Constraint Programming: Local Consistencies

Eric MONFROY

IRIN, University of Nantes

Objective

Solving constraint over finite domains

- exhaustive search vs. filtering algorithms
- recap about constraint propagation
- incomplete solvers and local consistency notion
- node consistency (NC algorithm)
- arc consistency (algorithms : AC-1, AC-3, AC-4)
- bound consistency

Intuitive approach to local consistencies

Finite domains

each set isomorphic to a *finite* part of $\mathbb{N}$

1. Set of natural integer that can be represented by a machine
2. Booleans : $\{\text{false}, \text{true}\}$ (or $\{0, 1\}$)
3. Letters : $A, B, C, \ldots$
4. Set of the members of a team
5. $\ldots$

➔ FD $=$ very important to model numerous industrial problems

CSP (reminder)

A *constraint satisfaction problem* (CSP) is defined by :

- a sequence of variables $X = x_1, \ldots, x_n$ with *domains* $D_1, \ldots, D_n$ (associated to the variables)
- a set of constraints $C_1, \ldots, C_l$, each C_i on a sub-sequence Y_i of X

implicitely, the CSP represents the constraint :

$$C_1 \wedge \ldots \wedge C_n \wedge x_1 \in D_1 \wedge \cdots \wedge x_n \in D_n$$

A *solution of the CSP* is a n-tuple $d = (a_1, \ldots, a_n)$ such that :

- $d \in D_1 \times \cdots \times D_n$
- and for each i, $d[Y_i] \in C_i$
  ($d[Y_i]$ satisfies C, or $C(a_{i_1}, \ldots, a_{i_l})$ is true)

Solving CSPs (1)

Look back : variables are instanciated, and “instanciated” constraints are tested

- non-incremental version : *generate and test*
- incremental version : *backtracking*

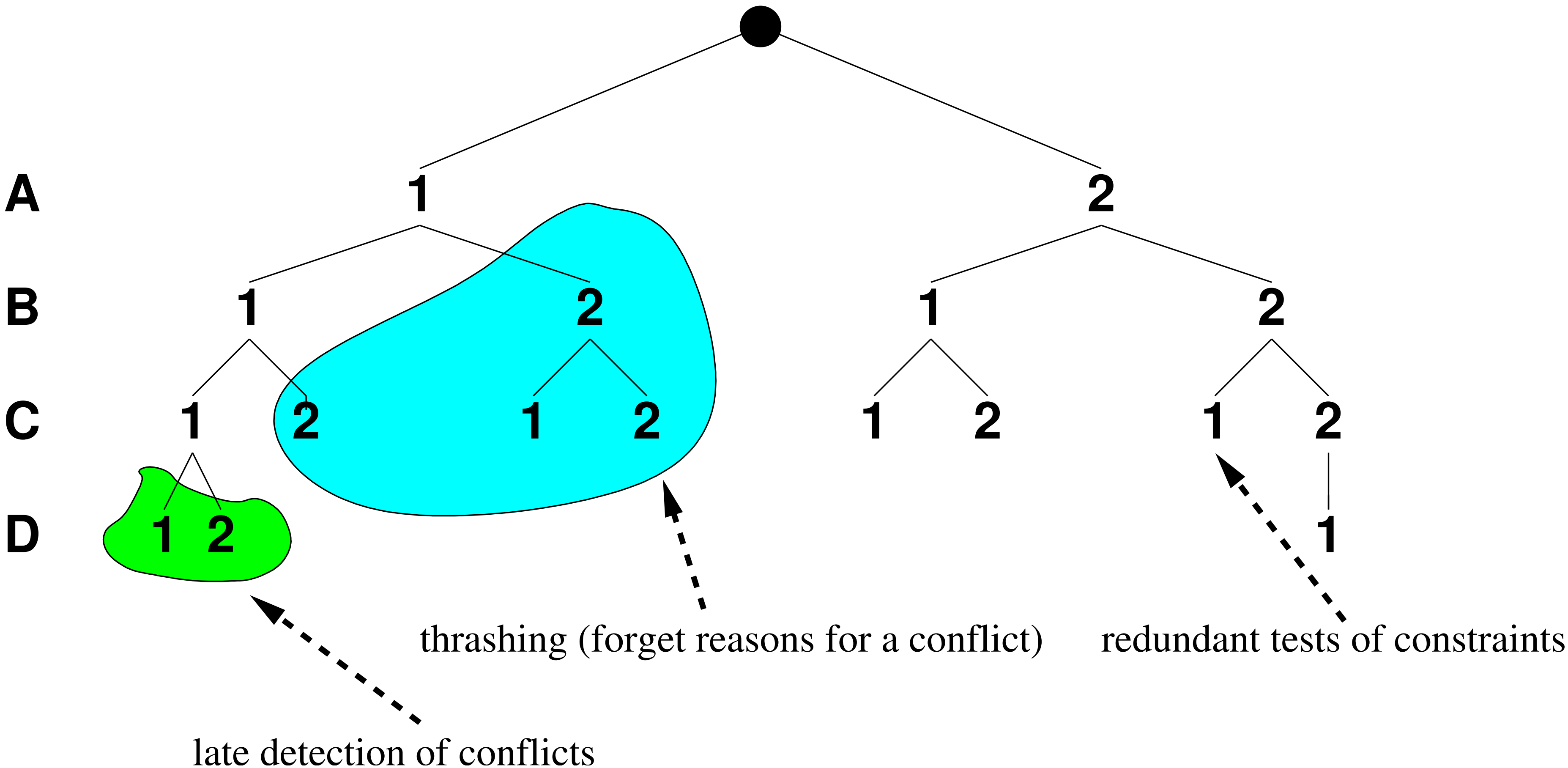
☺ complete and correct

☹ inefficient and costly

clever alternatives : *backjumping*, *backmarking*

Problem of the *look back*

$A \in \{1,2\}, B \in \{1,2\}, C \in \{1,2\}, D \in \{1,2\}$
$A > D \wedge B = C \wedge A = C$



Solving CSPs (2)

basic idea : from a given CSP, find an *equivalent* CSP with smaller domains (smaller search space)

- consider each atomic constraint separately
- filter domains of variables and eliminate *inconsistent* values

☺ active use of the constraints. Many values vialoting constraints are removed

☹ incomplete (complete with split and search)

Constraint solving framework

```
solve(CSP) :
        while not finished do
                pre-process
                constraint propagation
                if happy
                   then finished=true
                   else split
                           part-of search
                endif
        endwhile
```

where part-of search consists in calls to the solve function

Remark : part-of search is one of the mechanisms defining search

Constraint propagation

- replace a CSP by a CSP which is :
  - equivalent (same set of solutions)
  - “smaller” (domains are reduced)
  - “simpler” (constraints are reduced)
- constraint propagation mechanism :

  repeatedly reduce domains or constraints
- can be seen as a fixed point of application of reduction functions
  - reduction function to reduce domains or constraints
  - can be seen as an abstraction of the constraints by reduction functions

Constraint propagation : reducing domains

- Generally :
  - reduce domains using constraint and domains
  - $\rightarrow$ reduce the search space
- generic domain reduction :
  - Given a constraint C over $x_1, \ldots, x_n$ with domains $D_1, \ldots, D_n$
  - select a variable x_i to be reduced
  - delete from D_i all values for x_i that do not participate in a solution of C

Local consistency

- a criterion to stop propagation
- a way to characterize a CSP or a constraint
- why local ?
  - generally, unable to obtain global consistency (incomplete solvers without split and search)
  - thus, local means on a sub-set of a CSP
    $\longrightarrow$ usually, local to ONE constraint
    this sub-set is used to reduce domains

Local consistency (1)

at the beginning : for unary and binary constraints

- unary constraints : *node consistency*
  - for constraints such as : even(x), $y > 5, \ldots$
- binary constraints : *arc consistency*
  - for constraints such as : $x > y + 4$, $x \neq y, \ldots$

Local consistency (2)

then : for n-ary constraints and higher/stronger consistencies

- n-ary constraints : *hyper-arc consistency*
  - for constraints such as : $3.x + y = z$, $and(x, y, z)$, $\ldots$
- (m-)path consistency :
  - using several constraints at a time
- k-consistency :
  - every $(k-1)$-consistent instanciation can be extended to a k-consistent instanciation (k variables)

Local consistency (3)

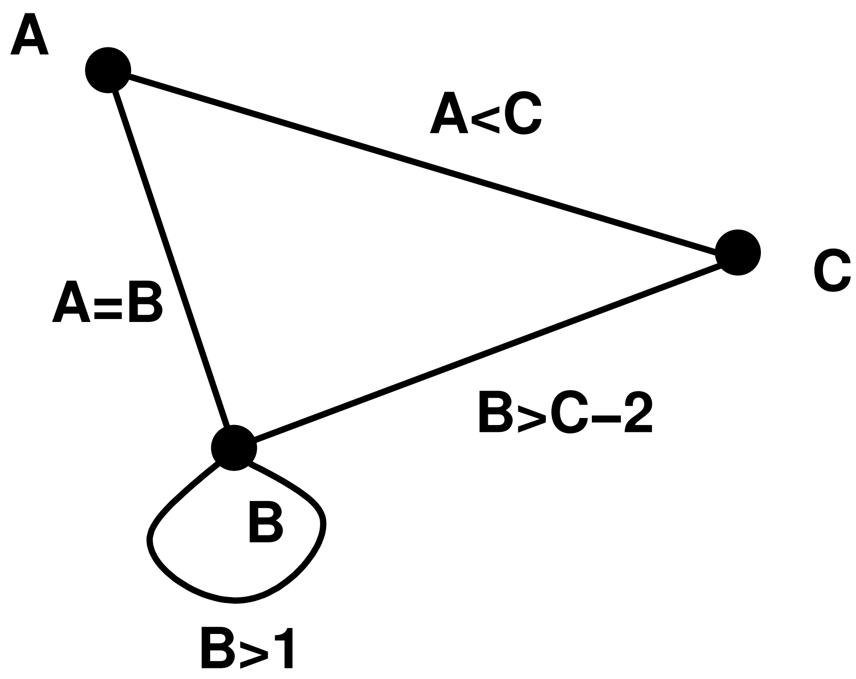
then : consistency on bounds of domains
(when domains are too big to consider each value)

- bound consistency (finite domains) :
  - for constraints such as : $3.x + y = z$ with domains $x \in [-10000..9000], y \in [-5000..9000], z \in [100..19000], \ldots$
- 2b consistency (real interval, “primitive” constraints)
  - for constraints such as : $3.23 * x * y = z$ with domains $x \in [-100.1547..9000.0], y \in [-5.12..9.0], z \in [0.99..1.01]$
- box consistency (real interval)
  - for constraints such as : $3.23 * x + y * x = z^2 + exp(x)$ with domains $x \in [-10.147..90.0], y \in [-5.1..9.0], z \in [0.99..1.01]$

Local consistency : intuitive (1)

$\{B > 1, A < C, A = B, B > C - 2;\ A, B, C \in \{1, 2, 3\}\}$

the CSP can be represented by the graph :

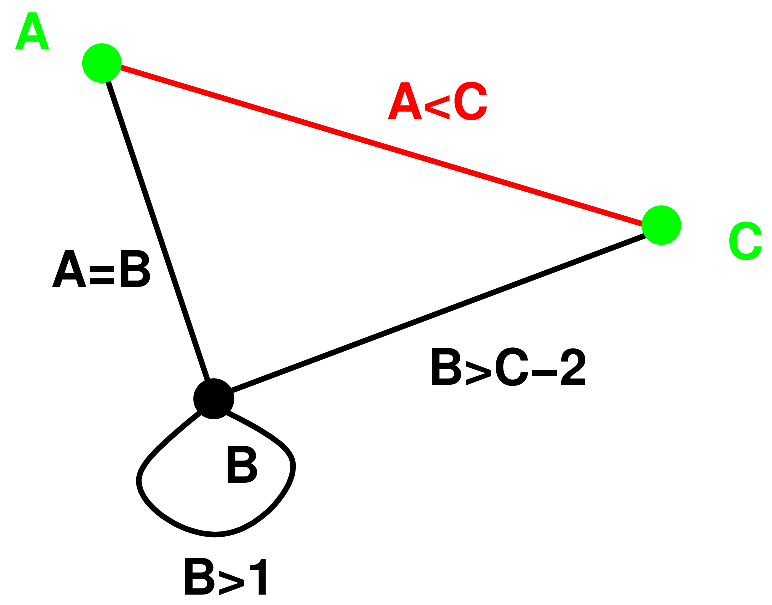


how to reduce domains ?

Idea : follow arcs of the graph

Local consistency : intuitive (2)

using $A < C$, A and/or C may be reduced



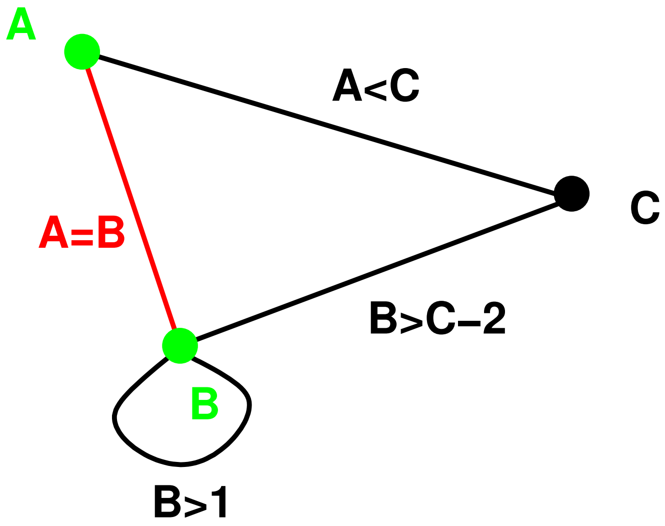
A and C reduced to $A \in \{1, 2\}$, $C \in \{2, 3\}$

now, reduce B using $A = B$ or $B > C - 2$

Local consistency : intuitive (3)

$\{B > 1, A < C, A = B, B > C - 2;\ A \in \{1, 2\}, B \in \{1, 2, 3\}, C \in \{2, 3\}\}$

using $A = B$, A and/or B may be reduced



B reduced to $B \in \{1, 2\}$

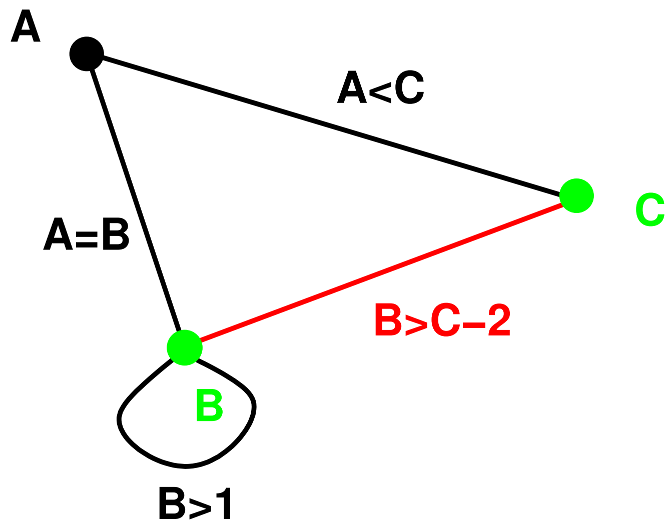
A not reduced, so useless to use $A < C$

now, reduce C and/or B using $B > C - 2$, or reduce B using $B > 1$

Local consistency : intuitive (4)

$\{B > 1, A < C, A = B, B > C - 2;\ A \in \{1, 2\}, B \in \{1, 2\}, C \in \{2, 3\}\}$

using $B > C - 2$, B and/or C may be reduced



B, C not reduced

$B > 1$ can be used,
and so on until no domain can be reduced anymore

Local consistencies : definition

Node consistency : definition

Definition : an atomic unary constraint C over the variable x with the domain D_x is node consistent iff :

$$\forall a \in D_x : a \in C \quad (\text{or } C(a))$$

Remarks :

- a non unary constraint is always considered as node consistent
- a CSP is node consistent if all its constraints are node consistent

Examples :

- $x \in \{4, 6\}, \quad \text{even}(x)$ is node consistent
- $x \in [2..12], x > 5$ is not node consistent

Node consistency : algorithm

```
node_consistency(C, D)
begin
    let C ≡ C_1, ··· , C_n
    for i ← 1 to n do
        D ← revise_node(C_i, D)
    endfor
    return(D)
end

revise_node(C, D)
begin
    if (|var(C)| == 1) then
        {x} ← var(C)
        D_x ← {d ∈ D_x | d ∈ C}
    endif
    return(D)
end
```

Arc Consistency : definition

Definition : an atomic binary constraint C over the variables x and y with domains D_x and D_y is arc consistent iff :

- $\forall a \in D_x \exists b \in D_y$ s.t. $(a, b) \in C$
- $\forall b \in D_y \exists a \in D_x$ s.t. $(a, b) \in C$

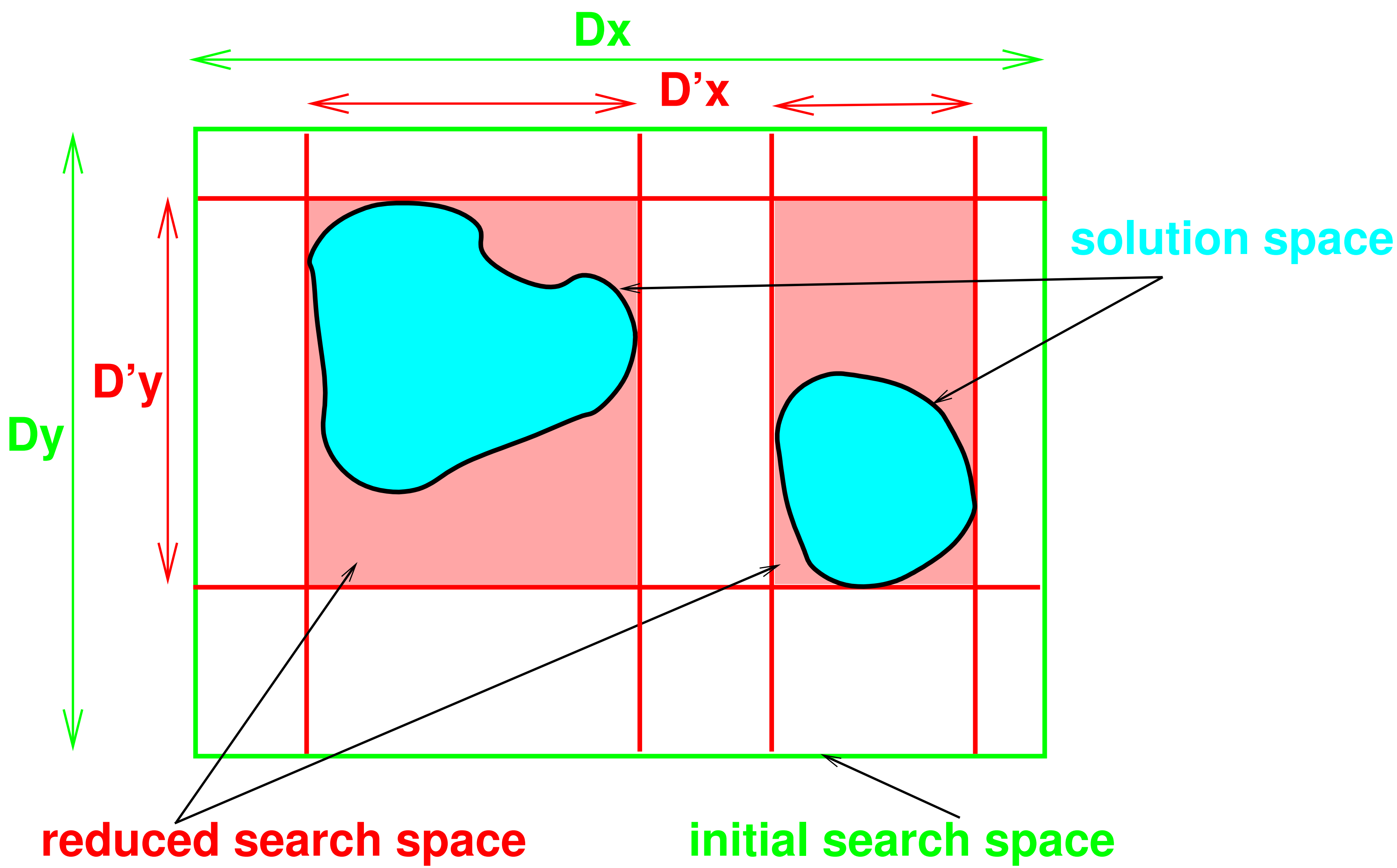
Remarks :

- a non binary constraint is arc consistent
- a CSP is arc consistent iff all its constraints are arc consistent

Examples :

- $x \in \{1, 3\}, y \in \{2, 4\}, x + y = 5$ is arc consistent
- $x \in \{1, 2\}, y \in \{1, 7\}, x = y$ is not arc consistent

Arc Consistency : intuition



Arc consistency : AC-1 algorithm

AC-1$(C, \mathbf{D})$
begin
  let $C \equiv C_1, \cdots, C_n$
  repeat
    $\mathbf{D}' \leftarrow \mathbf{D}$
    for $i \leftarrow 1$ **to** n **do**
      $\mathbf{D} \leftarrow$ revise_arc$(C_i, \mathbf{D})$
    endfor
  until$(\mathbf{D}' = \mathbf{D})$
  return$(\mathbf{D})$
end

revise_arc$(C, \mathbf{D})$
begin
  if $(|$var$(C)| == 2)$ **then**
    $\{x, y\} \leftarrow$ var(C)
    $\mathbf{D}_x \leftarrow \{a \in \mathbf{D}_x \mid \exists b \in \mathbf{D}_y : (a, b) \in C\}$
    $\mathbf{D}_y \leftarrow \{b \in \mathbf{D}_y \mid \exists a \in \mathbf{D}_x : (a, b) \in C\}$
  endif
  return$(\mathbf{D})$
end

Local consistency : example

Consider the CSP

$$\{X < Y, Y < Z, Z \leqslant 2;\ D_X, D_Y, D_Z \in \{1, 2, 3\}\}$$

Computation of node consistency

➔ 3 removed from D_z

Computation for arc consistency

➔ inconsistent

Generally : incompleness. Algorithm returns some domains for the variables. All kept values are not necessarily solution !

Arc consistency $\neq$ consistency

Consider the CSP

$$\{x = y, x \neq y, D_x \in \{a, b\}, D_y \in \{a, b\}$$

the CSP is arc consistency

➔ a and b cannot be reduced using $x = y$ or $x \neq y$

However, the CSP is not consistent

➔ no solution

Problems of AC-1

- inefficient
- wake-up constraints when useless
  - no modification of variable domains
- no early detectection of failed CSP
  - two loops with failed CSP

Idea of AC-3

Idea : wake up constraints when variables have effectively been modified

Mechanism :

- manage a set of constraints to use
- update this set after each reduction attemp
  - add constraints with at least one modified variable
- stop
  - when no more constraint to consider
- failed CSP
  - stop as soon as one domain is empty

Local consistency : AC-3 algorithm

AC-3($C \equiv C_1, \cdots, C_n, \mathbf{D}$)

begin

$\mathcal{S} \leftarrow \{C_1, \cdots, C_n\}$

while $(\mathcal{S} \neq \varnothing)$

choose and extract C from $\mathcal{S}$

$\mathbf{D}' \leftarrow$ revise_arc$(C, \mathbf{D})$

if $(\mathbf{D}' = \varnothing)$ **then return**$(\varnothing)$ **endif**

$\mathcal{S} \leftarrow \mathcal{S} \cup \{C_i \mid \exists x \in$ var(C_i) s.t. $\mathbf{D}'_x \neq \mathbf{D}_x\}$

$\mathbf{D} \leftarrow \mathbf{D}'$

endwhile

return$(\mathbf{D})$

end

Revise_arc unchanged

Local consistency : AC-4 algorithm

Possible speed-up for AC-3 : to keep in memory for each binary constraints $c(x, y)$ *support* relations between values of D_x and D_y :

- how many values of D_y support each value of D_x
- what are the values of D_x supported by a particular value of D_y

and vice-versa.

☺ when a value is removed, we know precisely the changes that are induced, and which constraints to wake-up

☹ memory space

AC-4 : best theoretical complexity... often the worst in pratice

Hyper-arc consistency (1)

what about n-ary constraints for $n > 2$?

hyper-arc consistency : a constraint C over the variables $x_1, \ldots, x_n$ with domains $D_1, \ldots, D_n$ is *hyper-arc consistent* w.r.t. x_i $(i \in \{1, \ldots, n\})$ iff :

$$\forall a \in D_i, \ \ \exists d \in D_1 \times \ldots \times D_n \ \ \text{s.t.} \ \ d \in C \ \ \text{and} \ \ a = d[x_i]$$

Hyper-arc consistency (2)

- a constraint C over $x_1, \ldots, x_n$ with domains $D_1, \ldots, D_n$ is hyper-arc consistent iff c is hyper-arc consistent w.r.t. x_i for all $i \in \{1, \ldots, n\}$.
- a CSP is hyper-arc consistent iff all its constraints are hyper-arc consistent

Hyper-arc consistency (3)

Examples : constraints

- $x \in \{3, 5, 7\}, y \in \{1, 4\}, z \in \{4, 6, 14\}$
  $x + 2 * y = z + 1$ is hyper-arc consistent
- $x \in \{1, 2, 4\}, y \in \{3, 5\}, z \in \{4, 5\}$
  $x + y - z = 0$ is not hyper-arc consistent
  (not hyper-arc consistent w.r.t. x, e.g., value 4)

Examples : CSP

- { and(x,y,z), or(x,y,1) ; $x \in \{1\}, y \in \{0, 1\}, z \in \{0, 1\}\}$
  the CSP is hyper-arc consistent
- { and(x,y,z), or(x,y,1) ; $x \in \{0, 1\}, y \in \{0, 1\}, z \in \{1\}\}$
  the CSP is not hyper-arc consistent

Directional arc consistency (1)

Idea : directional propagation

consider an ordering $<$ on variables :

directional arc consistency : a constraint C over the variables x, y with domains D_x, D_y is directionally arc consistent w.r.t. $<$ iff :

- if $x < y$:

$$\forall a \in D_x \exists b \in D_y \ (a, b) \in C$$

- if $y < x$:

$$\forall b \in D_y \exists a \in D_x \ (a, b) \in C$$

a CSP is directionally arc consistent w.r.t. $<$ iff all its constraints are

Directional arc consistency (2)

example :

$$\{x < y;\;\; x \in [2..7], y \in [3..7]\}$$

- the CSP is not arc consistent
- the CSP is directionally arc consistent w.r.t. $y < x$
- the CSP is not directionally arc consistent w.r.t. $x < y$

Limitations of arc/hyper-arc consistency (1)

Problem : determining arc/hyper-arc consistency can be too costly

Example :
$\{x = y + z, 2.x = 4.y;\ x, y, z \in \{1, 2, 8, 12, 34, \ldots, 110000\}\}$
domain reduction : each value must be tested ! ! !

Idea : to relax consistency ➔test only *bounds*

Limitations of arc/hyper-arc consistency (2)

Example : $\{x < y, y < z, z < x;\ x, y, z \in [1..10000]\}$

domain reduction :

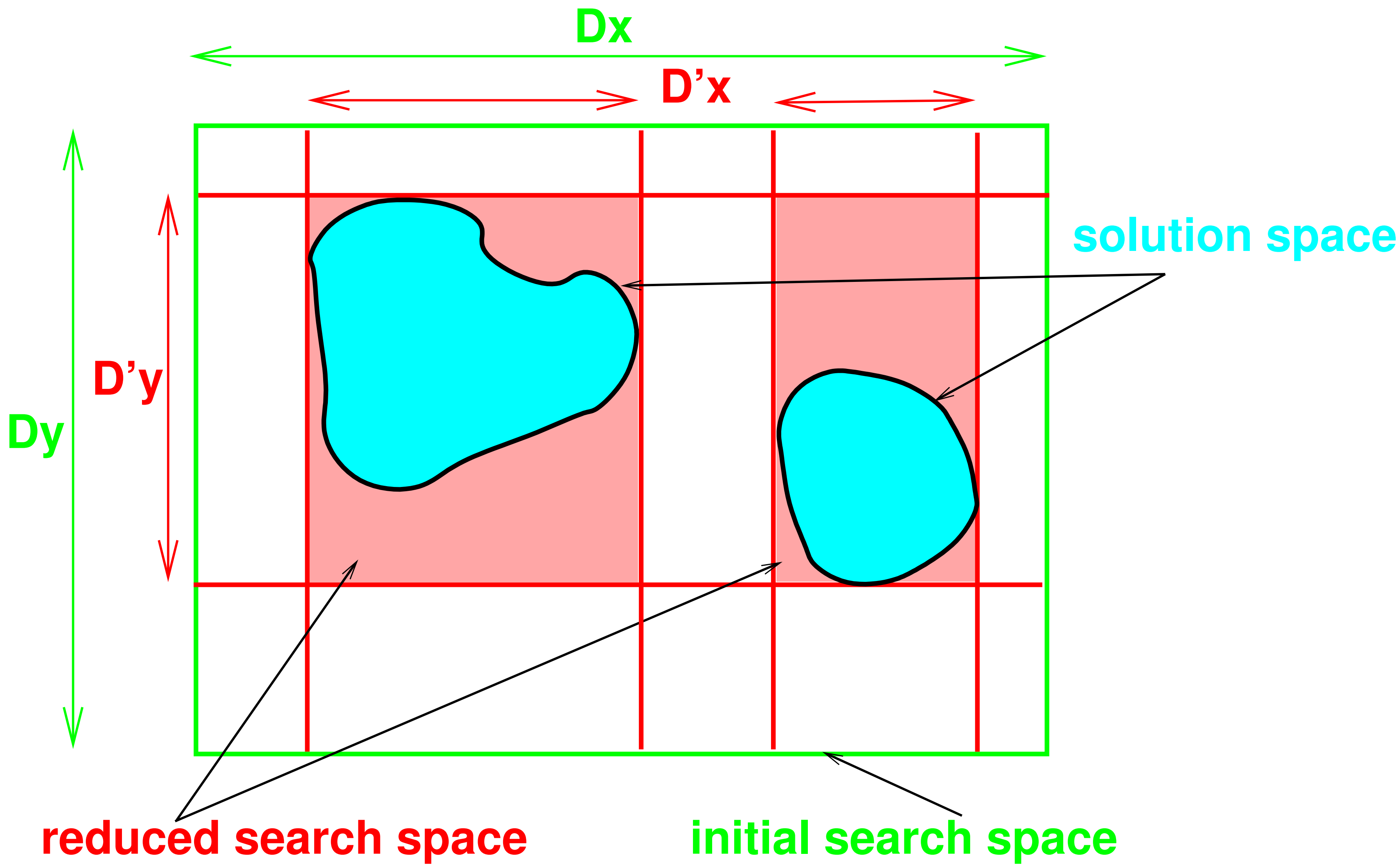
- using the first constraint :
  $\{x < y, y < z, z < x;\ x \in [1..9999], y \in [2..10000], z \in [1..10000]\}$
- using the second constraint :
  $\{x < y, y < z, z < x;\ x \in [1..9999], y \in [2..9999], z \in [3..10000]\}$
- using the third constraint :
  $\{x < y, y < z, z < x;\ x \in [4..9999], y \in [2..9999], z \in [3..9998]\}$
- . . . until a domain is empty

Idea 1 : testing bounds does not change the cost

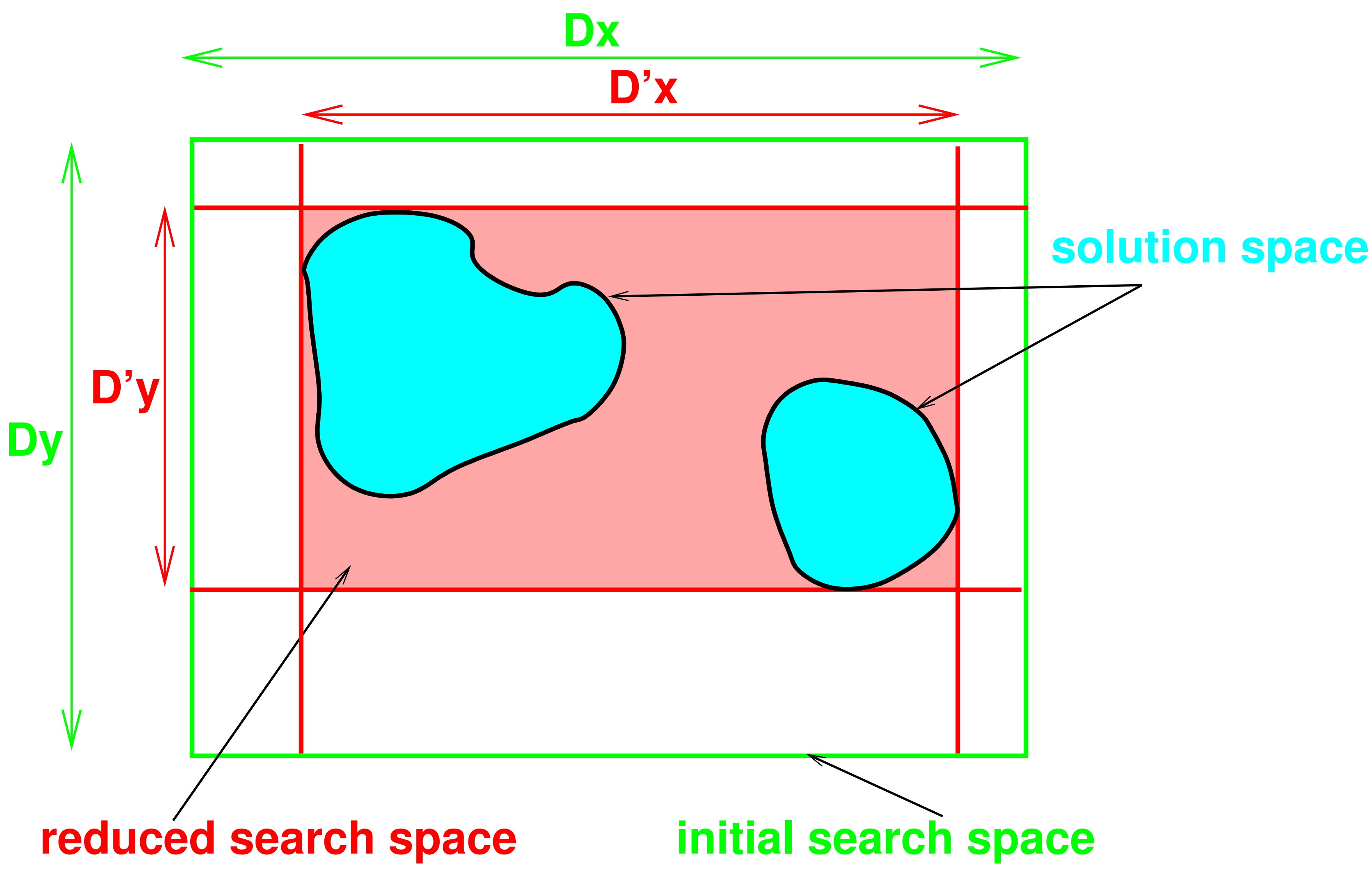
Idea 2 : symbolic computation $\rightarrow$ direct proof (transitivity of $<$)

Idea 3 : using two constraints at a time $\rightarrow$ *path consistency*

Arc consistency : intuition (recap)



Bound consistency : intuition



Bound consistency (1)

Idea : domains are represented by intervals

bound consistency : a constraint C over the variables $x_1, \ldots, x_n$ with domains $D_1, \ldots, D_n$ is bound consistent w.r.t. x_i with domain $D_i = [l, r]$ $(i \in \{1, \ldots, n\})$ iff :

$$\exists d \in D_1 \times \ldots \times D_n \quad \text{s.t.} \quad d[x_i] = l \ \text{ and } \ d \in C$$

$$\text{and}$$

$$\exists d \in D_1 \times \ldots \times D_n \quad \text{s.t.} \quad d[x_i] = r \ \text{ and } \ d \in C$$

Bound consistency (2)

- a constraint c is bound consistent iff it is w.r.t. x_i for all $i \in \{1, \ldots, n\}$.
- a CSP is bound consistent iff all its constraints are bound consistent

Examples :

- $x \in [3..6], y \in [2, 3], z \in [5, 9], x + y = z$
  is bound consistent
- $x \in [2..3], y \in [3..6], z \in [1..19], 3 * x = y + z$
  is not bound consistent
  (not bound consistent w.r.t. z, e.g., value 19)

Bound consistency (3)

computing bound consistency for “primitive” constraints : reasonning only on bounds

➔ easy, less complexe

Examples :

- $x + y = z$ with $D_x = [a..b]$, $D_y = [c..d]$, $D_z = [e, f]$
- $x \leqslant y$ with $D_x = [a..b]$, $D_y = [c..d]$

Bound consistency : $x \leqslant y$

constraint :

$$x \leqslant y$$

to get bound consistency :

$$x \leqslant \max D_y$$
$$y \geqslant \min D_x$$

Bound consistency : $x \leqslant y$

% $x \leqslant y$

revise_leq($D_x = [a..b], D_y = [c..d]$)

begin

$D_x \leftarrow [a.. \min\{b, d\}]$

$D_y \leftarrow [\max\{a, c\}..d]$

return(D_x, D_y)

end

Bound consistency : $x + y = z$

constraint :

$$x + y = z \quad \equiv \quad x = z - y \quad \equiv \quad y = z - x$$

to get bound consistency :

$$\begin{array}{ll} z \geqslant \min D_x + \min D_y & z \leqslant \max D_x + \max D_y \\ x \geqslant \min D_z - \max D_y & x \leqslant \max D_z - \min D_y \\ y \geqslant \min D_z - \max D_x & y \leqslant \max D_z - \min D_x \end{array}$$

Bound consistency : $x + y = z$

% x+y=z

revise_addition($D_x = [a..b], D_y = [c..d], D_z = [e..f]$)

begin

$D_x \leftarrow D_x \cap [e - d..c - f]$

$D_y \leftarrow D_y \cap [e - b..f - a]$

$D_z \leftarrow D_z \cap [a + c..b + d]$

return(D_x, D_y, D_z)

end

Combination BT/AC

solvers using only local consistency : incomplete
realizing a complete solver ➔ combination with backtracking

Look ahead : instanciation of some variables with filtering of domains
➔ *forward checking*, *partial look-ahead*, *full look-ahead*

☺ no exploration of branches trivialy without solution

☹ more work after each instanciation