CSP: Solving by Inference

Pedro Meseguer
IIIA-CSIC
Bellaterra, Spain

Overview

Incomplete Inference

- Motivation: solving subnetworks
- Node consistency
- Arc consistency
- Constraint propagation
- Path consistency
- K-consistency
- Generalized Arc consistency

Inference

legal operations on variables, domains, constraints

Inference: $P \longrightarrow P'$

- P' is equivalent to P: $Sol(P) = Sol(P')$
- P' is *presumably easier* to solve than P
  - *smaller* search space
  - constraints are *more* explicit

Inference can be:

- complete: produces the solution
  adaptive consistency
- incomplete: requires further search
  arc consistency

Incomplete Inference: Local Consistency

P constraint network of n variables: Is P solvable ?

Simpler problem:

- P^1_i, P^2_j, P^3_k,…subnetworks of P of 1, 2,3,…variables
- Are they solvable?
  - YES, but there are values that do not appear in any solution $\Rightarrow$ they can be removed from P
  - NO, $\Rightarrow$ P has no solution

#vars in subnet:	1	2	3	…….
	node consistency	*arc consistency*	*path consistency*	

Empty domain: P has no solution !!

Node Consistency (NC)

NC:

- Variable x_i is *node consistent* iff every value of D_i is allowed by R_i
- P is NC iff every variable is NC

Unitary constraint on x_i

NC Algorithm:

```
procedure NC-1 (X,D,C)
  for all xi□X do
    for all a□ Di do
      if a□ Ri then Di:= Di-{a};
```

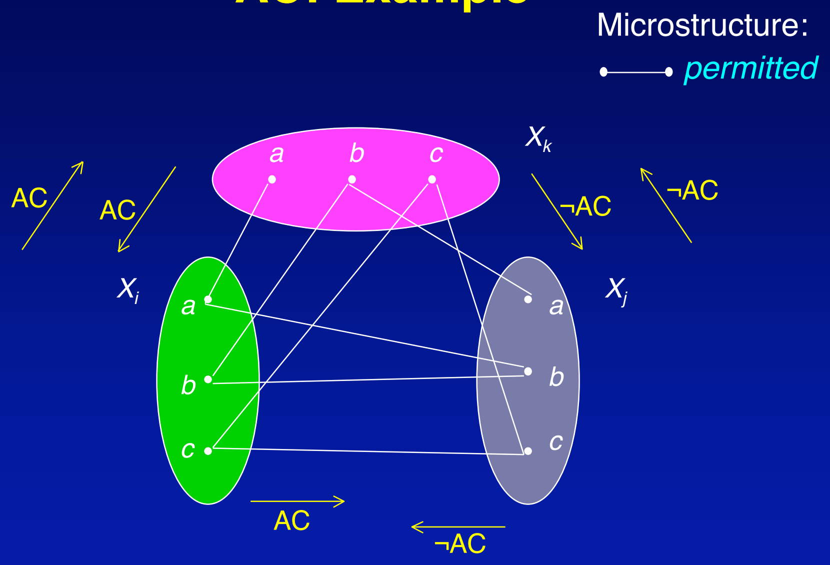
D_i := D_i □ R_i
i: 1,…, n

Binary Arc Consistency (AC)

AC:

- Constraint C_{ij} is *directional arc consistent* (i □ j) iff for all a□ D_i there is b□ D_j such that (a, b)□ R_{ij}
- Constraint C_{ij} is AC iff it is directional AC in both directions
- A problem is AC iff every constraint is AC

AC: Example



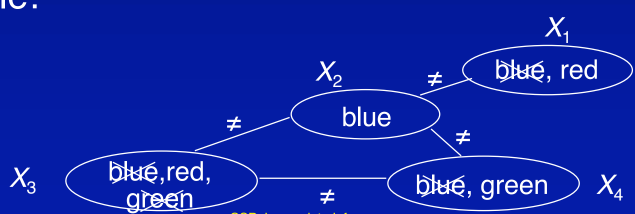
Filtering by Arc Consistency

If for $a \in D_i$ there not exists $b \in D_j$ such that $(a, b) \in R_{ij}$, *a* can be removed from D_i (*a* will not be in any sol)

Domain filtering:

- Remove arc-inconsistent values
- Until no changes

Example:



Function `revise(i,j)`

Function `revise(i,j):`

- R_{ij} becomes directional arc-consistent
- it may remove values of D_i
- iterate over other constraints

```
function revise (i,j variable): bool;
  change := FALSE;
  for all a∈Di do
    if ¬∃b∈Dj st (a,b)∈Rij then
      Di := Di - {a};
      change := TRUE;
  return change;
```

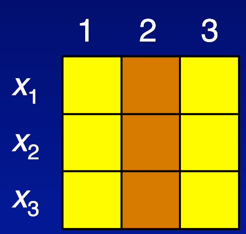
Complexity: $O(d^2)$

Example: 3-queens

c_{12} is not arc-consistent because value 2 of d_1

c_{21} is not arc-consistent because value 2 of d_2

c_{32} is not arc-consistent because value 2 of d_3



Constraint Propagation

- **AC**(c): procedure to make c arc consistent

- To make P arc-consistent, process each constraint ?

AC

c_1 c_2 … c_r

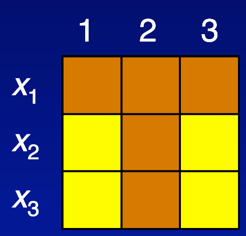
- But **AC**(c) may render other constraints arc-inconsistent

- To make P arc-consistent, iterate:
  - Apply **AC** on $\{c_1,c_2,\ldots,c_r\}$
  - Until no changes in domains: *fix point*

Example: 3-queens

value 2 of d_3 was removed
(to make c_{23} arc-consistent)

this makes c_{13} arc-inconsistent



c_{13} is not arc-consistent
because value 1 of d_1

c_{13} is not arc-consistent
because value 3 of d_1 → domain d_1 empty

no solution !!

Example: Equations on Integers

$x + y = 9$

$2x + 4y = 24$

x	0	1	2	3	4	5	6	7	8	9
y	0	1	2	3	4	5	6	7	8	9

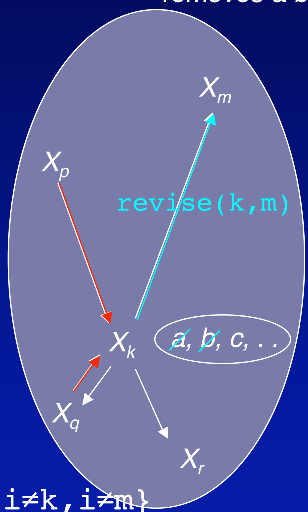
fix point

AC-3: AC Algorithm

`revise(k,m)` removes *a b*

Arcs to revisit?
Arcs no longer AC because *b*, *c* removal

If arc was AC, after `revise(k,m)`
(k,_): it remains AC
(_,k): it may not be AC




```
procedure AC-3 (G)
  Q := {(i,j)|(i,j)∈arc(G)i≠j}
  while Q≠Ø do
    pop arc (k,m) from Q
    if revise(k,m) then
      Q:=Q∪{(i,k)|(i,k)∈arc(G)i≠k,i≠m}
```

Complexity: d × $2e$ × d^2 : $O(ed^3)$
times per constraint, #constraints, revise

Function `revise2001(i,j)`

Function `revise2001(i,j):`

- static ordering on domain values: <
- special structure: last support for (X_i, a) in X_j

```
function revise2001 (i,j variable): bool;
  change := FALSE;
  for all a□D_i do
    if last(i,a,j)□D_j then
      if □b□D_j st last(i,a,j)<b □ (a,b)□R_ij
      then last(i,a,j) := b;
      else
        D_i := D_i - {a};
        change := TRUE;
  return change;
```


AC-2001: AC Algorithm

```
procedure AC-2001 (G)
 | Q := {(i,j)|(i,j)□ arc(G)i≠j}
 | while Q≠Ø do
 |  | pop arc (k,m) from Q
 |  | if revise2001(k,m) then
 └  └  └ Q:=Q□{(i,k)|(i,k)□ arc(G)i≠k,i≠m}
```

Complexity:

- Value (X_i, a) is tested d times on `last(i,a,j)`
- Value (X_i, a) is tested d times on `D_j`
- d values per variable

d (values) □ $(d + d)$ (revise) □ $2e$ (#constraints) : $O(ed^2)$ — OPTIMAL

Binary Path Consistency

A pair of values $((i, a)\ (j, b))$, st $(a, b) \in R_{ij}$, is path consistent iff for all X_k, $i \neq k$, $j \neq k$, there exists $c \in D_k$ such that,

$$(a, c) \in R_{ik} \quad \text{and} \quad (c, b) \in R_{kj}$$

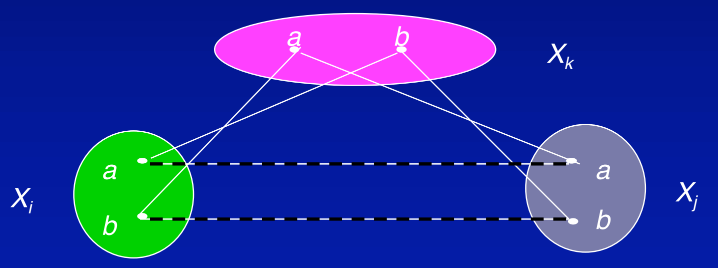
A pair of variables (X_i, X_j) is path consistent iff every pair of values $(a, b) \in R_{ij}$ is path consistent

A problem is path consistent iff every pair of variables is path consistent

Filtering by Path Consistency

If for $(a,b) \in R_{ij}$ there not exists $c \in D_k$ such that $(a, c) \in R_{ik}$, $(b, c) \in R_{jk}$, pair (a,b) can be removed from R_{ij} (it will not be in any sol)

Example:



R_{ij} = empty constraint ⇒ *no solution* !!

Function `revise3(i,j,k)`

Function **`revise3(i,j,k)`**:

- makes C_{ij} path consistent with x_k
- it can remove pairs of permitted value

```
function revise3 (i,j,k variable): bool;
  change := FALSE;
  for all (a,b)∈Rij do
    if ¬∃c∈Dk st (a,c)∈Rik ∧ (b,c)∈Rjk then
      Rij := Rij - {(a,b)};
      change := TRUE;
  return change;
```

Complexity: $O(d^3)$

PC-2: PC Algorithm

PC-2: `revise3` on all possible triangles (including universal constraint)

```
procedure PC-2 (X,D,C)
  Q := {(i,j,k)|1≤i<j≤n,1≤k≤n,k≠i,k≠j};
  while Q≠Ø do
    pop (i,j,k) from Q;
    if revise(i,j,k) then
      Q:=Q ∪ {(l,i,j)(l,j,i)|1≤l≤n,l≠j,l≠i};
```

Complejidad: $O(n^3 d^5)$

K-Consistency

K-Consistency:

- for any subset of k-1 variables $\{X_1, X_2, \ldots, X_{k-1}\}$ consistently assigned;
- for any X_k there exists $d \in D_k$ such that $\{X_1, X_2, \ldots, X_k\}$ is consistent

K-strong-consistency: J-consistent, for $1 \leq J \leq K$

Algorithms for K-strong-consistency:

- Freuder 82, Cooper 89, O(exp K)

Function `reviseK(1,2,..,k)`

Function `reviseK(1,2,…,k)`:

- Makes a subnetwork K consistent
- Discovers nogoods of size K-1

```
function reviseK (1,2,…,k variable): bool;
  change := FALSE;
  for all (a,b,…,p) cons. for (1,…,k-1) do
    if ¬∃q∈Dk st (a,b,…,p,q) is cons. then
      (a,b,…,p) is a nogood of size k-1;
      change := TRUE;
  return change;
```

Complexity: $O(d^k)$

Local Consistency Levels

Nogood: forbidden tuple of values

- In the initial constraints
- Discovered during search / local consistency process

Local consistency levels:		nogood size	
• Node consistency:	1-consistency	0	
• Arc consistency:	2-consistency	1	*removes values from domains*
• Path consistency:	3-consistency	2	*discovers forbidden value pairs*
• …..			
• K-consistency:		K-1	*discovers forbidden value combinations*

Generalized Arc Consistency

- c is arc-consistent iff: every possible value of every variable in *var* (c) appears in *rel*(c)

- If c is not arc-consistent because $a \in D_x$:
  - a will not be in any solution
  - a can be removed: $D_x \leftarrow D_x - \{a\}$ *(domain filtering)*
  - if D_x becomes empty, P has no solution

  inference

- P is arc-consistent iff: every constraint is arc-consistent

- If P is arc-consistent $\not\rightarrow$ P has solution

incomplete inference !!