Consistency Enforcement and Constraint Propagation

Chapter 3 @ Constraint Processing by Rina Dechter

Constraint propagation

- Inference methods used in everyday life that can be imitated by computers
- Party example (Boolean constraint propagation)
  - Invite Alex, Bill and Chris to a party
  - (A) Alex comes, (B) Bill comes, (C) Chris comes
  - If Alex comes, Bill will come as well ($A \rightarrow B$)
  - If Chris comes, Alex will come as well ($C \rightarrow A$)
  - Fact: Chris will come to the party; Inference: Alex and Bill will come too
  - Fact: Bill did not go to the party; Inference: Alex and Chris did not go!

Why propagate constraints?

- Inference narrows the search space of possible partial solutions
  - By creating equivalent, yet more explicit, networks

- Another constraint network
  - Variables x, y, z with domains {red, blue}
  - Constraints (1) $x=y$, (2) $y=z$, (3) $x\neq z$
  - Infer (4) $x=z$ from (1) and (2)
  - Since (4) conflicts with (3) the constraint network is inconsistent

- Constraints can become explicit enough to go directly to the solution!
  - In general this is too hard, requiring exponential number of constraints

Another example

- Set of constraints R = {x=y, y=z}
- Infer x=z, resulting in R’ = {x=y, y=z, x=z}
- R and R’ are equivalent
  - Have the same set of solutions
  - But R’ is more explicit / tighter than R
- *Assignment {x=red, z=blue} is partial solution to R but not to R’*

Limited constraint inference

- Algorithms that perform a bounded amount of constraint inference
  - Local consistency enforcing
  - Bounded consistency inference
  - Constraint propagation algorithms

- Consistency enforcing algorithms assist search
  - By extending a solution by one more variable
  - i.e. a partial solution of a subnetwork is extended to surrounding network

Consistency enforcing algorithms

- Characterized by size of the subnetwork
  - Either number of variables or number of constraints
- Arc-consistency algorithms
  - Infer constraints based on pairs of variables
  - Ensures that any legal value in the domain of a variable has a legal match in the domain of any other variable
- Path consistency algorithms
  - Infer constraints based on trios of variables
  - Ensure that any consistent solution to a 2-variable subnetwork is extendible to any 3rd variable
  - In general, i-consistency algorithms extend solutions to i-1 variables

Consistency vs search

- Time and space cost of enforcing i-consistency is exponential in i
  - So there is a trade-off

(partial solution)

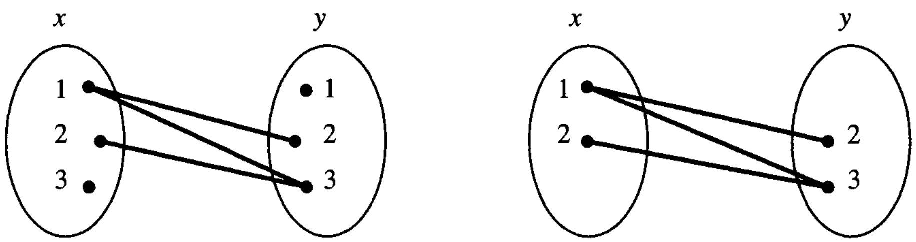
Given a constraint network $\mathcal{R}$, we say that an assignment of values to a subset of the variables $S = \{x_1, \ldots, x_j\}$ given by $\bar{a} = (\langle x_1, a_1 \rangle, \langle x_2, a_2 \rangle, \ldots, \langle x_j, a_j \rangle)$ is consistent relative to $\mathcal{R}$ iff it satisfies every constraint R_{S_i} such that $S_i \subseteq S$. The assignment $\bar{a}$ is also called a partial solution of $\mathcal{R}$. The set of all partial solutions of a subset of variables S is denoted by ρ_S or $\rho(S)$. ●

Arc-consistency

- A constraint is arc-consistency (or not) relative to a given variable
- A variable is arc-consistency (or not) relative to other variables

Arc-consistency: example

- Variables x, y with domains {1,2,3}; constraint R_{xy} = {x<y}
- Matching diagram (a): domains are sets of points; arcs connect consistent pairs of variables
  - R_{xy} is not arc-consistent relative to x: value 3 ∈ D_x has no consistent matching value in D_y
  - R_{xy} is not arc-consistent relative to y: value 1 ∈ D_y has no consistent matching value in D_x
- Shrink domains to achieve arc consistency (b)



Arc-consistency: definition

(arc-consistency)

Given a constraint network $\mathcal{R} = (X, D, C)$, 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. •

The REVISE procedure

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**

- 1+2+3 = $D_i \leftarrow D_i \cap \pi_i(R_{ij} \bowtie D_j)$
- REVISE has complexity $O(k^2)$ where k bounds the domain size

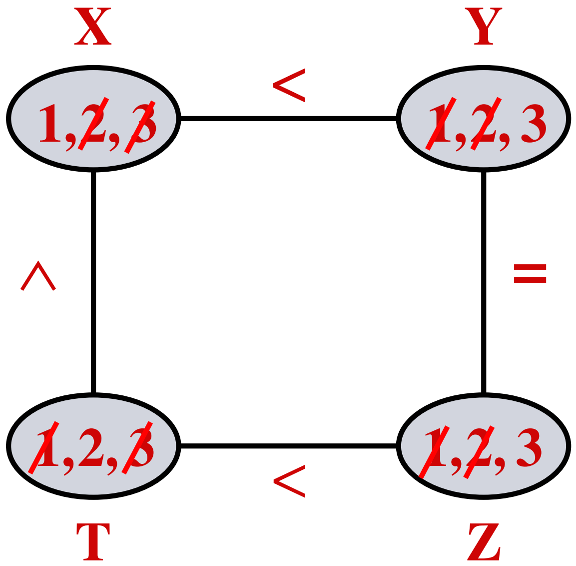
Arc-consistency: exercise

You have
5 minutes!

- Variables X, Y, Z, T
- Domains {1,2,3}
- Constraints $X < Y$
  $Y = Z$
  $T < Z$
  $X < T$

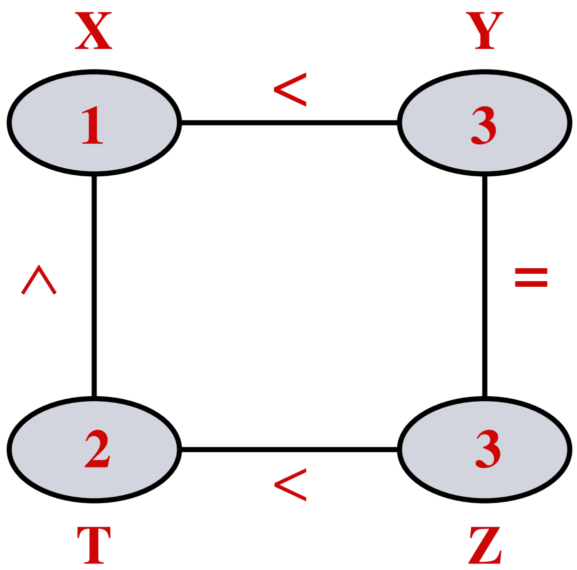
Original constraint network

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



Revised constraint network

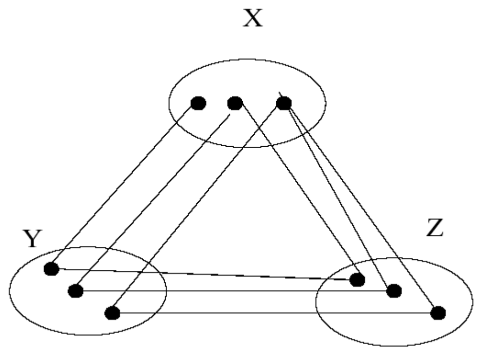
$1 \leq X, Y, Z, T \leq 3$
$X < Y$
$Y = Z$
$T < Z$
$X < T$



Arc-consistency & REVISE procedure

- Applying REVISE just once to all pairs of variables may not be enough…
- Consider the 3-variable constraint network
  - X and Y are initially arc-consistent
  - But later making {X,Z} arc-consistency makes {X,Y} inconsistent!

You have
3 minutes!



AC-1: a brute-force algorithm

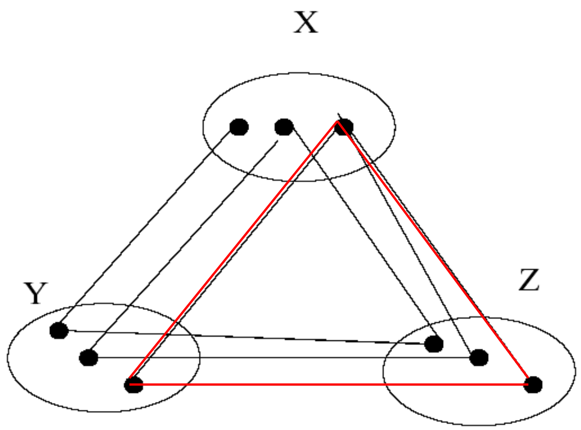
AC-1($\mathcal{R}$)

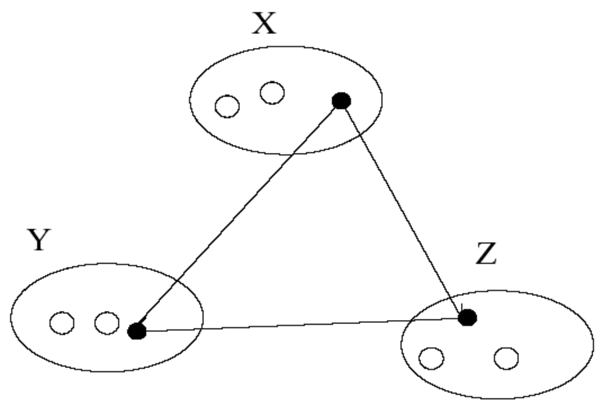
input: a network of constraints $\mathcal{R} = (X, D, C)$
output: $\mathcal{R}'$ which is the loosest arc-consistent network equivalent to $\mathcal{R}$

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

- AC-1 has complexity $O(nek^3)$ for n variables, e binary constraints and domain sizes bounded by k

AC-1: example





Another example

- Variables x,y,z with domains {1,2,3}
- Constraints x<y, y<z, z<x

You have
3 minutes!

- Revise R_{xy} in both directions… D_x={1,2} and D_y={2,3}
- Revise R_{yz}… D_y = {2} and D_z = {3}
- Revise R_{zx}… D_z = {}… the network is inconsistent!

Is AC-1 efficient?

- No need to (re)process ALL the constraints…
- Solution: Implement a queue of constraints to be processed!
    - Initially, each pair of variables in a binary constraint is placed twice in the queue (one for each ordering)
    - A pair is removed from the list after being processed
    - A pair is placed back in the queue if the domain of its second variable is modified

AC-3: an efficient algorithm

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

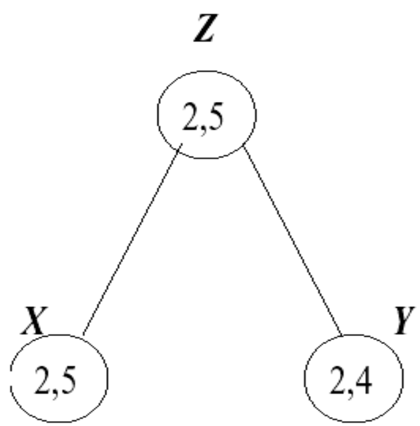
- AC-3 has *time* complexity $O(ek^3)$ for *e* binary constraints and domain sizes bounded by *k*
  - And processes each constraint at most 2k times

AC-3: example (I)

- 3-variable network X,Y,Z
- D_x={2,5}, D_y={2,4}, D_z={2,5}
- R_{zx} = {z evenly divides x}, R_{zy} = {z evenly divides y}

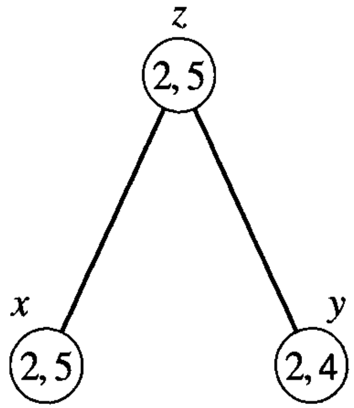
You have
5 minutes!

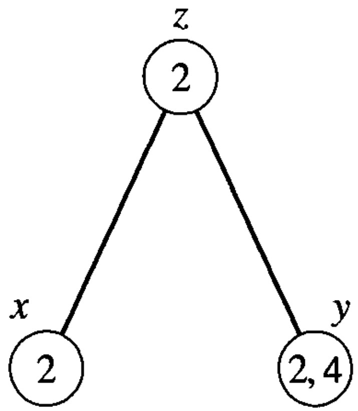
- Evenly divides = no remainder!



AC-3: example (II)

- Put {z,x}, {x,z}, {z,y}, {y,z} onto the queue
- No changes processing {z,x} and {x,z}
- Process {z,y}… Delete 5 from D_z… place {x,z} on the queue
- Process {y,z}… No changes
- Process {x,z}… Delete 5 from D_x… place {z,x} on the queue
- Process {z,x}… No changes





AC-4: time and space complexity $O(ek^3)$

- Space can be further reduced!
- Each value a_i in domain of x_i is associated with the amount of support from variable x_j, that is, the number of values in the domain of x_j that are consistent with a_i
  - A value is removed when it has no support from a neighboring variable
- Required data structures
  - List of currently unsupported variable-value pairs
  - Counter array (x_i,a_i,x_j) of supports
  - Array $S(x_j,a_j)$ that points to all values in other variables supported by $<x_j,a_j>$

AC-4: algorithm

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

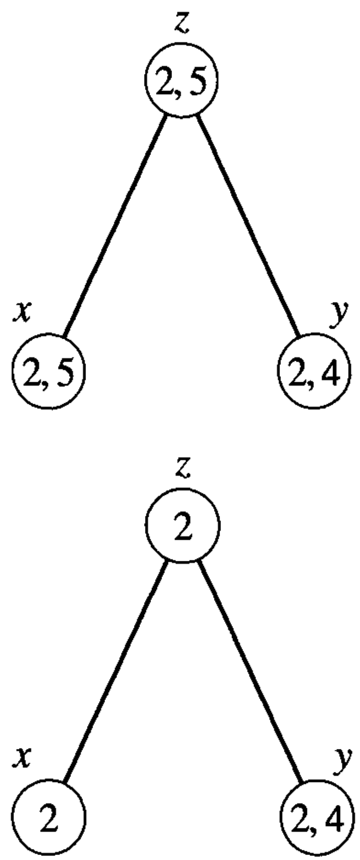
AC-4: example

- Supporting arrays

  $S_{(z,2)} = \{\langle x, 2\rangle, \langle y, 2\rangle, \langle y, 4\rangle\}, S_{(z,5)} = \{\langle x, 5\rangle\}, S_{(x,2)} = \{\langle z, 2\rangle\},$

  $S_{(x,5)} = \{\langle z, 5\rangle\}, S_{(y,2)} = \{\langle z, 2\rangle\}, S_{(y,4)} = \{\langle z, 2\rangle\}.$

- M = ∅, Counters:
  - Counter(x,2,z) = 1, Counter(x,5,z) = 1
  - Counter(z,2,x) = 1, Counter(z,5,y) = 0 (implies LIST = {(z,5)})
  - Counter(y,2,z) = 1, Counter(y,4,z) = 1
- Move (z,5) from LIST to M… Counter(x,5,z) = 0… Add (x,5) to LIST
- Move (x,5) from LIST to M...
  - The only value it supports is (z,5) that is in M
  - LIST remains empty and the process stops



Path consistency

- Arc-consistency can *sometimes* decide inconsistency
  - By discovering an empty domain
  - But it is not complete for deciding consistency… only unary and binary constraints are taken into account!
- Path consistency addresses 3-ary constraints

Arc-consistency vs path consistency

- Variables x, y, z with domains {red, blue}
- Constraints (1) $x \neq y$, (2) $y \neq z$, (3) $x \neq z$
- Arc-consistency reduces no domains…
  - Although with domain size 2, $x \neq y$ and $y \neq z$ allow inferring $x=z$
- Path consistency finds inconsistency
  - Relating 3 variables: x, y and z

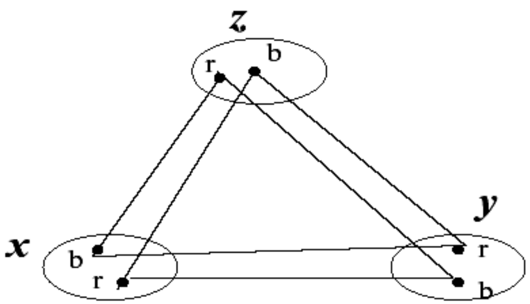
Path consistency: definition

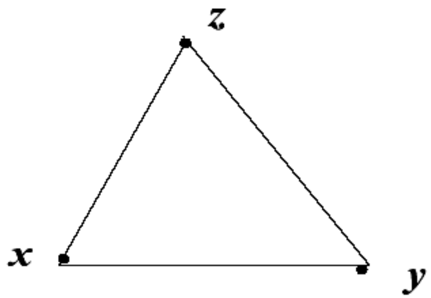
(path-consistency)

Given a constraint network $\mathcal{R} = (X, D, C)$, a two-variable set $\{x_i, x_j\}$ is path-consistent relative to variable x_k if and only if for every consistent assignment $(\langle x_i, a_i \rangle, \langle x_j, a_j \rangle)$ there is a value $a_k \in D_k$ such that the assignment $(\langle x_i, a_i \rangle, \langle x_k, a_k \rangle)$ is consistent and $(\langle x_k, a_k \rangle, \langle x_j, a_j \rangle)$ is consistent. 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 $a_k \in D_k$ such that $(a_i, a_k) \in R_{ik}$ 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. •

Path consistency: graphical picture

- Matching diagram should be extended to a triangle!
- Not possible with constraints (1) x ≠ y, (2) y ≠ z, (3) x ≠ z





REVISE-3: analogous to REVISE in AC-1

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**

- Can be summarized with $R_{xy} \leftarrow R_{xy} \cap \pi_{xy}(R_{xz} \bowtie D_z \bowtie R_{zy})$

PC-1: algorithm

```
PC-1(𝓡)
input: a network 𝓡 = (X, D, C).
output: a path consistent network equivalent to 𝓡.
repeat
    for k ← 1 to n
        for i, j ← 1 to n
            R_ij ← R_ij ∩ π_ij(R_ik ⋈ D_k ⋈ R_kj)/* (Revise − 3((i, j), k))
        endfor
    endfor
until no constraint is changed.
```

- Resembles AC-1

PC-2: algorithm

PC-2($\mathcal{R}$)

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

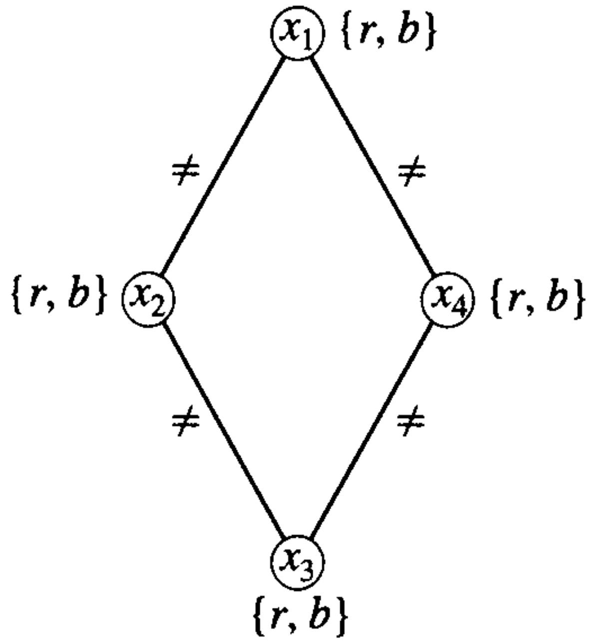
output: $\mathcal{R}'$ a path consistent network equivalent to $\mathcal{R}$.

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

- Resembles AC-3
- Maintains a queue of ordered triplets to be (re)processed

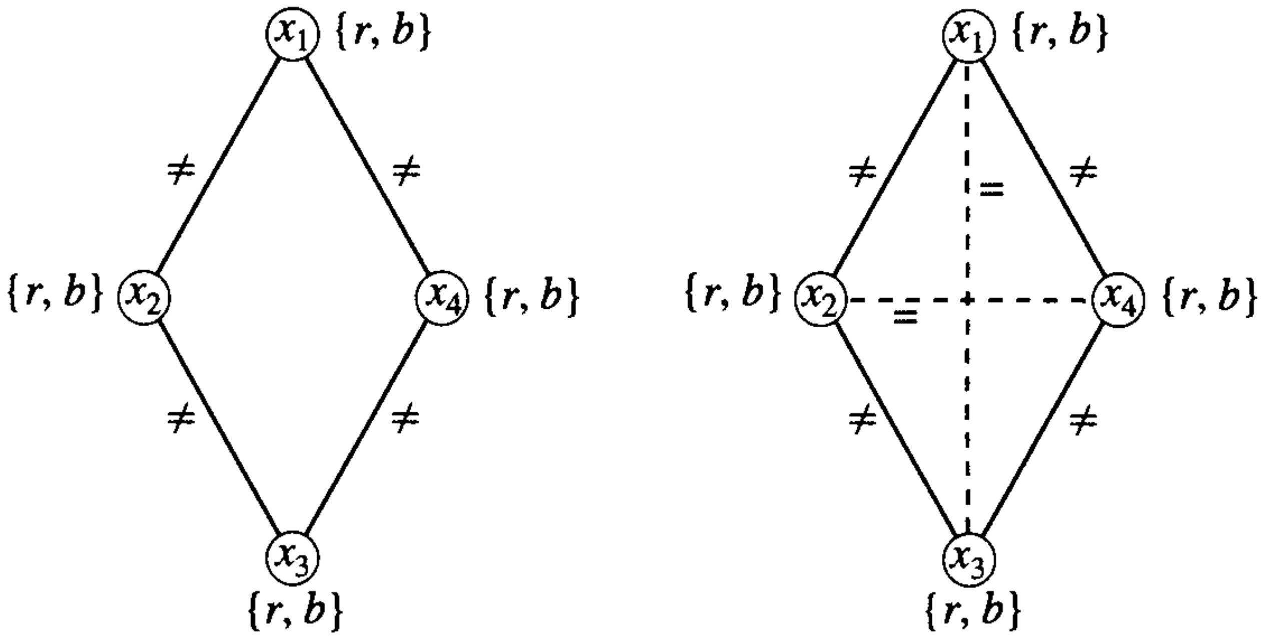
Path consistency: example (I)

- Consider the following example



You have
3 minutes!

Path consistency: example (II)



Path consistency constraint: definition

(path-consistent constraint)

A constraint R_{ij} is path-consistent, relative to the path of length m through the nodes $(i = i_0,\ i_1, \ldots, i_m = j)$, if for any pair $(a_i, a_j) \in R_{ij}$ there is a sequence of values $a_{il} \in D_{i_l}$ such that $(a_i = a_{i_0}, a_{i_1}) \in R_{i_0 i_1}$, $(a_{i_0}, a_{i_1}) \in R_{i_0 i_1}, \ldots,$ and $(a_{i_{m-1}}, a_{i_m} = a_j) \in R_{i_{m-1} i_m}$. •

Higher levels of i-consistency

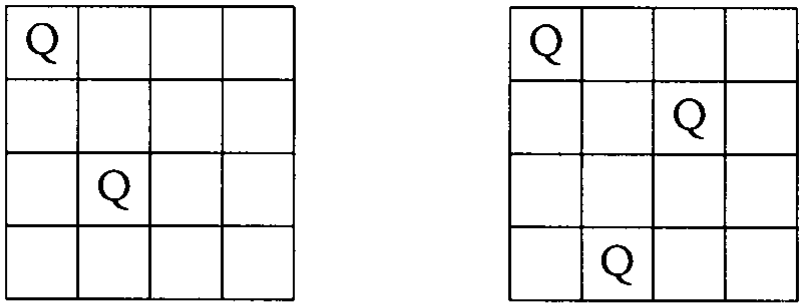
(*i*-consistency, global consistency)

Given a general network of constraints $\mathcal{R} = (X, D, C)$, a relation $R_S \in C$ where $|S| = i - 1$ is i-consistent relative to a variable y not in S iff for every $t \in R_S$, there exists a value $a \in D_y$, such that (t, a) is consistent. 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*. •

i-consistency: example

- 3-consistency? 4-consistency?

You have
3 minutes!



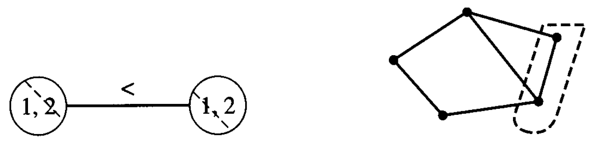
(a) (b)

- (a) is not 3-consistent; (b) is not 4-consistent

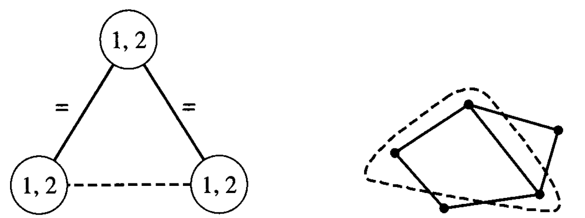
Observation: queens are
placed sequentially by column

Summary

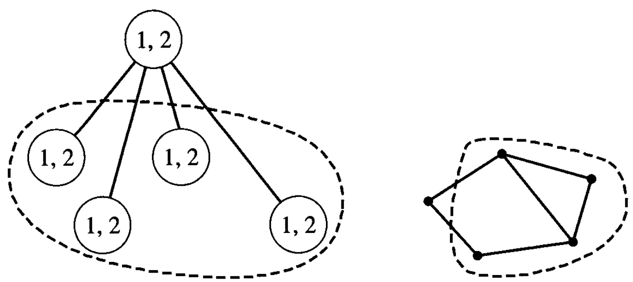
- (a) arc-consistency
- (b) path consistency
- (c) i-consistency



(a)



(b)



(c)


That's all Folks!