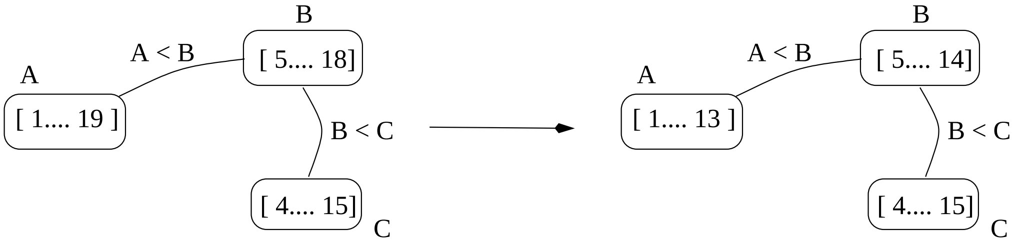
$\surd$ **Constraint inference**
$\surd$ **Label inference**
$\surd$ **Value inference**

$\times$ **Expression inference**
$\times$ **Relaxation**
$\times$ **Relaxation labeling**

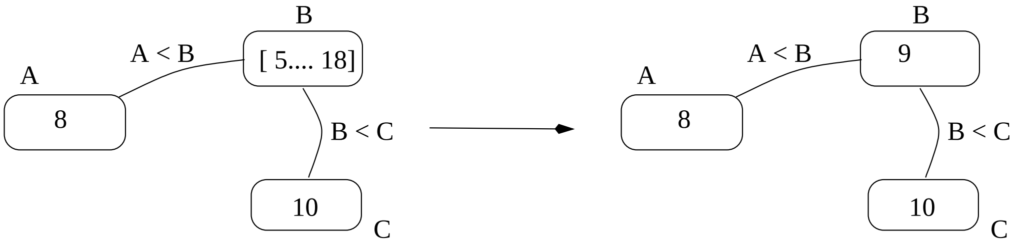
Constraint inference



Label inference

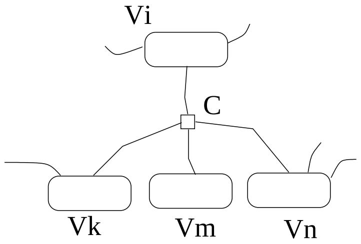


Value inference (dataflow constraint)



Waltz Algo. for label inference

`Refine`$(C(V_i, V_k, V_m, V_n),\ V_i)$ finds a new label for V_i consistent with C



`Revise`$(C(V_i, V_k, V_m, V_n))$ **refines** the domains of V_i, V_k, V_m, V_n by iterating over these variables until quiescence (no domains is further refined)

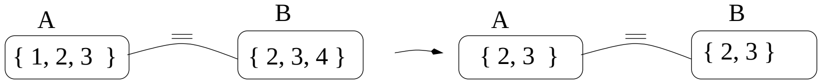
`Waltz Algorithm` **revises** all constraints by iterating over each constraint connected to a variable whose domain has been revised

`Waltz Algorithm` $\rightarrow$ ancestor of arc-consistency

WARNING

$\longrightarrow$ Completeness:

Running the `Waltz Algorithm` does not solve the problem



A=2 and B=3 is still not a solution!

$\longrightarrow$ Quiescence:

The Waltz algorithm may go into infinite loops even if problems is solvable

$$x \in [0, 100] \quad x = y$$
$$y \in [0, 100] \quad x = 2y$$

$\longrightarrow$ Davis provides a classification of problems f(constraint types, domain types) in terms of completeness and quiescence of the Waltz algorithm. (Table 3)

Importance of this paper

$\longrightarrow$ Establishes that constraints of bounded differences (temporal reasoning) can be efficiently solved by the Waltz algorithm ($O(n^3)$, n number of variables)

$\longrightarrow$ Early paper that attracts attention on the difficulty of label propagation in interval labels (continuous domains).

This work has been continued by Faltings (AIJ 92) who studied the early-quiescence problem of the Waltz algorithm.

Consistency checking algorithms in finite, binary CSPs and their complexity

Mackworth AIJ'77 and Mackwork and Freuder AIJ'87

Notation: Given variables i, j, k with values x, y, z, the predicate $P_{ijk}(x, y, z)$ is true iff the 3-tuple $\langle x, y, z \rangle \in C_{ijk}$

Node consistency: checking $P_i(x)$

Arc consistency: checking $P_{ij}(x, y)$

Path consistency: binary matrix manipulations

Worst-case complexity $\left\{ \begin{array}{c} \text{time} \\ \text{space} \end{array} \right\}$ as a function of the input parameters

$f(n)$ is $\mathcal{O}(g(n))$ means that $f(n) \leq c.g(n)$
f grows as g or slower

$f(n)$ is $\Omega(h(n))$ means that $f(n) \geq c.h(n)$

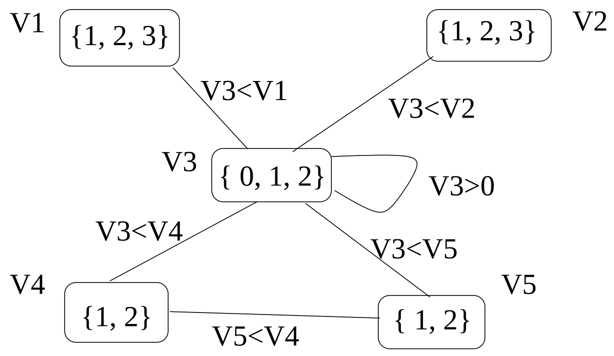
[Papadimitriou 94, Sect 1.1]

Input parameters for a CSP:

$a =$	(max) size of a domain
$d_k =$	degree of V_k ($\leq n-1$)
$n =$	number of variables
$e =$	number of edges (or constraints) $\in [(n-1), n(n-1)/2]$

Node consistency (NC)

Procedure NC(i):
$D_i \leftarrow D_i \cap \{x | P_i(x)\}$
Begin
  for $i \leftarrow 1$ until n do NC(i)
end



Complexity of NC

Procedure NC(i):
$D_i \leftarrow D_i \cap \{x|P_i(x)\}$
Begin
for $i \leftarrow 1$ until n do NC(i)
end

For each variable, we check a values

We have n variables, we do $n.a$ checks

NC is $\mathcal{O}(a.n)$

Procedure REVISE

Revises the domains of a variable i

Procedure REVISE(i,j):
Begin
  DELETE $\leftarrow$ false
  for each $x \in D_i$ **do**
    if there is no $y \in D_j$ such that $P_{ij}(x, y)$ **then**
      begin
        delete x from D_i;
        DELETE $\leftarrow$ true
      end
  return DELETE
end

REVISE is directional

Complexity of REVISE? a^2

Arc consistency (AC-1)

Procedure AC-1:

Begin
    for $i \leftarrow 1$ **until** n **do** NC(i)
    $Q \leftarrow \{(i,j) | \ (i,j) \in \text{arcs}(G), i \neq j$
    repeat
        begin
        CHANGE $\leftarrow$ false
        for each $(i,j) \in Q$ **do** CHANGE $\leftarrow$ (REVISE(i, j) **or** CHANCE)
        end
    until $\neg$ CHANGE
end

AC-1 does not update Q, the queue of arcs

Complexity of AC-1

```
Procedure AC-1:
1 Begin
2     for i ← 1 until n do NC(i)
3     Q ← {(i, j)| (i, j) ∈ arcs(G), i ≠ j
4     repeat
5         begin
6         CHANGE ← false
7         for each (i, j) ∈ Q do CHANGE ← (REVISE(i, j) or CHANCE)
8         end
9     until ¬ CHANGE
10 end
```

Q is not modified, and $|Q| = 2e$

4—9 repeats at most $n.a$ times

Each iteration has $|Q| = 2e$ calls to `REVISE`

`REVISE` requires at most a^2 checks of P_{ij}

$\longrightarrow$ AC-1 is $\mathcal{O}(a^3.n.e)$

AC-1 does not update Q, the queue of arcs

AC-2 iterates over arcs connected to at least one node whose domains has been modified. Nodes are ordered.

AC-3 same as AC-2, nodes are not ordered

Arc consistency (AC-3)

AC-3 iterates over arcs connected to at least one node whose domains has been modified

Procedure AC-3:

Begin
  for $i \leftarrow 1$ **until** n **do** NC(i)
  $Q \leftarrow \{(i,j) \mid (i,j) \in \text{arcs}(G), i \neq j$
  While Q is not empty **do**
    begin
    select and delete any arc (k,m) from Q
    if REVISE(k,m) **then** $Q \leftarrow Q \cup \{(i,k) \mid (i,k) \in \text{arcs}(G), i \neq k, i \neq m\}$
    end
end

Complexity of AC-3

Procedure AC-3:
1 **Begin**
2 **for** $i \leftarrow 1$ **until** n **do** NC(i)
3 $Q \leftarrow \{(i, j) |\ (i, j) \in \text{arcs}(G), i \neq j$
4 **While** Q is not empty **do**
5 **begin**
6 select and delete any arc (k, m) from Q
7 **if** REVISE(k, m) **then** $Q \leftarrow Q \cup \{(i, k) | (i, k) \in \text{arcs}(G), i \neq k, i \neq m\}$
8 **end**
9 **end**

First $|Q| = 2e$, then it grows and shrinks 4—8

Worst case:
$$\begin{cases} \text{1 element is deleted } D_{V_k} \text{ per iteration} \\ \text{none of the arcs added is in } Q \end{cases}$$

Line 7: $a.(d_k - 1)$
Lines 4–8: $2e + \sum_{k=1}^{n} a.(d_k - 1) = 2e + a.(2e - n)$
`REVISE`: a^2 checks of P_{ij}

$\longrightarrow$ AC-3 is $\mathcal{O}(a^2(2e + a(2e - n)))$
Connected graph $\Rightarrow e \geq n - 1$, AC-3 is $\mathcal{O}(a^3 e)$

If AC-p, AC-3 is $\mathcal{O}(a^2 e) \Rightarrow$ AC-3 is $\Omega(a^2 e)$
Complete graph: $\mathcal{O}(a^3 n^2)$

Constraints as relations

Check Section 1.2, Chapter 1, Dechter

Binary relations:

Given two sets D_a and D_b, any set of 2-tuples $\langle x, y \rangle$ with $x \in D_a$ and $y \in D_b$ defines a <u>relation</u> R_{ab}

Function: (vs. Relation)

For any element x in D_a there is <u>at most one</u> tuple $\langle x, ? \rangle$ in R_{ab}

D_a is called the <u>domain</u> of the the function

D_b is called the <u>range</u> of the function

A constraint is a relation

For constraints of arity k, the relation is a set of k-tuples

$$C_{V_a, V_b, \ldots, V_i} = R_{V_a, V_b, \ldots, V_i} = \{\, (x, y, \ldots, z)\} \subseteq D_{V_a} \times D_{V_b} \times \ldots \times D_{V_i}$$

Relations/constraints as matrices

Binary constraint/relation → 2-dim binary matrix:

$$C_{V_k,V_l} = \{\,(x,y)\} \subseteq D_{V_k} \times D_{V_l}$$

$$\begin{pmatrix} 1 & 0 & 1 & \ldots & 1 \\ 0 & \ldots & & & 1 \\ 1 & \ldots & & & 1 \end{pmatrix}$$

k**-ary constraint** → k-dimensional <u>binary</u> matrix

Operations on relations

- Intersection
- Union
- Difference
- Selection
- Projection
- Join
- Composition

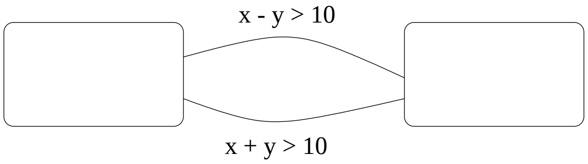
Intersection, union, difference: relations must have same scope
Selection, projection: defined on one relation
Join: any number of relations with any scope
Composition: 2 binary relations with 1 common variable/attribute

Intersection

Input: two or more relations of the same scope

Output: a new more <u>restrictive</u> relation with the same scope made of tuples that are in <u>all</u> the input relations (simultaneously)

Matrice operation: logical `AND`



R

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

R'

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

R''

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

$R \cap R'$?

$R \cap R''$?

Union

Input: two or more relations of the same scope

Output: a new less <u>restrictive</u> relation with the same scope made of tuples that are in <u>any</u> of the input relations

Matrice operation: logical `OR`

R

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

R'

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

R''

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

$R \cup R'$?

$R \cup R''$?

Difference

Input: two relations R and R' of the same scope

Output: a new more restrictive relation than R made of tuples that are in R but not in R'

Matrice operation: Boolean difference

R

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

R'

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

R''

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

$R - R'$?

$R - R''$?

Selection

Input: A relation R and some test/predicate

Output: A relation R', same scope than R but with only a subset of the tuples in R (those that satisfy the predicate)

Relation operation: row selection

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

Select such than $x_1 > x_2$

Projection

Input: A relation R and a subset s of the scope (attributes)

Output: A relation R' of scope s with the tuples rewritten such than positions not in s are removed

Relation operation: column elimination

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

Project R on x_1, x_2?

Composition of relations [Montanari'74]

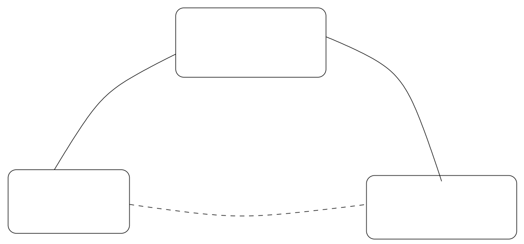
Input: two binary relations R_{ab} and R_{bc} with 1 variable in common

Output: a new induced relation R_{ac}, to be combined by intersection to an existing relation between them, if any.

Matrice operation: binary matrix multiplication

$$R_{ac} = R_{ab} \cdot R_{bc} \quad R_{ac,rs} = \bigvee_{t=1}^{D_b} R_{ab,rt} \wedge R_{ab,ts}$$

Note: non-binary relations? relational consistency (Dechter)



$\longrightarrow$ **Direct vs. induced relations**

$$R_{ab} = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \text{ and } R_{bc} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{pmatrix}$$

R_{ac}?

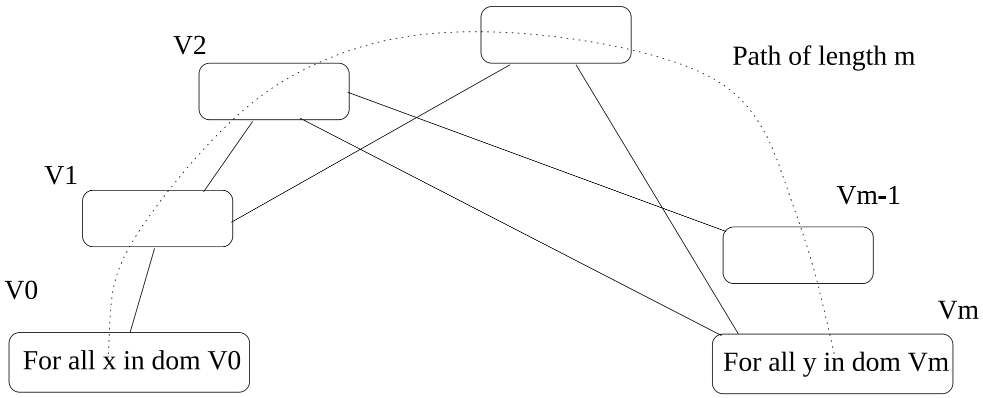
Path consistency (I)

A path $(V_0, V_1, V_2, \ldots, V_m)$ of length m is **path consistent** iff:

1. for any value x in domain of $V0$ and

2. for any valye y is domain of Vm, that are consistent (i.e., $P_{V_0 V_m}(x, y)$)

$\exists$ a sequence of values $z_1, z2, \ldots, z_{m-1}$ in the domains of variables $V_1, V_2, \ldots, V_{m-1}$, such that all constraints **among them** are satisfied (i.e., $P_{V_0 V_1}(x, z_1) \wedge P_{V_1 V_2}(z_1, z_2) \wedge \ldots$)



Definition: A CSP is path consistent iff every path is consistent

(i.e., any length of path)

Path consistency (II)

⟶ A path of length m is path consistent

⟶ A CSP is path consistent

Theorem: If every path of length 2 of network is path consistent, then the network is path consistent

Definition: Complete graph

Theorem: In a **complete** graph, if every path of length 2 is consistent, the network is path consistent (and is minimal).

⟶ Proof by induction

New developments for Path consistency

Definition: triangulated or chordal graph

Bliek & Sam-Haroud (IJCAI 99):

- Theorem holds for triangulated graphs
- For convex constraints: PC has same pruning power on triangulated as on complete graphs
- For non-convex constraints: comparable performance of pruning shown by experimental evidence (not theoretic).

Path consistency (PC-1)

Achieved by composition and intersection (of binary relations expressed as matrices) over all paths of length two.

Procedure PC-1:

Begin
    $Y^n \leftarrow R$
    repeat
        begin
        $Y^0 \leftarrow Y^n$
        For$k \leftarrow 1$ **until** n **do**
            For$i \leftarrow 1$ **until** n **do**
                For$j \leftarrow 1$ **until** n **do**
                    $Y_{ij}^{k} \leftarrow Y_{ij}^{k-1} \cap Y_{ik}^{k-1} \cdot Y_{kk}^{k-1} \cdot Y_{kj}^{k-1}$
        end
    until $Y^n = Y^0$
    $Y \leftarrow Y^n$
end

$\longrightarrow$ terminates, complete, sound

PC-2 Improvement similar to (AC-1 $\rightarrow$ AC-3)

Complexity of PC-1

Procedure PC-1:

1 **Begin**
2 $\quad Y^n \leftarrow R$
3 $\quad$ **repeat**
4 $\qquad$ **begin**
5 $\qquad Y^0 \leftarrow Y^n$
6 $\qquad$ For$k \leftarrow 1$ **until** n **do**
7 $\qquad\quad$ For$i \leftarrow 1$ **until** n **do**
8 $\qquad\qquad$ For$j \leftarrow 1$ **until** n **do**
9 $\qquad\qquad\quad Y_{ij}^{k} \leftarrow Y_{ij}^{k-1} \cap Y_{ik}^{k-1} \cdot Y_{kk}^{k-1} \cdot Y_{kj}^{k-1}$
10 $\qquad$ end
11 $\quad$ **until** $Y^n = Y^0$
12 $\quad Y \leftarrow Y^n$
10 **end**

Line 9: a^3
Lines 6–10: $n^3.a^3$
Line 3: at most n^2 relations$\times a^2$ elements

PC-1 is $\mathcal{O}(a^5 n^5)$

PC-3 is $\mathcal{O}(a^5 n^3)$ and $\Omega(a^3 n^3)$

Main improvements

Mohr & Henderson (AIJ 86):

$$\begin{cases} \text{AC-3, } \mathcal{O}(a^3e) \longrightarrow \text{AC-4, } \mathcal{O}(a^2e) \\ \text{AC-4 is optimal.} \\ \\ \text{PC-2, } \mathcal{O}(a^5n^3) \longrightarrow \text{PC-3, } \mathcal{O}(a^3n^3) \\ \text{Open question: PC-3 optimal?} \end{cases}$$

Han & Lee (AIJ 88):

$$\begin{cases} \text{PC-3 is incorrect} \\ \text{PC-4, } \mathcal{O}(a^3n^3) \text{ space and time} \end{cases}$$

In practical applications:

$\surd$ AC-1, AC-2, etc. (up to AC-7?)
$\times$ PC-1, PC-2, etc.

Summary

The more consistent, the better for search

- New terms: Waltz algorithm, AC, PC, minimal network, path consistent, complete graph, triangulated graph, Refine, Revise,
- Operation on constraints/relations, etc.
- Complexity of AC/PC algorithms
- Warning: Worst-case complexity is pessimistic.
  AC-4 has better worst-case complexity than AC-3, but AC-3 has better average complexity..
- Future: higher level of consistency? integration with search?