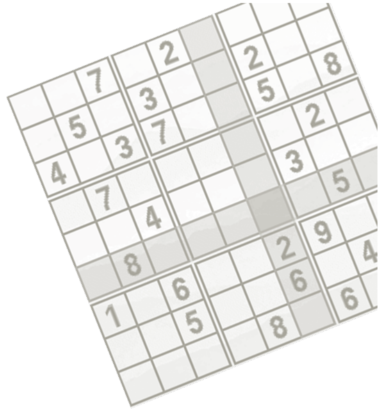


Constraint Programming

Roman Barták
Department of Theoretical Computer Science and Mathematical Logic

Consistency Techniques: Path Consistency

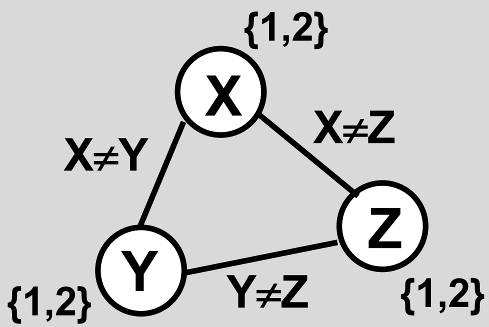
Arc consistency:

- **The arc** *(V_i,V_j)* is **arc consistent** iff for each value *x* from the domain D_i there exists a value *y* in the domain D_j such that the assignment $V_i = x$ a $V_j = y$ satisfies all the binary constraints on V_i, V_j.

Note: The concept of arc consistency is directional, i.e., arc consistency of (V_i,V_j) does not guarantee consistency of (V_j,V_i).

- **CSP** is **arc consistent** iff every arc (V_i,V_j) is arc consistent (in both directions).

Example:



CSP is arc consistent but there is no solution!

Sometimes **AC directly provides a solution**.

any domain is empty $\rightarrow$ no solution exists

all domains are singleton $\rightarrow$ this is a solution

In general, AC **decreases the search space.**

How to strengthen the consistency level?

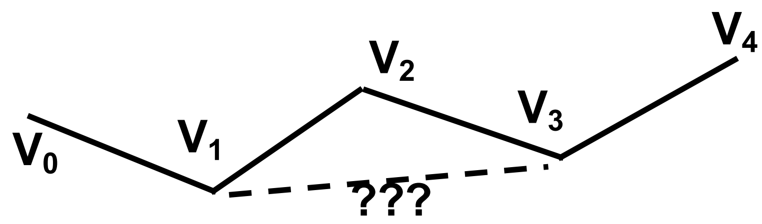
More constraints are assumed **together**!

Definition:

- **The path** $(V_0,V_1,\ldots, V_m)$ is **path consistent** iff for every pair of values $x \in D_0$ a $y \in D_m$ satisfying all the binary constraints on V_0,V_m there exists an assignment of variables $V_1,\ldots,V_{m-1}$ such that all the binary constraints between the neighbouring variables V_i,V_{i+1} are satisfied.
- CSP is **path consistent** iff every path is consistent.

Beware:

- only the **constraints between the neighboring variables** must be satisfied



PC and paths of length two

It is not very practical to make all paths consistent.
Fortunately, it is enough to make path of length 2 consistent!

Theorem: CSP is PC if and only if all paths of length 2 are PC.

Proof:

1) PC $\Rightarrow$ paths of length 2 are PC

2) All paths of length 2 are PC $\Rightarrow$ $\forall$N paths of length N are PC $\Rightarrow$ PC

induction using the path length

a) N=2 trivially true

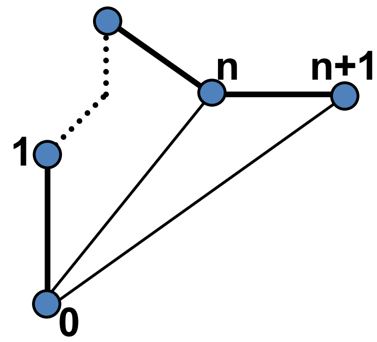
b) N+1 (assuming that the theorem holds for N)

i) take any N+2 nodes $V_0,V_1,\ldots, V_{n+1}$

ii) take any two consistent values $x_0 \in D_0$ a $x_{n+1} \in D_{n+1}$

iii) using a) find the value $x_n \in D_n$ st. $P_{0,n}$ and $P_{n,n+1}$ holds

iv) using induction find the other values $V_0,V_1,\ldots, V_n$



Does PC cover AC (if CSP PC, then is it also AC)?

- arc (i, j) is consistent (AC), if the path (i,j,i) is consistent (PC)
- PC implies AC

Is PC stronger than AC (is there any CSP whish is AC but not PC)?

Example: X in {1,2}, Y in {1,2}, Z in {1,2}, $X \neq Z$, $X \neq Y$, $Y \neq Z$

- It is AC, but not PC (X=1, Z=2 is not consistent over X,Y,Z)

AC removes inconsistent values from the domains.

What is done by PC algorithms?

- **PC removes pairs of inconsistent values**
- PC makes all relations explicit ($A<B, B<C \Rightarrow A+1<C$)
- unary constraint = domain of the variable

PC algorithms will remove pairs of values

↳ we need to represent the constraints explicitly

Binary constraints = {0,1}-matrix

0 – pair of values is inconsistent

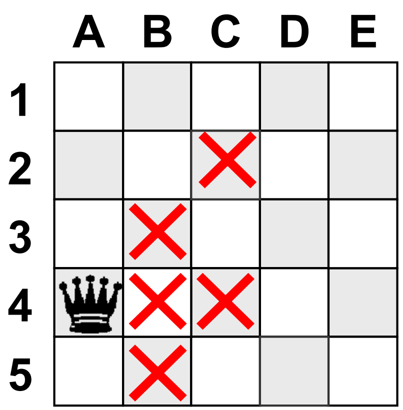
1 – pair of values is consistent

Example (5-queens problem)

constraint between queens *i* and *j*: r(i) $\neq$ r(j) & |i-j| $\neq$ |r(i)-r(j)|

Matrix representation for constraint A(1) - B(2)

0 0 1 1 1
0 0 0 1 1
1 0 0 0 1
1 1 0 0 0
1 1 1 0 0



Matrix representation for constraint A(1) - C(3)

0 1 0 1 1
1 0 1 0 1
0 1 0 1 0
1 0 1 0 1
1 1 0 1 0

Constraint intersection R_{ij} & $R`_{ij}$

bitwise AND

A<B	**&**	***A≥B-1***		***→ B-1≤A<B***
011		**110**		**010**
001	**&**	**111**	**=**	**001**
000		**111**		**000**

Constraint join $R_{ik} * R_{kj} \rightarrow R_{ij}$

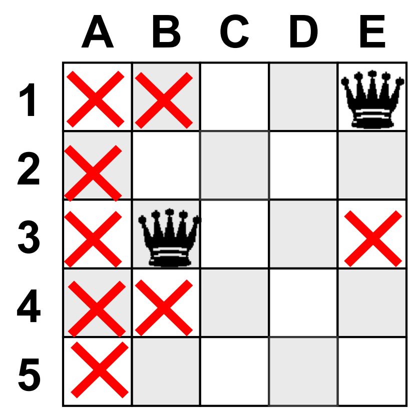
Binary matrix multiplication

A<B	*****	***B<C →***		***A<C-1***
011		**011**		**001**
001	*****	**001**	**=**	**000**
000		**000**		**000**

Induced constraint is intersected with the original constraint

R_{ij} & $(R_{ik} * R_{kj}) \rightarrow R_{ij}$

$\mathbf{R_{25}}$	**&**	$\mathbf{(R_{21}}$	*****	$\mathbf{R_{15})}$	**→**	$\mathbf{R_{25}}$
01101		**00111**		**01110**		**01101**
10110		**00011**		**10111**		**10110**
11011	**&**	**10001**	*****	**11011**	**=**	**01010**
01101		**11000**		**11101**		**01101**
10110		**11100**		**01110**		**10110**



Notes:

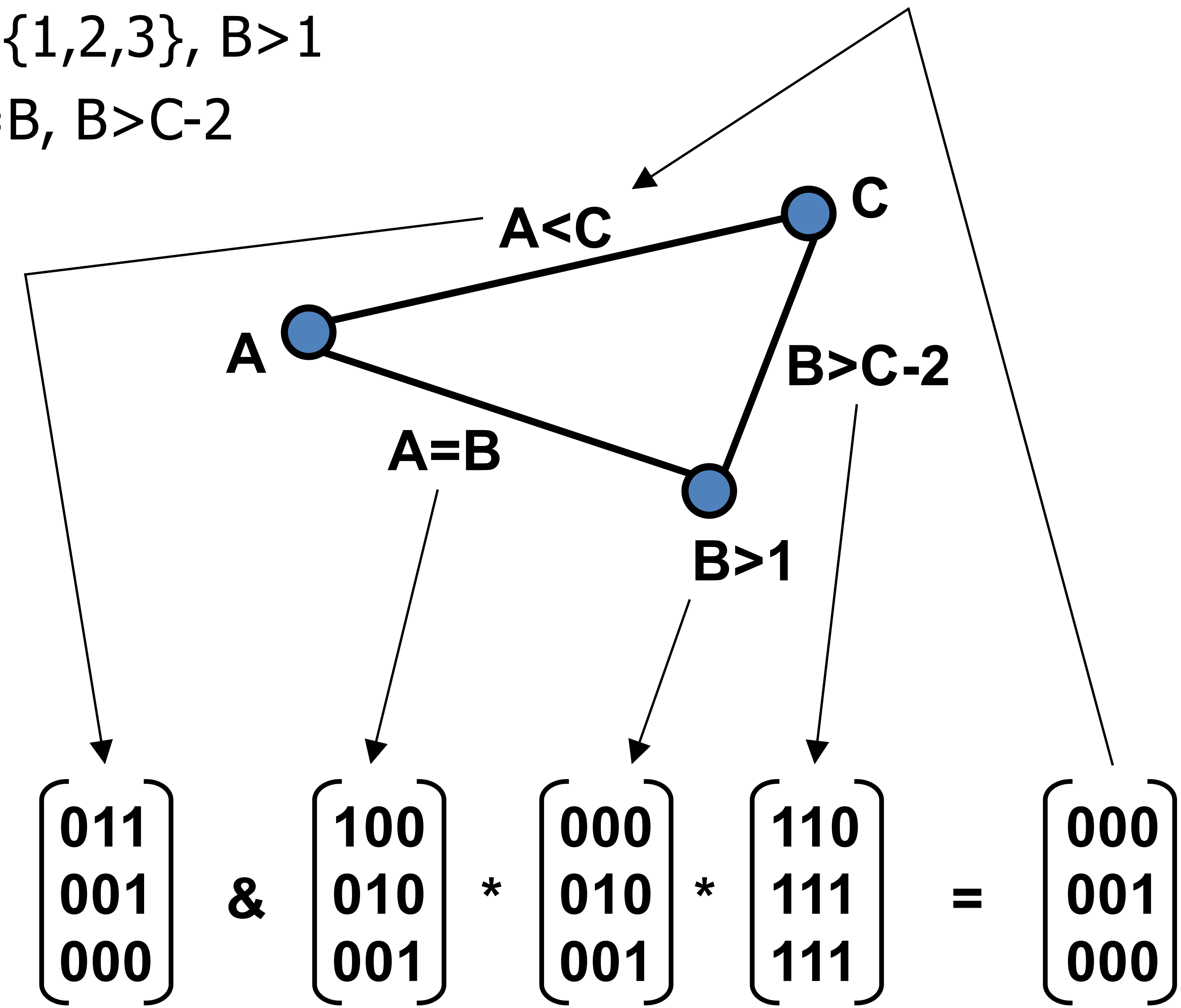
$R_{ij} = R^T_{ji}$, R_{ii} is a diagonal matrix representing the domain of variable

REVISE((i,j)) from the AC algorithms is $R_{ii} \leftarrow R_{ii}$ & $(R_{ij} * R_{jj} * R_{ji})$

Composing constraints

A,B,C in {1,2,3}, B>1

A<C, A=B, B>C-2



Mackworth (1977)

Algorithm PC-1

How to make the path (i,k,j) consistent?

$R_{ij} \leftarrow R_{ij}$ & $(R_{ik} * R_{kk} * R_{kj})$

How to make a CSP path consistent?

Repeated revisions of paths (of length 2) while any domain changes.

procedure PC-1(Vars,Constraints)
    $n \leftarrow |Vars|$, $Y^n \leftarrow$ Constraints
    repeat
        $Y^0 \leftarrow Y^n$
        for $k = 1$ to n **do**
            for $i = 1$ to n **do**
                for $j = 1$ to n **do**
                    $Y^k_{ij} \leftarrow Y^{k-1}_{ij}$ & $(Y^{k-1}_{ik} * Y^{k-1}_{kk} * Y^{k-1}_{kj})$
    until $Y^n = Y^0$
    Constraints $\leftarrow Y^0$
end PC-1

If we use
$Y^k_{ii} \leftarrow Y^{k-1}_{ii}$ & $(Y^{k-1}_{ik} * Y^{k-1}_{kk} * Y^{k-1}_{ki})$
then we get AC-1

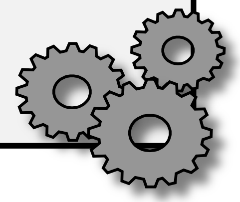
Is there any inefficiency in PC-1?

- just a few „bits“
  - it is not necessary to keep all copies of Y^k one copy and a bit indicating the change is enough
  - some operations produce no modification ($Y^k_{kk} = Y^{k-1}_{kk}$)
  - half of the operations can be removed ($Y_{ji} = Y^T_{ij}$)
- **the grand problem**
  - after domain change all the paths are re-revised but it is enough to revise just the influenced paths

Algorithm of path revision

procedure REVISE_PATH((i,k,j))
    $Z \leftarrow Y_{ij}$ & $(Y_{ik} * Y_{kk} * Y_{kj})$
    if $Z=Y_{ij}$ **then return** false
    $Y_{ij} \leftarrow Z$
    return true
end REVISE_PATH

If the domain is pruned then the influenced paths will be revised.



Because $Y_{ji} = Y^T_{ij}$ it is enough to revise only the paths (i,k,j) where $i \leq j$.
Let the domain of the constraint (i,j) be changed when revising (i,k,j):

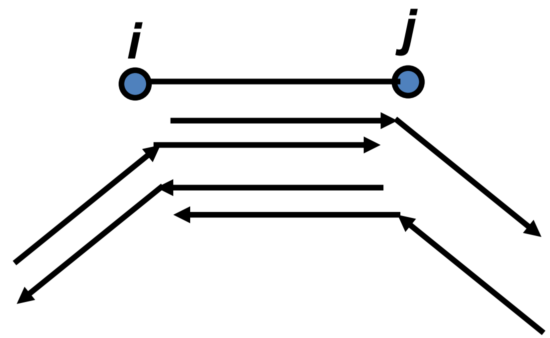
Situation a: i<j

all the paths containing (i,j) or (j,i) must be re-revised

but the paths (i,j,j), (i,i,j) are not revised again (no change)

$$
\begin{aligned}
S_a = \; & \{(i,j,m) \mid i \leq m \leq n \;\&\; m \neq j\} \\
\cup \; & \{(m,i,j) \mid 1 \leq m \leq j \;\&\; m \neq i\} \\
\cup \; & \{(j,i,m) \mid j < m \leq n\} \\
\cup \; & \{(m,j,i) \mid 1 \leq m < i\}
\end{aligned}
$$

$| S_a | = 2n-2$



Situation b: i=j

all the paths containing i in the middle of the path are re-revised

but the paths (i,i,i) and (k,i,k) are not revised again

$S_b = \{(p,i,m) \mid 1 \leq m \leq n \;\&\; 1 \leq p \leq m\} - \{(i,i,i),(k,i,k)\}$

$| S_b | = n*(n-1)/2 - 2$

Mackworth (1977)

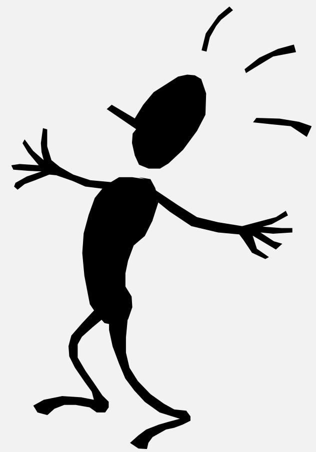
Algorithm PC-2

Paths in one direction only (attention, this is not DPC!)

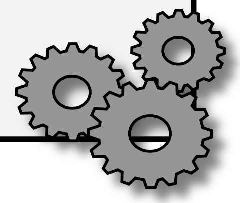
After every revision, the **affected paths are re-revised**

Algorithm PC-2

```
procedure PC-2(G)
    n ← |nodes(G)|
    Q ← {(i,k,j) | 1 ≤ i ≤ j ≤ n & i≠k & j≠k}
    while Q non empty do
        select and delete (i,k,j) from Q
        if REVISE_PATH((i,k,j)) then
            Q ← Q ∪ RELATED_PATHS((i,k,j))
    end while
end PC-2
```



```
procedure RELATED_PATHS((i,k,j))
    if i<j then return S_a else return S_b
end RELATED_PATHS
```



- **PC-3 (Mohr, Henderson 1986)**
  - based on computing supports for a value (like AC-4)
    - If pair (*a*,*b*) at arc (*i*,*j*) is not supported by another variable, then *a* is removed from D_i and *b* is removed from D_j.
  - **this algorithm is not sound!**
- **PC-4 (Han, Lee 1988)**
  - correction of the PC-3 algorithm
  - based on computing supports of pairs (*b*,*c*) at arc (*i*,*j*)
- **PC-5 (Singh 1995)**
  - uses the ideas behind AC-6
  - only one support is kept and a new support is looked for when the current support is lost

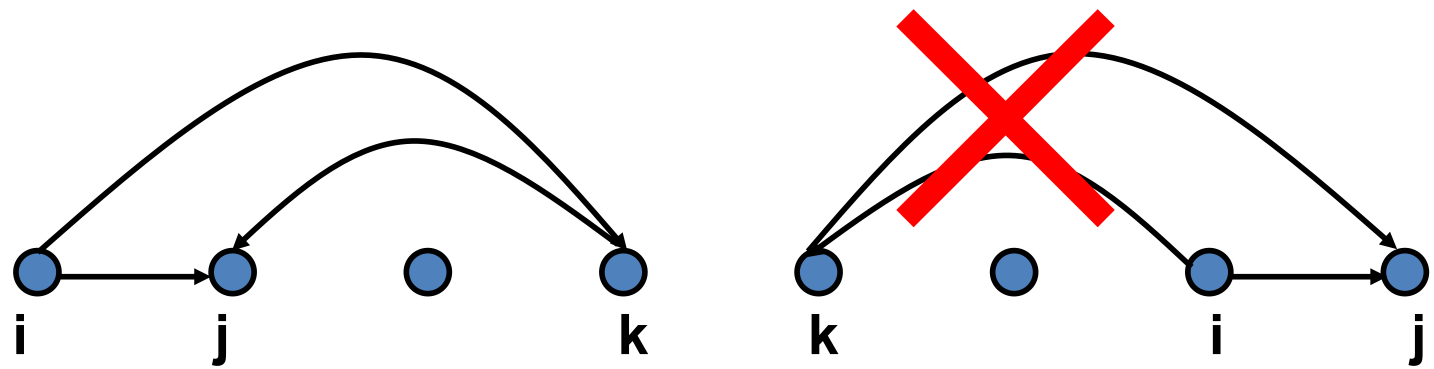
Similarly to AC we can decrease complexity of PC by assuming paths in one direction only.

Definition:

CSP is **directional path consistent** for a given order of variables if and only if all paths (i,k,j) st. i ≤ k and j ≤ k are path consistent.

Notes:

- Notice that requirements i ≤ k and j ≤ k are different from i ≤ j that is used to break symmetries of paths!
- We can also use the requirement i ≤ j in DPC algorithms.



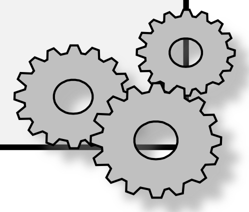
Algorithm DPC-1

Similarly to DAC-1 we can explore each path exactly once (by going in the reverse order).

We can remove some constraint checks via symmetry ($i \leq j$).

Algorithm DPC-1

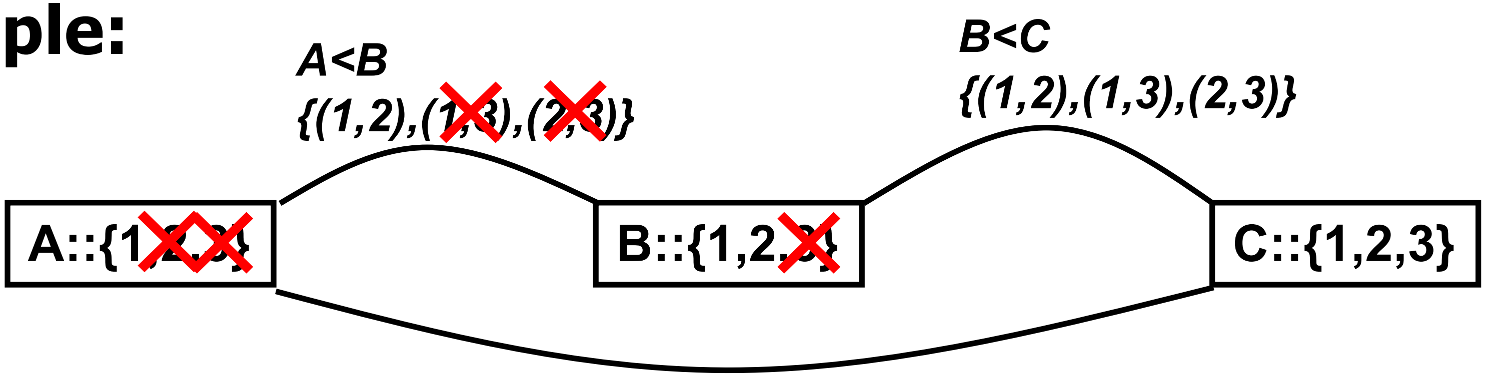
```
procedure DPC-1(Vars,Constraints)
    n ← |Vars|, E ← { (i,j) | i<j & C_{i,j}∈Constraints}
    for k = n to 1 by -1 do
        for i = 1 to k do
            for j = i to k do
                if (i,k)∈E & (j,k)∈E then
                    C_ij ← C_ij & (C_ik * C_kk * C_kj)
                    E ← E ∪ {(i,j)}
            end for
        end for
    end for
end DPC-1
```



Clearly PC implies DPC.

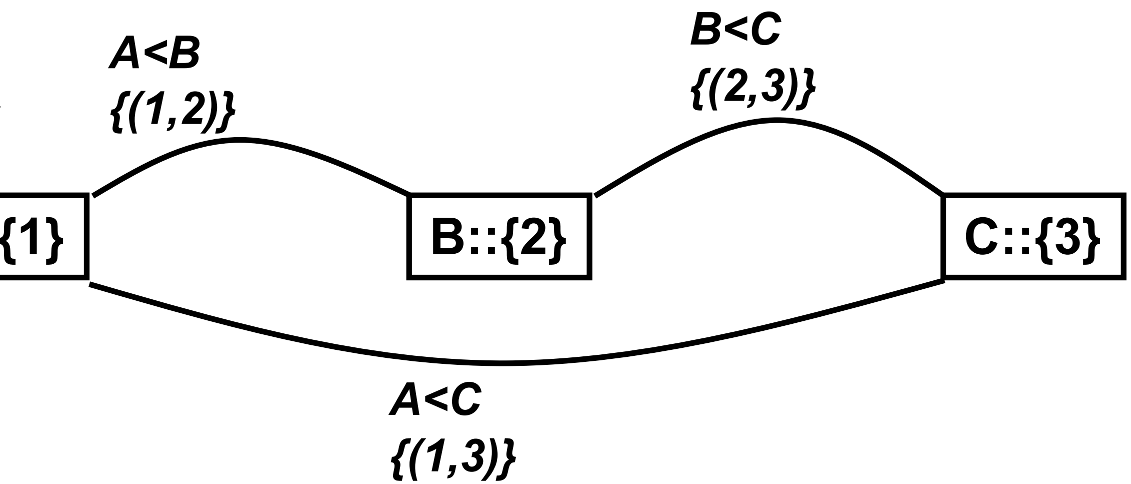
What about the other direction (does DPC imply PC)?

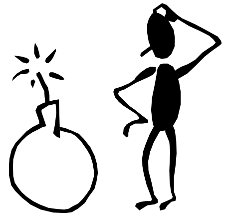
Example:



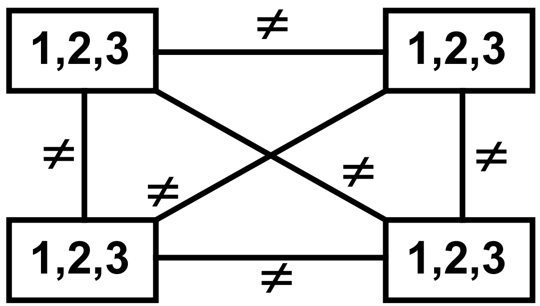
CSP is DPC, but not PC!
It is even not AC.

PC and AC network





- **memory consumption**
  - because PC eliminates pairs of values, we need to keep all the compatible pairs extensionally, e.g. using {0,1}-matrix
- **bad ratio strength/efficiency**
  - PC removes more (or same) inconsistencies than AC, but the strength/efficiency ratio is much worse than for AC
- **modifies the constraint network**
  - PC adds redundant arcs (constraints) and thus it changes connectivity of the constraint network
  - this complicates using heuristics derived from the structure of the constraint network (like density, graph width etc.)
- **PC is still not a complete technique**
  - A,B,C,D in {1,2,3}
    $A \neq B$, $A \neq C$, $A \neq D$, $B \neq C$, $B \neq D$, $C \neq D$
    is PC but has no solution

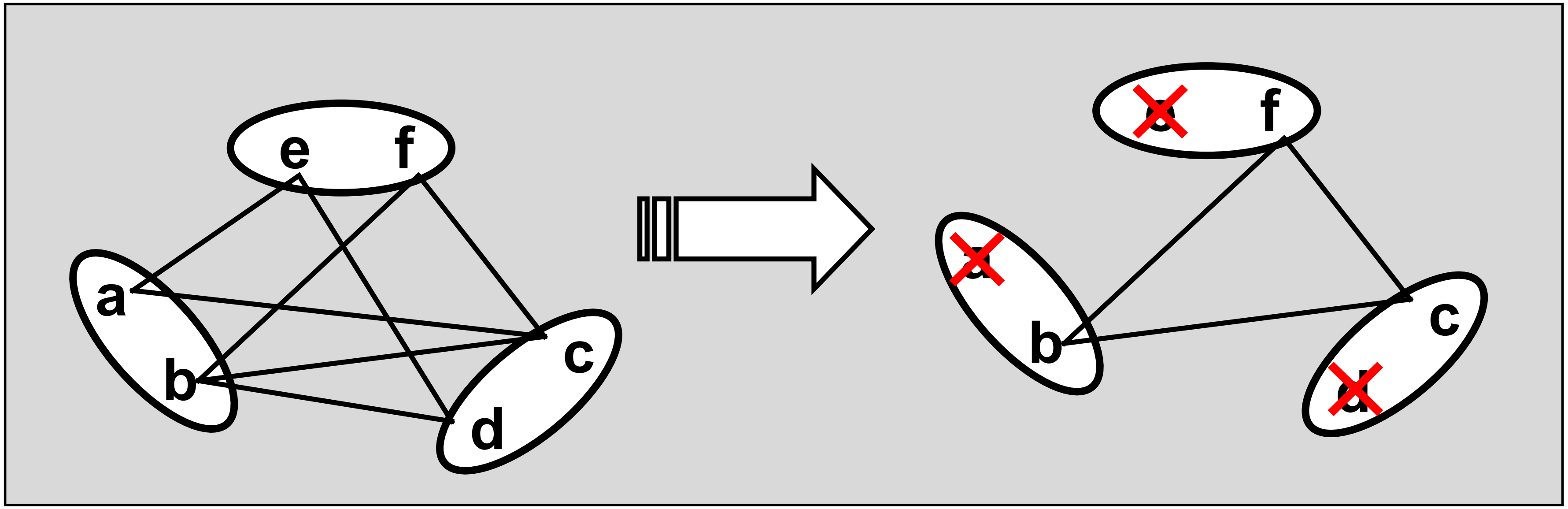


Can we make an algorithm:

- that is stronger than AC,
- without the drawbacks of PC (memory consumption, changing the constraint network)?

We can do the PC consistency check only when there is a chance for filtering some value out!

Example:



Restricted path consistency

PC is checked only when filtering out a value pair means filtering some of the values out of the domain.

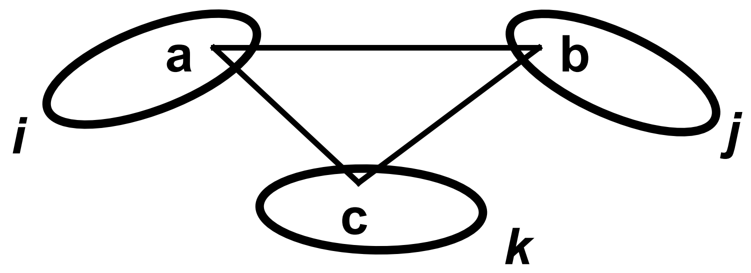
How do we recognize such a situation?

- If a given value pair is the only support for one of the values.

Definition:

Node *i* is **restricted path consistent** if any only if:

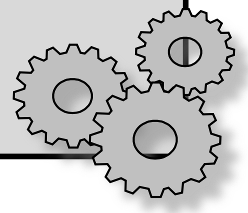
- each arc going from *i* is arc consistent
- for each **a** $\in D_i$ it holds that
  if **b** is the only support for **a** in the node *j* then for each node *k* (connected to both *i* and *j*) we can find a value **c** such that the pairs (**a**,**c**) and (**b**,**c**) are consistent with respective constraints (PC).



Algorithm RPC - initialisation

Based on AC-4: a support counter + a queue for PC

```
procedure INITIALIZE(G)
    Q_AC ← {} , Q_PC ← {} , S ← {}                    % preparing data structures
    for each (i,j)∈arcs(G) do
        for each a∈D_i do
            total ← 0
            for each b∈D_j do
                if (a,b) is consistent according to the constraint C_i,j then
                        total ← total + 1,  S_j,b ← S_j,b ∪ {<i,a>}
            end for
            counter[(i,j),a] ← total
            if counter[(i,j),a] = 0 then
                Q_AC ← Q_AC ∪ {<i,a>},  delete a from D_i
            else if counter[(i,j),a] = 1 then
                    for each k such that (i,k)∈arcs(G) & (k,j)∈arcs(G) do
                        Q_PC ← Q_PC ∪ {(<i,a>,j,k)}
            end if
        end for
    end for
    return (Q_AC, Q_PC)
end INITIALIZE
```



procedure PRUNE(Q_{AC}, Q_{PC})
    while Q_{AC} non empty **do**
        select and delete any pair <j,b> from Q_{AC}
        for each <i,a> from $S_{j,b}$ **do**
            counter[(i,j),a] $\leftarrow$ counter[(i,j),a] - 1
            if counter[(i,j),a] = 0 & "a" is still in D_i **then**
                    delete "a" from D_i
                    $Q_{AC} \leftarrow Q_{AC} \cup \{\langle i,a \rangle\}$
            else if counter[(i,j),a] = 1 **then**
                    for each k such that (i,k)$\in$arcs(G) & (k,j)$\in$arcs(G) **do**
                        $Q_{PC} \leftarrow Q_{PC} \cup \{(\langle i,a \rangle,j,k)\}$
                else
                    for each k such that (i,k)$\in$arcs(G) & (k,j)$\in$arcs(G) **do**
                      if counter[(i,k),a] = 1 **then**
                        $Q_{PC} \leftarrow Q_{PC} \cup \{(\langle i,a \rangle,k,j)\}$
            end if
        end for
    end while
    return Q_{PC}
end PRUNE

Algorithm RPC

First, make the problem AC and then test PC for selected paths and restore AC if necessary.

```
procedure RPC(G)
    (Q_AC, Q_PC) ← INITIALIZE(G)
    Q_PC ← PRUNE(Q_AC, Q_PC)                      % first run of AC
    while Q_PC non empty do
        select and delete any triple (<i,a>,j,k) from Q_PC
        if a∈D_i then
            {<j,b>} ← {<j,x> ∈ S_ia | x∈D_j }     % the only support for a
            if {<k,c> ∈ S_ia ∩ S_jb | c ∈ D_k } = ∅ then
                counter[(i,j),a] ← 0
                delete "a" from D_i
                Q_PC ← PRUNE({<i,a>}, Q_PC)       % repeat AC
            end if
        end if
    end while
end RPC
```

Department of Theoretical Computer Science and Mathematical Logic
bartak@ktiml.mff.cuni.cz