Foundations of constraint satisfaction

4

Roman Barták
Charles University in Prague

bartak@ktiml.mff.cuni.cz
http://ktiml.mff.cuni.cz/~bartak

Path consistency (PC)

How to strengthen the consistency level?

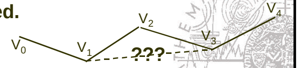
More constraints are assumed together!

Definition:

- *The path* (V_0,V_1,…, V_m) is *path consistent* iff for every pair of values x Î D_0 a y Î D_m satisfying all the binary constraints on V_0,V_m there exists an assignment of variables V_1,…,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.

Attention!

Path consistency does not guarantee that all the constraints among the variables on the path are satisfied; only the constraints between the neighbouring variables must be satisfied.



PC and paths of length 2 (Montanari)

It is not very practical to ensure consistency of all paths

fortunately, only the paths of length 2 can be explored!

Theorem: CSP is PC iff every path of length 2 is PC.

Proof:

1) PC Þ paths of length 2 are PC

2) (paths of length 2 are PC Þ "N paths of length N are PC) Þ PC

induction using the path length

a) N=2 visibly satisfied

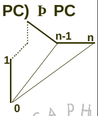
b) N+1 (proposition already holds for N)

i) take arbitrary N+1 vertices V_0,V_1,…, V_n

ii) take arbitrary pair of compatible values x_0 Î D_0 a x_n Î D_n

iii) from a) we can find x_{n-1} Î D_{n-1} s.t. constraints $C_{0,n-1}$ a $C_{n-1,n}$ hold

iv) from the induction we can find the values for V_0,V_1,…, V_{n-1}



Relation between PC and AC

Does PC subsumes AC (i.e. if CSP is PC, is it AC as well)?

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

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

Example: X in {1,2}, Y in {1,2}, Z in {1,2}, X¹Z, X¹Y, Y¹Z

it is AC, but not PC (X=1, Z=2 cannot be extended to X,Y,Z)

AC removes incompatible values from the domains, what will be done in PC?

- PC removes pairs of values
- PC makes constraints explicit (A<B,B<C Þ A+1<C)
- a unary constraint = a variable's domain

A matrix representation of constraints

In PC we need to exclude the pairs of values

the constraints must be represented in explicit form

Binary constraint = {0,1}-matrix

0 - the values are incompatible

1 - the values are compatible

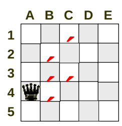
Example:

5-queens problem

the constraint between queens *i, j*: r(i)¹r(j) & |i-j| ¹ |r(i)-r(j)|

a matrix for queens 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
```



a matrix for queens 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
```


Operations over the constraints

Intersection R_{ij} & R'_{ij}

bitwise AND

A<B	&	*A³B-1*	®	*B-1£A<B*
011		110		010
001	&	111	=	001
000		111		000

Composition R_{ik} * R_{kj} ® R_{ik}

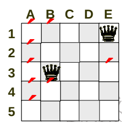
binary matrix multiplication

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

The induced constraint is joined with the original constraint

R_{ij} & (R_{ik} * R_{kj}) ® R_{ij}

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



Notes:

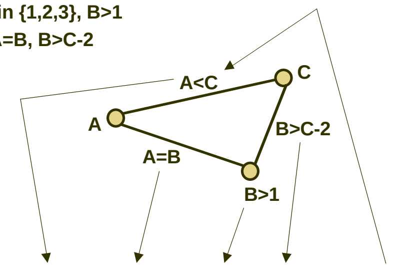
$R_{ij} = R^T_{ji}$, R_{ii} is a diagonal matrix representing the domain

REVISE((i,j)) from AC is equivalent to R_{ii} ¬ R_{ii} & (R_{ij} * R_{jj} * R_{ji})

Composing the constraints on the path

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

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



$$\begin{pmatrix}011\\001\\000\end{pmatrix} \& \begin{pmatrix}100\\010\\001\end{pmatrix} * \begin{pmatrix}000\\010\\001\end{pmatrix} * \begin{pmatrix}110\\111\\111\end{pmatrix} = \begin{pmatrix}000\\001\\000\end{pmatrix}$$

Algorithm PC-1 (Mackworth 1977)

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

R_{ij} ¬ R_{ij} & (R_{ik} * R_{kk} * R_{kj})

How to make a CSP consistent?

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

Algorithm PC-1

```
procedure PC-1(Vars,Constraints)
    n ¬ |Vars|, Yⁿ ¬ Constraints
    repeat
        Y⁰ ¬ Yⁿ
        for k = 1 to n do
            for i = 1 to n do
                for j = 1 to n do
                    Yᵏ_ij ¬ Yᵏ⁻¹_ij & (Yᵏ⁻¹_ik * Yᵏ⁻¹_kk * Yᵏ⁻¹_kj)
    until Yⁿ=Y⁰
    Constraints ¬ Y⁰
end PC-1
```

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

How to improve PC-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 it is enough to revise just the influenced paths

Algorithm of path revision

```
procedure REVISE_PATH((i,k,j))
    Z ¬ Y_ij & (Y_ik * Y_kk * Y_kj)
    if Z=Y_ij then return false
    Y_ij ¬ Z
    return true
end REVISE_PATH
```

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

Which paths are influenced by the revision?

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

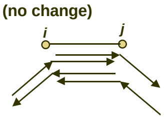
Situation a: i<j

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

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

S_a = {(i,j,m) | i £ m £ n & m¹j}
È {(m,i,j) | 1 £ m £ j & m¹i}
È {(j,i,m) | j < m £ n}
È {(m,j,i) | 1 £ m < i}

| S_a | = 2n-2



Situation b: i=j

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

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

S_b = {(p,i,m) | 1 £ m £ n & 1 £ p £ m} - {(i,i,i),(k,i,k)}

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

Algorithm PC-2 (Mackworth 1977)

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 & i¹k & j¹k}
    while Q non empty do
        select and delete (i,k,j) from Q
        if REVISE_PATH((i,k,j)) then
            Q ¬ Q È 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
```


Other path consistency algorithms

PC-3 (Mohr, Henderson 1986)

- based on computing supports for a value (like AC-4)
- this algorithm is not sound!
  - If the pair (*a*,*b*) at the arc (i,j) is not supported by another variable, then *a* is removed from D_i and *b* is removed from D_j.

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

Drawbacks of path consistency

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

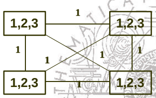
– PC removes more (or same) inconsistencies than AC, but the strength/efficiency ratio is much worse than for AC

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 tightness, graph width etc.)

PC is still not a complete technique

– A,B,C,D in {1,2,3}
A¹B, A¹C, A¹D, B¹C, B¹D, C¹D
is PC but has not solution



Half way between AC and PC

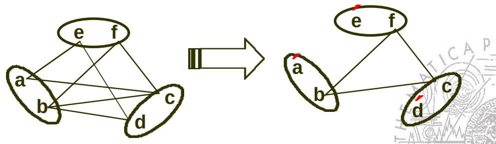
Can we make an algorithm:

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

Restricted path consistency (Berlandier 1995)

based on AC-4 (uses the support sets)

as soon as a value has only one support in another variable, PC is evoked for this pair of values



k-consistency

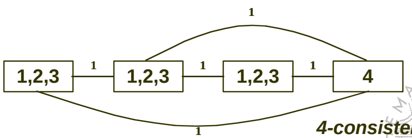
Is there a common formalism for AC and PC?

AC: a value is extended to another variable

PC: a pair of values is extended to another variable

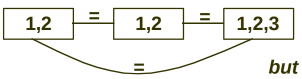
… we can continue

Definition: CSP is k-consistent iff any consistent valuation of (k-1) different variables can be extended to a consistent valuation of one additional variable.



4-consistent graph

Strong k-consistency



3-consistent graph

but not 2-consistent graph!

Definition: CSP is strongly k-consistent iff it is j-consistent for every j£k.

Visibly: strong k-consistency Þ k-consistency

Moreover: strong k-consistency Þ j-consistency "j£k

In general: Ø k-consistency Þ strong k-consistency

NC = strong 1-consistency = 1-consistency

AC = (strong) 2-consistency

PC = (strong) 3-consistency

sometimes we call NC+AC+PC together *strong path consistency*

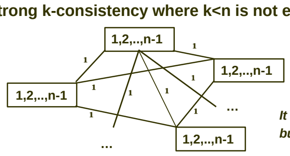
What *k-consistency* is enough?

Assume that the number of vertices is *n*. What level of consistency do we need to find out the solution?

Strong n-consistency for graphs with *n* vertices!

n-consistency is not enough - see the previous example

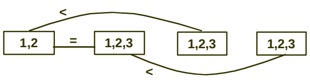
strong k-consistency where k<n is not enough as well



graph with n vertices
domains 1..(n-1)

It is strongly k-consistent for k<n
but it has no solution

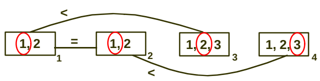
And what about this graph?



(D)AC is enough!
Because this a tree..

Backtrack-free search

Definition: CSP is solved using backtrack-free search if for some order of variables we can find a value for each variable compatible with the values of already assigned variables.



How to find out a sufficient consistency level for a given graph?

Some observations:

- variable must be compatible with all the "former" variables
  i.e., across the „backward" edges
- for *k* „backward" edges we need (*k*+1)-consistency
- let *m* be the maximum of backward edges for all the vertices, then strong (*m*+1)-consistency is enough
- the number of backward edges is different for different variable order
- of course, the order minimising *m* is looked for

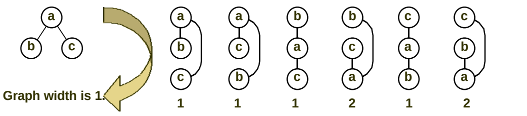
Graph width

Ordered graph is a graph with a given total order of vertices.

Vertex width in the ordered graph is the number of edges going back from this vertex.

Width of the ordered graph is maximum among the width of vertices.

Graph width is the maximum among the widths of its ordered graphs.




```
procedure MinWidthOrdering((V,E))
    Q ← {}
    while V not empty do
        N ← select and delete node with the smallest #edges from (V,E)
        enqueue N to Q
    return Q
end MinWidthOrdering
```


Graph width and consistency level

Theorem: Let w be the width of the constraint graph. If the constraint graph is strongly k-consistent for any k>w then there exists an order of variables giving backtrack-free solution.

Proof:

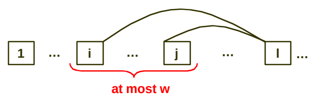
w is a graph width, i.e., there is some ordered graph of this width

thus the max. number of backward edges for each vertex is w

let us assign the variables in the order given by this ordered graph

now, if the variable is being labelled:

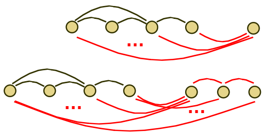
- we must find a value compatible with the labelled variables connected with the current variable
- let there is m such variables, then m ≤ w
- the graph is (m+1)-consistent, thus a compatible value must exist



(i,j)-consistency

k-consistency extends instantiation of (k-1) variables to a new variable, we remove (k-1)- tuples that cannot be extended to another variable.

We can do even more!



Definition: CSP is (i,j)-consistent iff every consistent instantiation of *i* variables can be extended to a consistent instantiation of any *j* additional variables.

CSP is strongly (i,j)-consistent, iff it is (k,j)-consistent for every k≤i.

k-consistency	= (k-1,1) consistency
AC	= (1,1) consistency
PC	= (2,1) consistency

Inverse consistencies

Worst case time and space complexity of (i,j)-consistency is exponential in *i*, moreover we need to record forbidden *i*-tuples extensionally (see PC).

What about keeping *i*=1 and increasing *j*?

We already have such an example:

RPC is (1,1)-consistency and sometimes (1,2)-consistency

Definition: (1,k-1)-consistency is called k-inverse consistency.

We remove values from the domain that cannot be consistently extended to additional (k-1) variables.

Inverse path consistency (PIC) = (1,2)-consistency

Neighbourhood inverse consistency (NIC) (Freuder , Elfe 1996)

We remove values of *v* that cannot be consistently extended to the set of variables directly linked to *v*.

Singleton consistencies

Can we strengthen any consistency technique?

YES! Let's assign a value and make the rest of the problem consistent.

Definition: CSP P is singleton A-consistent for some notion of A-consistency iff for every value *h* of any variable *X* the problem $P_{|X=h|}$ is A-consistent.

Features:

+ we remove only values from variable's domain - like NIC and RPC
+ easy implementation (meta-programming)
- not so good time complexity (be careful when using SC)

1) singleton A-consistency ≥ A-consistency

2) A-consistency ≥ B-consistency ⇒ singleton A-consistency ≥ singleton B-consistency

3) singleton (i,j)-consistency > (i,j+1)-consistency (SAC>PIC)

4) strong (i+1,j)-consistency > singleton (i,j)-consistency (PC>SAC)

Consistency techniques at glance

NC = 1- consistency

AC = 2- consistency = (1,1)- consistency

PC = 3- consistency = (2,1)- consistency

PIC = (1,2)- consistency