Title: Consistency Algorithms and operations on constraints
Material: Algorithms for Constraint Satisfaction Problems
[Mackworth and Freuder AIJ'85]

Foundations of Constraint Satisfaction
CSCE421/821, Fall 2002
www.cse.unl.edu/~choueiry/F02-421-821/

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

Required reading:

Algorithms for Constraint Satisfaction Problems, Mackworth and Freuder AIJ'85

Recommended reading:

1. Networks of Constraints: Fundamental Properties and Application to Picture Processing, Montanari, Information Sciences 74
2. Consistency in Networks of Relations, Mackworth AIJ'77
3. Constraint Propagation with Interval Labels, Davis, AIJ'87
4. Path Consistency on Triangulated Constraint Graphs, Blieck & Sam-Haroud IJCAI'99

Upon demand:

Notes of Rina Dechter, Chapter 3. Confidential, available only to class

Outline

1. Motivation and material
2. Davis: Waltz algorithm, quick discussion
3. Node and arc consistency and their complexity
4. Criteria for performance comparison and CSP parameters
   Which AC algorithm to choose? Project results may tell!
5. Path consistency and its complexity
6. Minimality and decomposability
7. Other related results: box consistency, higher levels of consistency, (i, j)-consistency, etc.

Consistency checking

Motivation

CSP are solved with search (conditioning)

Search performance is affected by:
- problem size
- amount of backtracking

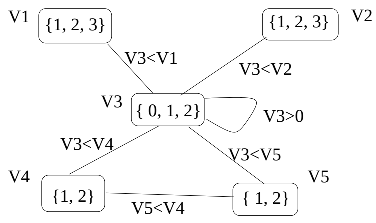
Backtracking may yield **thrashing**

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: eliminate inconsistent combinations

Algorithms

Binary constraints:

Node consistency
Arc consistency (AC-1, AC-2, AC-3, ..., AC-7, AC-3.1, etc.)
Path consistency (PC-1, PC-2, DPC, PPC, 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

→ Mackworth and Freuder concentrate on finite domains
→ More people work on finite domains than on 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)

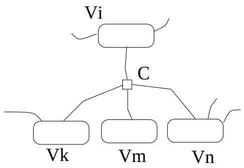
- Concerned with Waltz algorithm
  (e.g., quiescence, completeness)
  Constraint types vs. domain types (Table 3)

- Addresses applications from reasoning
  about Physical Systems (circuit, `SPAM`)
  about time relations (`TMM`)

⟶ Advice: read Section 3 at least (more if you wish)

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 <u>finite</u>, <u>binary</u> CSPs and their complexity

Notation:
Given variables i, j, k with values x, y, z, the predicate

$$P_{ijk}(x, y, z) \text{ is true iff the 3-tuple } \langle x, y, z \rangle \in C_{ijk}$$

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

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

Lower bound: $f(n)$ is $\Omega(h(n))$ means that $f(n) \geq c.h(n)$
f grows as g or faster

Input parameters for a CSP:

$a =$ (max) size of a domain

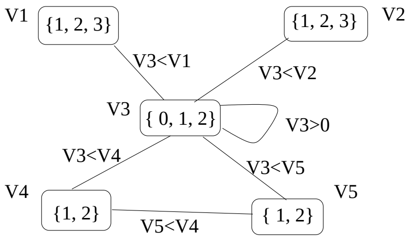
$d_k =$ degree of V_k ($\leq n-1$)

$n =$ 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

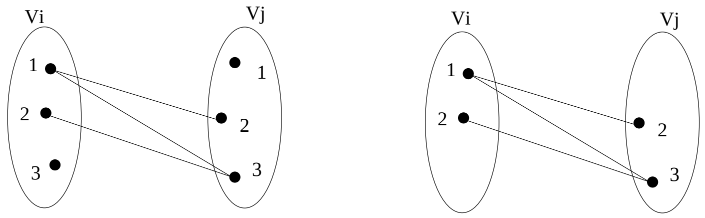
NC is $\mathcal{O}(a.n)$

Arc-consistency: Definition

Adapted from Rina Dechter

Given a constraint graph $\mathcal{G}$,

- A variable V_i is arc-consistent relative to V_j iff for every value a in D_{V_i}, there exists a value b in D_{V_j} such than $(a, b) \in C_{V_i,V_j}$.



- The constraint C_{V_i,V_j} is arc-consistent iff V_i is arc-consistent relative to V_j and V_j is arc-consistent relative to V_i.
- A binary CSP is arc-consistent iff every constraint (or subgraph of size 2) is arc-consistent

Procedure REVISE

Revises the domains of a variable i

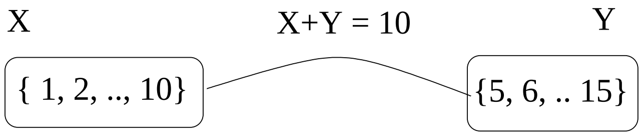
Procedure REVISE(i,j):
Begin
  DELETE $\leftarrow$ false
  for each $x \in D_i$ **do**
    if there is no $y \in D_j$ such that $P_{ij}(x, y)$ **then**
      begin
        delete x from D_i;
        DELETE $\leftarrow$ true
      end
  return DELETE
end

REVISE is directional

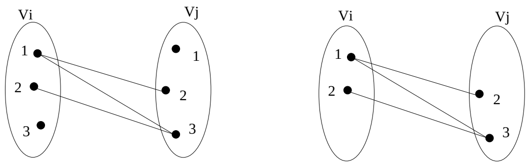
Complexity of REVISE? a^2

REVISE: Example *(courtesy of R. Dechter, UC-Irvine)*

Apply the REVISE procedure to the following example



Question: *Adapted from Rina Dechter*



Given two variables V_i and V_j, their domains D_{V_i} and D_{V_j}, and the constraint C_{V_i,V_j}, write the effect of the REVISE procedure as a sequence of operations in relational algebra

Hint:
Think about the domain D_{V_i} as a unary constraint C_{V_i} and consider the composition of this unary constraint and the binary one..

Solution:
$D_{V_i} \leftarrow D_{V_i} \cap \pi_{V_i}(C_{V_i,V_j} \bowtie D_{V_i})$, actually equivalent to

$D_{V_i} \leftarrow \pi_{V_i}(C_{V_i,V_j} \bowtie D_{V_i})$

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

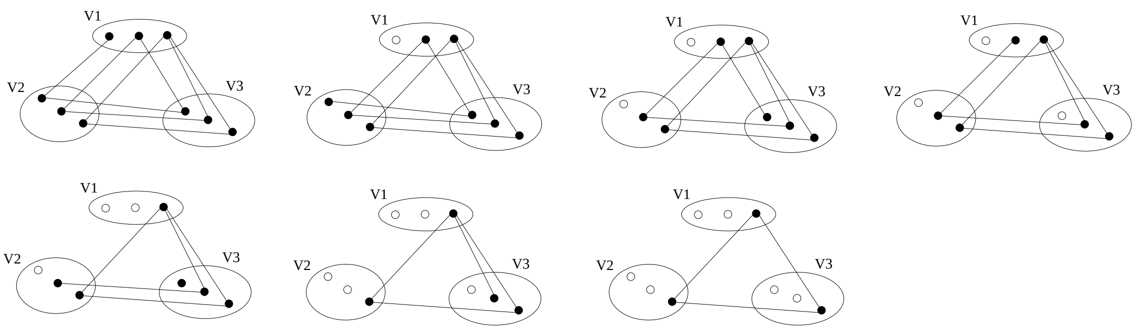
end

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

Note: no algorithm can have time complexity below ea^2

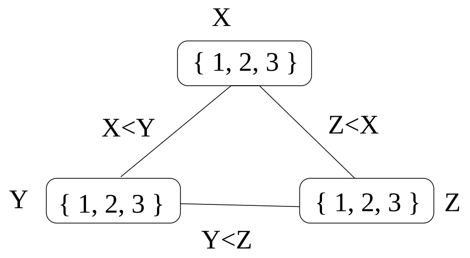
AC may discover the solution (I)

Example borrowed from Rina Dechter



AC may discover inconsistency (II)

Example borrowed from Rina Dechter



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)

  $Q \leftarrow \{(i,j) |\ (i,j) \in \text{arcs}(G), i \neq j$

  While Q is not empty **do**

    begin

    select and delete any arc (k,m) from Q

    if REVISE(k,m) **then** $Q \leftarrow Q \cup \{(i,k) | (i,k) \in \text{arcs}(G), i \neq k, i \neq m\}$

    end

end

Complexity of AC-3

```
Procedure AC-3:
1 Begin
2   for i ← 1 until n do NC(i)
3   Q ← {(i, j)| (i, j) ∈ arcs(G), i ≠ 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 ← Q ∪ {(i, k)|(i, k) ∈ arcs(G), i ≠ k, i ≠ 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}

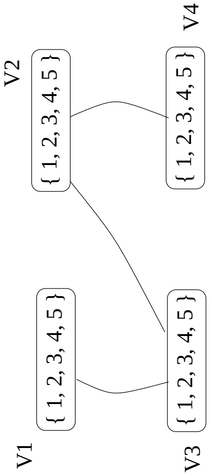
$\longrightarrow$ AC-3 is $\mathcal{O}(a^2(2e + a(2e - n)))$

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: $\mathcal{O}(a^3 n^2)$

Example: Apply AC-3



$D_{V_1} = \{1, 2, 3, 4, 5\}$

$D_{V_2} = \{1, 2, 3, 4, 5\}$

$D_{V_3} = \{1, 2, 3, 4, 5\}$

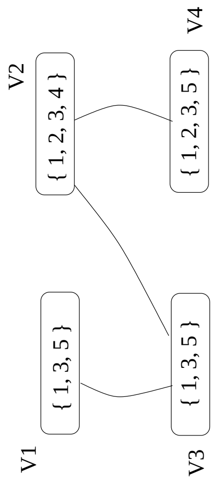
$D_{V_4} = \{1, 2, 3, 4, 5\}$

$C_{V_2,V_3} = \{(2,2),(4,5),(2,5),(3,5),(2,3),(5,1),(1,2),(5,3),(2,1),(1,1)\}$

$C_{V_1,V_3} = \{(5,5),(2,4),(3,5),(3,3),(5,3),(4,4),(5,4),(3,4),(1,1),(3,1)\}$

$C_{V_2,V_4} = \{(1,2),(3,2),(3,1),(4,5),(2,3),(4,1),(1,1),(4,3),(2,2),(1,5)\}$

Thanks to Xu Lin



Applying AC-3

Thanks to Xu Lin

- Queue = $\{C_{V_2,V_4}, C_{V_4,V_2}, C_{V_1,V_3}, C_{V_2,V_3}, C_{V_3,V_1}, C_{V_3,V_2}\}$
- Revise(V_2, V_4): $D_{V_2} \leftarrow D_{V_2} \setminus \{5\} = \{1,2,3,4\}$
- Queue = $\{C_{V_4,V_2}, C_{V_1,V_3}, C_{V_2,V_3}, C_{V_3,V_1}, C_{V_3,V_2}\}$
- Revise(V_4, V_2): $D_{V_4} \leftarrow D_{V_4} \setminus \{4\} = \{1,2,3,5\}$
- Queue = $\{C_{V_1,V_3}, C_{V_2,V_3}, C_{V_3,V_1}, C_{V_3,V_2}\}$
- Revise(V_1, V_3): $D_{V_1} \leftarrow \{1,2,3,4,5\}$
- Queue = $\{C_{V_2,V_3}, C_{V_3,V_1}, C_{V_3,V_2}\}$
- Revise(V_2, V_3): $D_{V_2} \leftarrow \{1,2,3,4\}$
- Queue = $\{C_{V_3,V_1}, C_{V_3,V_2}\}$
- Revise(V_3, V_1): $D_{V_3} \leftarrow D_{V_3} \setminus \{2\} = \{1,3,4,5\}$
- Queue = $\{C_{V_2,V_3}, C_{V_3,V_2}\}$
- Revise(V_2, V_3): $D_{V_2} \leftarrow D_{V_2}$
- Queue = $\{C_{V_3,V_2}\}$
- Revise(V_3, V_2): $D_{V_3} \leftarrow \{1,3,5\}$
- Queue = $\{C_{V_1,V_3}\}$
- Revise(V_1, V_3): $D_{V_1} \leftarrow \{1,3,5\}$

Main improvements

Thanks to Xu Lin

Mohr & Henderson (AIJ 86): AC-3 $\mathcal{O}(a^3e) \longrightarrow$ AC-4 $\mathcal{O}(a^2e)$
AC-4 is optimal

Trades repetition of consistency-check operations with heavy bookkeeping on which and how many values support a given value for a given variable;

- *m*: values that are active, not filtered
- *s*: lists all vvp's that support a given vvp
- *counter*: given a vvp, provides the number of support provided by a given variable

1. Generates data structures
2. Prepares data structures
3. Iterates over constraints

Step 1: Three data-structures are generated: m, s, and *counter*.

m	
$(V_4, 5)$	T
$(V_3, 2)$	T
$(V_3, 3)$	T
$(V_2, 2)$	T
$(V_3, 1)$	T
$(V_2, 3)$	T
$(V_1, 2)$	T
$(V_2, 1)$	T
$(V_1, 3)$	T
$(V_3, 4)$	T
$(V_4, 2)$	T
$(V_3, 5)$	T
$(V_4, 3)$	T
$(V_1, 1)$	T
$(V_2, 4)$	T
$(V_2, 5)$	T
$(V_4, 1)$	T
$(V_1, 4)$	T
$(V_1, 5)$	T
$(V_4, 4)$	T

s	
$(V_4, 5)$	NIL
$(V_3, 2)$	NIL
$(V_3, 3)$	NIL
$(V_2, 2)$	NIL
$(V_3, 1)$	NIL
$(V_2, 3)$	NIL
$(V_1, 2)$	NIL
$(V_2, 1)$	NIL
$(V_1, 3)$	NIL
$(V_3, 4)$	NIL
$(V_4, 2)$	NIL
$(V_3, 5)$	NIL
$(V_4, 3)$	NIL
$(V_1, 1)$	NIL
$(V_2, 4)$	NIL
$(V_2, 5)$	NIL
$(V_4, 1)$	NIL
$(V_1, 4)$	NIL
$(V_1, 5)$	NIL
$(V_4, 4)$	NIL

$counter$	
$(V_4, V_2), 3$	0
$(V_4, V_2), 2$	0
$(V_4, V_2), 1$	0
$(V_2, V_3), 1$	0
$(V_2, V_3), 2$	0
$(V_2, V_3), 3$	0
$(V_4, V_2), 5$	0
$(V_2, V_4), 1$	0
$(V_2, V_3), 4$	0
$(V_4, V_2), 4$	0
$(V_2, V_4), 2$	0
$(V_2, V_3), 5$	0
$(V_2, V_4), 3$	0
$(V_2, V_4), 4$	0
$(V_2, V_4), 5$	0
$(V_3, V_1), 1$	0
$(V_3, V_1), 2$	0
$(V_3, V_1), 3$	0
$(V_3, V_2), 1$	0
$(V_3, V_1), 4$	0
$(V_3, V_2), 2$	0
$(V_3, V_1), 5$	0
$(V_1, V_3), 1$	0
$(V_3, V_2), 3$	0
$(V_3, V_2), 4$	0
$(V_1, V_3), 3$	0
$(V_3, V_2), 5$	0
$(V_1, V_3), 2$	0
$(V_1, V_3), 5$	0
$(V_1, V_3), 4$	0

Step 2: Prepare data-structures s, and *counter*.

Checks for every constraint C_{V_i,V_j} all tuples ($V_i = a_i, V_j = b_j$)

When the tuple is allowed, then update:

$$s(V_j, b_j) \leftarrow s(V_j, b_j) + (V_i, a_i) \text{ and}$$
$$counter(V_j, b_j) \leftarrow counter(V_j, b_j) + 1$$

```
Update counter ((V_2, V_3), 2) to value 1
Update counter ((V_3, V_2), 2) to value 1
Update s-htable (V_2, 2) to value ((V_3, 2))
Update s-htable (V_3, 2) to value ((V_2, 2))

Update counter ((V_2, V_3), 4) to value 1
Update counter ((V_3, V_2), 5) to value 1
Update s-htable (V_2, 4) to value ((V_3, 5))
Update s-htable (V_3, 5) to value ((V_2, 4))

...
```

Consider the two constraints C_{V_2,V_3} and C_{V_3,V_2}.

$counter((V_2, V_3), 2)$ is 4 means that in the value 2 for V_2 is supported by 4 values for V_3.

s	
$(V_4, 5)$	NIL
$(V_3, 2)$	$((V_2, 1), (V_2, 2))$
$(V_3, 3)$	$((V_2, 5), (V_2, 2))$
$(V_2, 2)$	$((V_3, 1), (V_3, 3), (V_3, 5), (V_3, 2))$
$(V_3, 1)$	$((V_2, 1), (V_2, 2), (V_2, 5))$
$(V_2, 3)$	$((V_3, 5))$
$(V_1, 2)$	NIL
$(V_2, 1)$	$((V_3, 1), (V_3, 2))$
$(V_1, 3)$	NIL
$(V_3, 4)$	NIL
$(V_4, 2)$	NIL
$(V_3, 5)$	$((V_2, 3), (V_2, 2), (V_2, 4))$
$(V_4, 3)$	NIL
$(V_1, 1)$	NIL
$(V_2, 4)$	$((V_3, 5))$
$(V_2, 5)$	$((V_3, 3), (V_3, 1))$
$(V_4, 1)$	NIL
$(V_1, 4)$	NIL
$(V_1, 5)$	NIL
$(V_4, 4)$	NIL

counter	
$(V_4, V_2), 3$	0
$(V_4, V_2), 2$	0
$(V_4, V_2), 1$	0
$(V_2, V_3), 1$	2
$(V_2, V_3), 2$	4
$(V_2, V_3), 3$	1
$(V_4, V_2), 5$	0
$(V_2, V_4), 1$	0
$(V_2, V_3), 4$	1
$(V_4, V_2), 4$	0
$(V_2, V_4), 2$	0
$(V_2, V_3), 5$	2
$(V_2, V_4), 3$	0
$(V_2, V_4), 4$	0
$(V_2, V_4), 5$	0
$(V_3, V_1), 1$	0
$(V_3, V_1), 2$	0
$(V_3, V_1), 3$	0
$(V_3, V_2), 1$	3
$(V_3, V_1), 4$	0
$(V_3, V_2), 2$	2
$(V_3, V_1), 5$	0
$(V_1, V_3), 1$	0
$(V_3, V_2), 3$	2
$(V_3, V_2), 4$	0
$(V_1, V_3), 3$	0
$(V_3, V_2), 5$	3
$(V_1, V_3), 2$	0
$(V_1, V_3), 5$	0
$(V_1, V_3), 4$	0

Note that $counter[(V_3, V_2), 4] = 0$,

we need to remove 4 from domain of V_3

and update m

Since we remove $(V_3, 4)$, then for every element in (V_k, l) in $s[(V_3, y)]$ we decrement the $counter(V_k, V_3), l)$ by 1. (not shown here)

Eventually:

m	
$(V_4, 5)$	T
$(V_3, 2)$	NIL
$(V_3, 3)$	T
$(V_2, 2)$	T
$(V_3, 1)$	T
$(V_2, 3)$	T
$(V_1, 2)$	NIL
$(V_2, 1)$	T
$(V_1, 3)$	T
$(V_3, 4)$	NIL
$(V_4, 2)$	T
$(V_3, 5)$	T
$(V_4, 3)$	T
$(V_1, 1)$	T
$(V_2, 4)$	T
$(V_2, 5)$	NIL
$(V_4, 1)$	T
$(V_1, 4)$	NIL
$(V_1, 5)$	T
$(V_4, 4)$	NIL

Summary of arc-consistency algorithms

- AC-4 is optimal (worst-case) [Mohr & Henderson, AIJ 86]
- Warning: Worst-case complexity is pessimistic.
  Better worst-case complexity: AC-4.
  Better average behavior: AC-3 [Wallace, IJCAI 93]
- AC-5: special constraints [Hentenryck, Deville, Teng 92]
  functional, anti-functional, and monotonic constraints
- AC-6, AC-7: general but rely heavily on data structures for bookkeeping [Bessière & Régin]
- Now, back to AC-3: AC-2000, AC-2001, and AC-3.1
- Non-binary constraints: GAC [Mohr & Masini 1988]

CSP parameters $\langle n, a, t, d \rangle$

- n is number of variables
- a is maximum domain size
- t is constraint tightness: $t = \frac{|forbidden\ tuples|}{|all\ tuples|}$
- d is constraint density $d = \frac{e - e_{min}}{e_{max} - e_{min}}$
  where e is the number of constraints,
  $e_{min} = (n-1)$, and
  $e_{max} = \frac{n(n-1)}{2}$

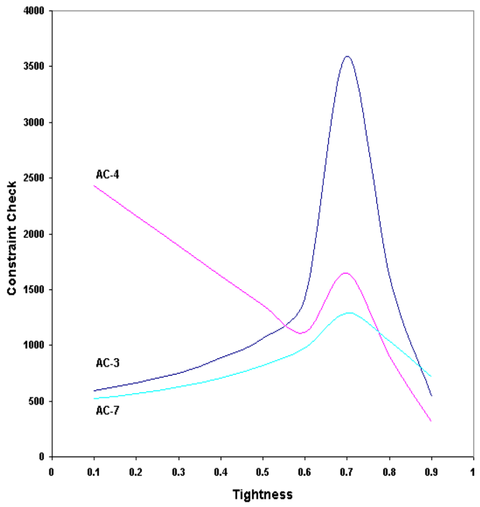
$\longrightarrow$ Constraints in random problem often generated uniform

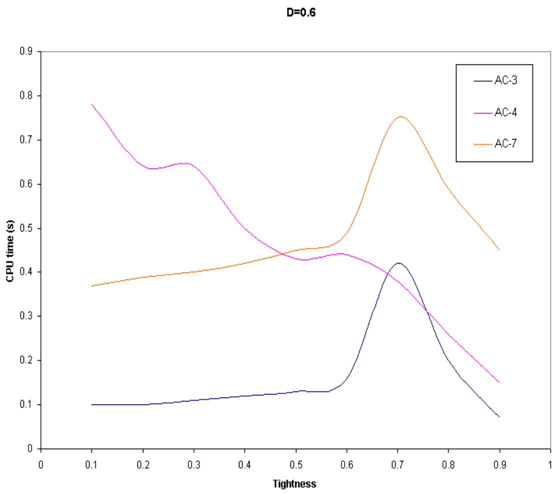
Criteria for performance comparison

- Worst-case, average-case complexity, time and space (theoretical)
- Number of constraint checks (theoretical, empirical)
- CPU time (empirical)

Performance comparison

Courtesy of Lin XU





Wisdom (?)

Free adaptation from Bessière

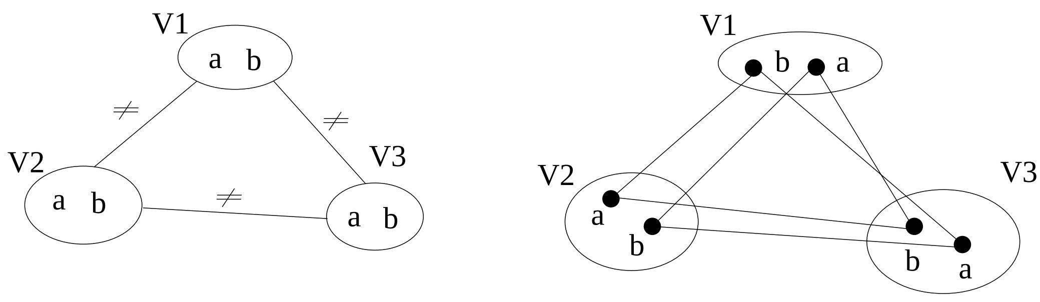
- When a single constraint check is very expensive to make, use AC-7.
- When there is a lot of propagation, use AC-4
- When little propagation and constraint checks are cheap, use AC-3x

Advice? *Instructor's personal opinion*

Use AC-3..

AC is not enough

Example borrowed from Rina Dechter



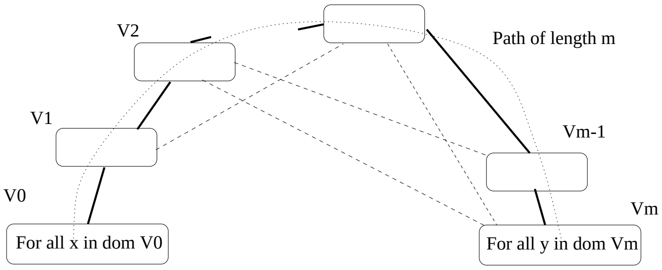
Arc-consistent?

Satisfiable?

$\longrightarrow$ seek higher levels of consistency

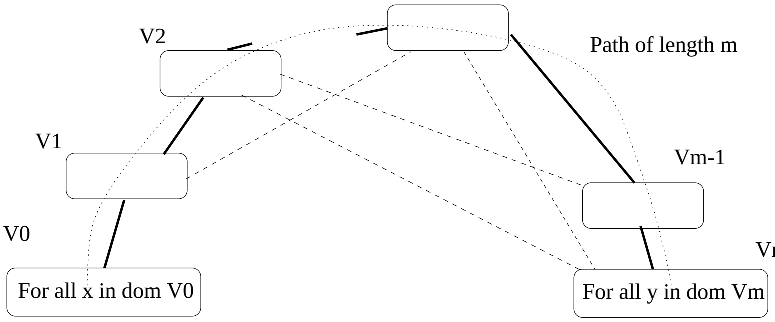
Consistency of a path

A path $(V_0, V_1, V_2, \ldots, V_m)$ of length m is **consistent** iff for any value x in domain of V_0 and for any value 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., $P_{V_0 V_1}(x, z_1) \wedge P_{V_1 V_2}(z_1, z_2) \wedge \ldots$)



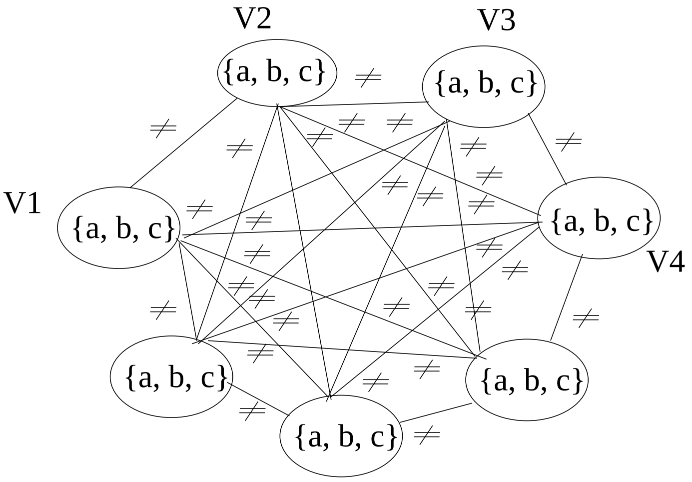
Note that

→ The same variable can appear more than once in the path
→ Every time, it may have a different value
→ Constraints considered: P_{V_0, V_m} and those along the path
→ All other constraints neglected



Example: consistency of a path

→ Check path length =2, 3, 4, 5, 6, ...



Path consistency: definition

⟶ A path of length m is path consistent
⟶ A CSP is path consistent *Property of a CSP*

Definition: A CSP is path consistent)PC) iff every path is consistent (i.e., any length of path)

Question: should we enumerate every path of any length?
Answer: No, only length 2, thanks to [Mackworth AIJ'77]

Making a CSP Path Consistent (PC)

Special case: Complete graph
Theorem:
In a **complete** graph, if every path of length 2 is consistent, the network is path consistent [Mackworth AIJ'77]
$\longrightarrow$ PC-1: two operations, composition and intersection
$\longrightarrow$ Proof by induction

Special case: Triangulated graph
Theorem:
In a **triangulated** graph, if every path of length 2 is consistent, the network is path consistent [Blieck & Sam-Haroud IJCAI'99]
$\longrightarrow$ PPC (partially PC) $\Rightarrow$ PC

Tools for PC-1: two operators

1. Constraint composition: $(\cdot)$

$$R_{13} = R_{12} \cdot R_{23}$$

2. Constraint intersection: $(\cap)$

$$R_{13} \leftarrow R_{13,old} \cap R_{13,induced}$$

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 ← R
    repeat
        begin
        Y^0 ← Y^n
        For k ← 1 until n do
            For i ← 1 until n do
                For j ← 1 until n do
                    Y_ij^k ← Y_ij^(k-1) ∩ Y_ik^(k-1) · Y_kk^(k-1) · Y_kj^(k-1)
        end
    until Y^n = Y^0
    Y ← Y^n
end
```

$$Y_{ij}^{k} \leftarrow Y_{ij}^{k-1} \cap Y_{ik}^{k-1} \cdot Y_{kk}^{k-1} \cdot Y_{kj}^{k-1}$$

Properties of PC-1

[Montanari'74]

Discrete CSPs:

1. PC-1 terminates
2. PC-1 results in a path consistent CSP

PC-1 terminates. It is complete, sound (for finding PC network)

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

Complexity of PC-1..

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 $\mathcal{O}(a^5 n^5)$

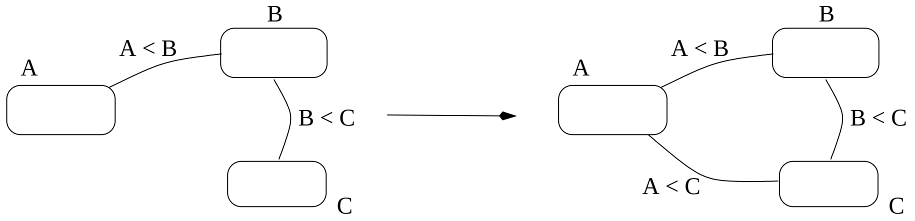
PC-3 is $\mathcal{O}(a^5 n^3)$ and $\Omega(a^3 n^3)$

Some improvements

- **Mohr & Henderson** (AIJ 86):
  PC-2 $\mathcal{O}(a^5 n^3) \longrightarrow$ PC-3 $\mathcal{O}(a^3 n^3)$
  Open question: PC-3 optimal?
- **Han & Lee** (AIJ 88):
  PC-3 is incorrect
  PC-4 $\mathcal{O}(a^3 n^3)$ space and time
- **Singh** (ICTAI 95):
  PC-5 uses ideas of AC-6 (support bookeeping)

Note:
PC is seldom used in practical applications unless in presence of special type of constraints (e.g., bounded difference)

Path consistency as inference of binary constraints

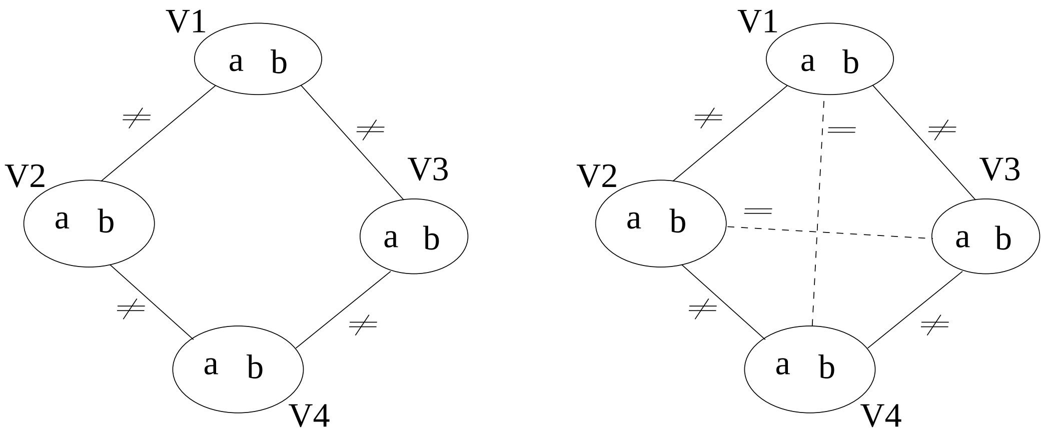


Path-consistency corresponds to inferring a new constraint (alternatively, tightening the existing constraint) between every two variables given the constraints that link them to a third variable

$\longrightarrow$ Considers all subgraphs of 3 variables
$\longrightarrow$ 3-consistency

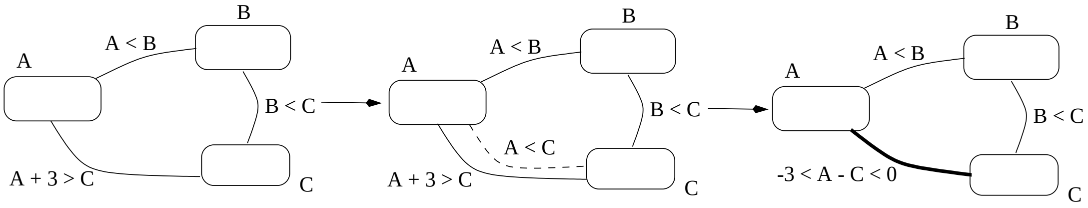
Path consistency as inference of binary constraints

Another example:



Question: *Adapted from Rina Dechter*

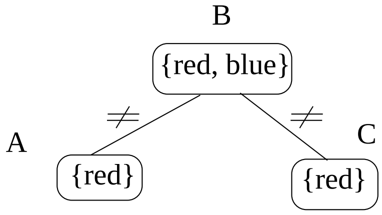
Given three variables V_i, V_k, and V_j and the constraints C_{V_i,V_k}, C_{V_i,V_j}, and C_{V_k,V_j}, write the effect of PC as a sequence of operations in relational algebra



Solution:

$$C_{V_i,V_j} \leftarrow C_{V_i,V_j} \cap \pi_{ij}(C_{V_i,V_k} \bowtie C_{V_k,V_j})$$

Warning: Does 3-consistency guarantee 2 consistency?



Is this CSP 3-consistent? is it 2-consistent?

Definitions: k-consistency vs. strong k-consistency

- k-consistency: any consistent assignment for $(k-1)$-variable can be extended to any k^{th} variable
- strong k-consistency: has to be j-consistent for all $j \leq k$

Important properties of a CSP

1. (Arc Consistency, AC)
2. Path Consistency, PC
3. Minimality
4. Decomposability (ref. lossless join decomposition)

Minimality

PC tightens the binary constraints

The tightest possible binary constraints yield the minimal network

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.

Note:

- Minimal $\Rightarrow$ path consistent
- The definition of minimal CSP is concerned with binary CSPs

Minimal CSP

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.

Informally:

- In a minimal CSP the remainder of the CSP does not add any further constraint the direct constraint C_{V_i,V_j} between the two variables V_i and V_j [Mackworth AIJ'77]
- A minimal CSP is perfectly explicit: as far as the pair V_i and V_j is concerned, the rest of the network does not add any further constraint to the direct constraint C_{V_i,V_j} [Montanari'74]
- The binary constraints are explicit as possible. [Montanari'74]

Decomposability

- Any combination of values for k variables that satisfy the constraints between them can be extended to a solution
- Decomposability generalizes minimality
  Minimality: any consistent combination of values for 2 variables is extendable to a solution
  Decomposability: any consistent combination of k values for variables is extendable to a solution
- Decomposable $\Rightarrow$ minimal $\Rightarrow$ path consistent

In general:

- Decomposability $\Rightarrow$ minimality $\Rightarrow$ path consistent
- PC is used to approximate minimality (which is the central problem)

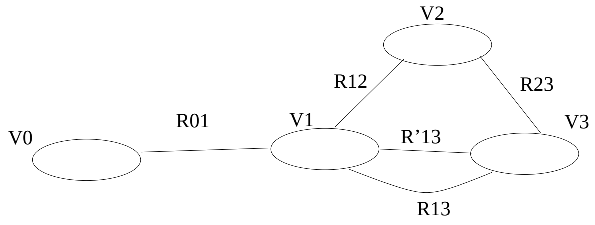
When is the approximation the real thing?

Special cases:

- When composition distributes over intersection, [Montanari'74]
  PC $\wedge$ complete graph $\Rightarrow$ minimal $\Rightarrow$ decomposable
- When constraints are convex, [Blieck & Sam-Haroud IJCAI'99]
  PC $\wedge$ triangulated graph $\Rightarrow$ minimal $\Rightarrow$ decomposable

PC guarantees minimality, decomposability

In PC-1, two operations: $\begin{cases} \text{intersection} & \cap \\ \text{composition} & \cdot \end{cases}$



$$R_{01} \cdot (R_{13} \cap R'_{13}) = (R_{01} \cdot R_{13}) \cap (R_{01} \cdot R'_{13})$$

When $(\cdot)$ distributes over $(\cap)$, then [Montanari'74]

1. PC guarantees that CSP is minimal and decomposable
2. 1 loop in PC-1 is sufficient (PC collapses to Floyd-Warshall)

Temporal reasoning: constraints of bounded difference

Variables: $X, Y, Z, etc.$

Constraints: $a \leq Y - X \leq b$, i.e. $Y - X = [a, b] = I$

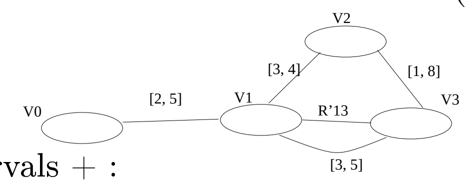
Composition: $I_1 \cdot I_2 = [a_1, b_1] \cdot [a_2, b_2] = [a_1 + a_2, b_1 + b_2]$
Interpretation: intervals indicate distances, composition is triangle inequality

Intersection: $I_1 \cap I_2 = [max(a_1, a_2), min(b_1, b_2)]$

Distributivity: $I_1 \cdot (I_2 \cap I_3) = (I_1 \cdot I_2) \cap (I_1 \cdot I_3)$

Prove: left as exercise

Example: constraints of bounded difference (temporal reasoning)



Composition of intervals $+$:
$R'_{13} = R_{12} + R_{23} = [4, 12]$
$R_{01} + R_{13} = [2, 5] + [3, 5] = [5, 10]$
$R_{01} + R'_{13} = [2, 5] + [4, 12] = [6, 17]$

Intersection of intervals: $R_{13} \cap R'_{13} = [4, 12] \cap [3, 5] = [4, 5]$

$$R_{01} + (R_{13} \cap R'_{13}) = (R_{01} + R_{13}) \cap (R_{01} + R'_{13})$$

$R_{01} + (R_{13} \cap R'_{13}) = [2, 5] + [4, 5] = [6, 10]$
$(R_{01} + R_{13}) \cap (R_{01} + R'_{13}) = [5, 10] \cap [6, 17] = [6, 10]$

Path consistency guarantees minimality

Side note:

PC-1 generalizes Floyd-Warshall algorithm (all-pairs shortest paths), where composition is 'scalar addition' and intersection is 'scalar minimal'

Convex constraints: temporal reasoning (again!)

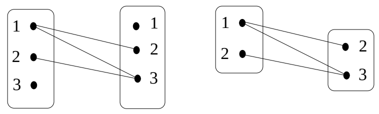
Thanks to Xu Lin (2002)

- Constraints of bounded difference are convex
- We triangulate the graph (good heuristics exist)
- Apply PPC: restrict propagations in PC to triangles of the graph (and not in the complete graph)
- According to [Bliek & Sam-Haroud IJCAI'99] PPC becomes equivalent to PC, thus guarantees minimality and decomposability

Constraint propagation

(courtesy of R. Dechter, UC-Irvine)

After Arc-consistency:

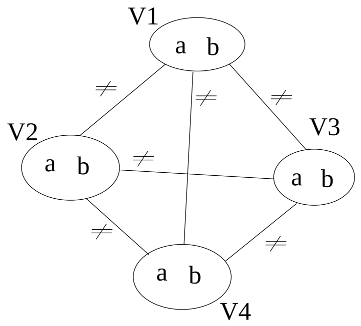


After Path-consistency:

Are these CSPs the same?
Which one is more explicit?
Are they equivalent?

The more propagation,
the more explicit the constraints,
the more search is directed towards a solution.

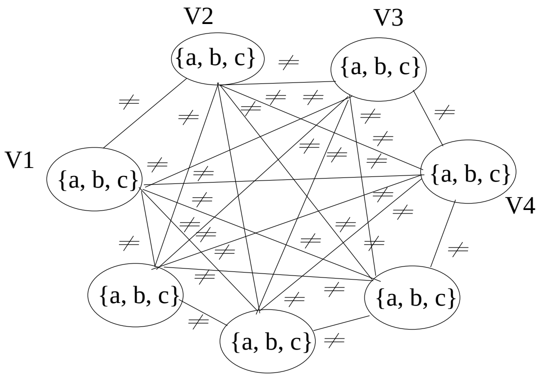
PC can detect unsatisfiability



Arc-consistent?

Path-consistent?

PC is not enough



Arc-consistent?

Path-consistent?

Satisfiable?

$\longrightarrow$ seek (even) higher levels of consistency

$\longrightarrow$ k-consistency, $k = 1, 2, 3, \ldots$

k-consistency

- 2-consistency: $D_{V_i} \leftarrow D_{V_i} \cap \pi_{V_i}(C_{V_i,V_j} \bowtie D_{V_j})$
  Projection of binary constraint is used to tighten unary constraint
- 3-consistency: $C_{V_i,V_j} \leftarrow C_{V_i,V_j} \cap \pi_{V_i,V_j}(C_{V_i,V_k} \bowtie C_{V_k,V_j})$
  Projection of ternary constraint is used to tighten binary constraint
- k-consistency: [Freuder CACM'78, JACM'82]
  Projection of k-ary constraint is used to tighten $(k-1)$-ary constraint

k-consistency: definition

A CSP is k-consistent iff given any consistent instantiation of any $(k-1)$ distinct variables, there exists an instantiation of any k^{th} variable such that the k values taken together satisfy all the constraints among the k variables.

Example: *(courtesy of R. Dechter, UC-Irvine)*

3-consistent?

Q			
	Q		

4-consistent?

Q			
		Q	
	Q		

Summary: the higher the consistency level, the better

- New terms: Waltz algorithm (Refine, Revise), AC (Revise), PC, path consistent, complete graph, minimal CSP, decomposable CSP
- Complexity of AC, PC algorithms
- AC tightens domains, PC tightens binary constrains
- Consistency operations in terms of composition, intersection and relational algebra
- Path consistency approximates minimality, which approximates decomposability
- Situations when path consistency guarantees minimality and decomposability
- Criteria for performance comparison
- Parameters for random CSPs

Related results

- Continuous domains: box consistency
- Higher consistency levels:
  (strong) k-consistency, (i, j)-consistency [Freuder]
- Integration with search:
  – Directed path consistency (DPC) [Dechter]
  – Forward Checking (FC) [Haralick & Elliott]
  – Maintaining arc consistency (MAC) [Sabin & Freuder]
- Non-binary constraints:
  – Generalized arc consistency (GAC) [Mohr & Masini]
  – Relational consistency [Dechter]
- Non-binary constraints and integration with search: nFC_0, nFC_1, …, nFC_5
- etc.