DM841
Constraint Programming

Constraint Propagation Algorithms

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Resume

- Definitions
  (CSP, restrictions, projections, istantiation, local consistency)
- Tigthtenings
- Global consistent (any instantiation local consistent can be extended to a solution) needs exponential time
  $\rightsquigarrow$ local consistency defined by condition Φ of the problem
- Tightenings by constraint propagation: reduction rules + rules iterations
  - reduction rules $\Leftrightarrow$ Φ consistency
  - rules iteration: reach fixed point, that is, closure of all tightenings that are Φ consistent

Outline

1. Local Consistency

2. Arc Consistency Algorithms

Node Consistency

We call a CSP node consistent if for every variable x every unary constraint on x coincides with the domain of x.

Example

- $\langle \mathcal{C}, x_1 \geq 0, \ldots, x_n \geq 0; x_1 \in \mathbb{N}, \ldots, x_n \in \mathbb{N} \rangle$
  and $\mathcal{C}$ does not contain other unary constraints
  node consistent
- $\langle \mathcal{C}, x_1 \geq 0, \ldots, x_n \geq 0; x_1 \in \mathbb{N}, \ldots, x_n \in \mathbb{Z} \rangle$
  and $\mathcal{C}$ does not contain other unary constraints
  not node consistent

A CSP is node consistent iff it is closed under the applications of the Node Consistency rule (propagator):

$$\frac{\langle C; x \in D \rangle}{\langle C; x \in C \cap D \rangle}$$

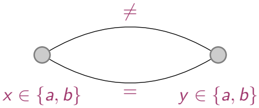
(the rule is parameterised by a variable x and a unary constraint C)

Arc Consistency

Arc consistency: every value in a domain is consistent with every binary constraint.

- $C = c(x, y)$ with $\mathcal{D} = \{D(x), D(y)\}$ is arc consistent iff
  - $\forall a \in D(x)$ there exists $b \in D(y)$ such that $(a, b) \in C$
  - $\forall b \in D(y)$ there exists $a \in D(x)$ such that $(a, b) \in C$
- $\mathcal{P}$ is arc consistent iff it is AC for all its binary constraints

In general arc consistency does not imply global consistency.
An arc consistent but inconsistent CSP:



A consistent but not arc consistent CSP:

Arc Consistency

A CSP is arc consistent iff it is closed under the applications of the Arc Consistency rules (propagators):

$$\frac{\langle C; x \in D(x), y \in D(y)\rangle}{\langle C; x \in D'(x), y \in D(y)\rangle}$$

where $D'(x) := \{a \in D(x) \mid \exists b \in D(y), (a, b) \in C\}$

$$\frac{\langle C; x \in D(x), y \in D(y)\rangle}{\langle C; x \in D(x), y \in D'(y)\rangle}$$

where $D'(y) := \{b \in D(y) \mid \exists a \in D(x), (a, b) \in C\}$

Generalized Arc Consistency (GAC)

Given arbitrary (non-normalized, non-binary) $\mathcal{P}$, $C \in \mathcal{C}$, $x_i \in X(C)$

(Value) $v \in D(x_i)$ is consistent with C in $\mathcal{D}$ iff $\exists$ a valid tuple τ for C: $v_i = \tau[x_i]$. τ is called support for (x_i, v_i)

(Variable) $\mathcal{D}$ is GAC on C for x_i iff all values in $D(x_i)$ are consistent with C in $\mathcal{D}$ (i.e., $D(x_i) \subseteq \pi_{\{x_i\}}(C \cap \pi_{\{X(C)\}}(\mathcal{D}))$)

(Problem) $\mathcal{P}$ is GAC iff $\mathcal{D}$ is GAC for all x in X on all $C \in \mathcal{C}$

$\mathcal{P}$ is arc inconsistent iff the only domain tighter than $\mathcal{D}$ which is GAC for all variables on all constraints is the empty set.

(aka, hyperarc consistency, domain consistency)

Example

$\langle x = 1, y \in \{0,1\}, z \in \{0,1\}; \mathcal{C} = \{x \wedge y = z\}\rangle$
is hyperarc consistent

$\langle x \in \{0,1\}, y \in \{0,1\}, z = 1; \mathcal{C} = \{x \wedge y = z\}\rangle$
is not hyper-arc consistent

Example: arc consistency $\neq$ 2-consistency, AC $<$ 2C on non-normalized binary CSP, and incomparable on arbitrary CSP (later)

Generalized Arc Consistency

A CSP is arc consistent iff it is closed under the applications of the Arc Consistency rules (propagators):

$$\frac{\langle C; x_1 \in D(x), \ldots, x_k \in D(x_k)\rangle}{\langle C; x_1 \in D(x_1), \ldots, x_{i-1} \in D(x_{i-1}), x_i \in D'(x_i), x_{i+1} \in D(x_{i+1}), \ldots, x_k \in D(x_k)\rangle}$$

where $D'(x_i) := \{a \in D(x_i) \mid \exists \tau \in C, a = \tau[x_i]\}$

Outline

1. Local Consistency

2. Arc Consistency Algorithms

Arc Consistency

Arc Consistency rule 1 (propagator):

$$\frac{\langle C; x \in D(x), y \in D(y)\rangle}{\langle C; x \in D'(x), y \in D(y)\rangle}$$

where $D'(x) := \{a \in D(x) \mid \exists b \in D(y), (a, b) \in C\}$

This can also be written as ($\bowtie$ represents the join):

$$D(x) \leftarrow D(x) \cap \pi_{\{x\}}(\bowtie(C, D(y)))$$

Arc Consistency rule 2 (propagator):

$$\frac{\langle C; x \in D(x), y \in D(y)\rangle}{\langle C; x \in D(x), y \in D'(y)\rangle}$$

where $D'(y) := \{b \in D(y) \mid \exists a \in D(x), (a, b) \in C\}$

This can also be written as:

$$D(y) \leftarrow D(y) \cap \pi_{\{y\}}(\bowtie(C, D(x)))$$

Generalized Arc Consistency

(Generalized) Arc Consistency rule (propagator):

$$\frac{\langle C; x_1 \in D(x), \ldots, x_k \in D(x_k)\rangle}{\langle C; x_1 \in D(x_1), \ldots, x_{i-1} \in D(x_{i-1}), x_i \in D'(x_i), x_{i+1} \in D(x_{i+1}), \ldots, x_k \in D(x_k)\rangle}$$

where $D'(x_i) := \{a \in D(x_i) \mid \exists \tau \in C, a = \tau[x_i]\}$

This can also be written as:

$$D(x_i) \leftarrow D(x_i) \cap \pi_{\{x_i\}}(C \cap \pi_{X(C)}(\mathcal{D}))$$

AC1 – Reduction rule

Revision: making a constraint arc consistent by propagating the domain from a variable to anohter
Corresponds to:

$$D(x) \leftarrow D(x) \cap \pi_{\{x\}}(\bowtie(C, D(y)))$$

for a given variable x and constraint C
Assume normalized network:

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**

Complexity: $O(d^2)$ or $O(rd^r)$
d values, r arity

AC1 – Rules Iteration

Binary case

```
procedure AC-1
 { Q <- {c(Xi,Xj) in C};
    repeat
      CHANGE <- false;
      for each c(Xi,Xj) in Q do
        { CHANGE <- REVISE(Xi,Xj) or CHANGE; }
    until not(CHANGE) }
```

Interaction among constraints

- Complexity (Mackworth and Freuder, 1986): $O(end^3)$
  e number of arcs, n variables
  (ed^2 each loop, a single succesful removal causes all loop again $\rightsquigarrow$ nd number of loops)
- best-case = $O(ed)$
- Arc-consistency is at least $O(ed^2)$ in the worst case (see later)
- $\rightsquigarrow$ too many calls to Revise

AC3 (Macworth, 1977)

General case – Arc oriented (coarse-grained)

function `Revise3`(**in** x_i: variable; c: constraint): **Boolean** ;
  begin
    CHANGE $\leftarrow$ **false**;
    foreach $v_i \in D(x_i)$ **do**
      if $\nexists \tau \in c \cap \pi_{X(c)}(D)$ *with* $\tau[x_i] = v_i$ **then**
        remove v_i from $D(x_i)$;
        CHANGE $\leftarrow$ **true**;
    return CHANGE ;
  end

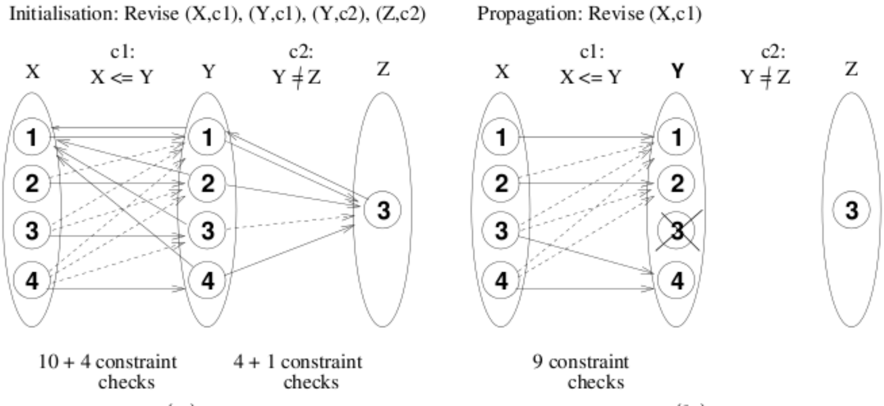
function `AC3/GAC3`(**in** X: set): **Boolean** ;
  begin
    /* initalisation */;
    $Q \leftarrow \{(x_i, c) \mid c \in C, x_i \in X(c)\}$;
    /* propagation */;
    while $Q \neq \emptyset$ **do**
      select and remove (x_i, c) from Q;
      if `Revise`(x_i, c) **then**
        if $D(x_i) = \emptyset$ **then** return **false** ;
        else $Q \leftarrow Q \cup \{(x_j, c') \mid c' \in C \wedge c' \neq c \wedge x_i, x_j \in X(c') \wedge j \neq i\}$;
    return **true** ;
  end

$O(er^3 d^{r+1})$ time
$O(er)$ space

AC3

Example

$$\mathcal{P} = \langle X = (x, y, z), \mathcal{D} = \{D(x) = D(y) = \{1, 2, 3, 4\}, D(z) = \{3\}\}, \mathcal{C} = \{C_1 \equiv x \leq y, C_2 \equiv y \neq z\}\}\rangle$$



AC4

Binary normalized problems – value oriented (fine grained)

function AC4(**in** X: set): **Boolean** ;
  begin
    /* initialization */;
    $Q \leftarrow \emptyset$; $S[x_j, v_j] = 0, \forall v_j \in D(x_j), \forall x_j \in X$;
    foreach $x_i \in X, c_{ij} \in C, v_i \in D(x_i)$ **do**
      initialize counter$[x_i, v_i, x_j]$ to $|\{v_j \in D(x_j) \mid (v_i, v_j) \in c_{ij}\}|$;
      if *counter*$[x_i, v_i, x_j] = 0$ **then** remove v_i from $D(x_i)$ and add (x_i, v_i) to Q;
      add (x_i, v_i) to each $S[x_j, v_j]$ s.t. $(v_i, v_j) \in c_{ij}$;
      if $D(x_i) = \emptyset$ **then** return **false** ;
    /* propagation */;
    while $Q \neq \emptyset$ **do**
      select and remove (x_j, v_j) from Q;
      foreach $(x_i, v_i) \in S[x_j, v_j]$ **do**
        if $v_i \in D(x_i)$ **then**
          counter$[x_i, v_i, x_j]$ = counter$[x_i, v_i, x_j] - 1$;
          if *counter*$[x_i, v_i, x_j] = 0$ **then**
            remove v_i from $D(x_i)$; add (x_i, v_i) to Q;
            if $D(x_i) = \emptyset$ **then** return **false** ;
    return **true** ;
  end

counter$[x_i, v_i, x_j]$: how many supports v_i has on c_{ij}

$S[x_j, v_j]$: all values that are supported by (x_j, v_j) on c_{ij}

$O(ed^2)$ time (optimal)
$O(ed^2)$ space
$O(erd^r)$ time for GAC

AC4
Example

$$\mathcal{P} = \langle X = (x, y, z),\ \mathcal{DE} = \{D(x) = D(y) = \{1, 2, 3, 4\}, D(z) = \{3\}\},$$
$$\mathcal{C} = \{C_1 \equiv x \leq y, C_2 \equiv y \neq z\}\}\rangle$$

$$\begin{array}{lll}
\texttt{counter}[x,1,y] = 4 & \texttt{counter}[y,1,x] = 1 & \texttt{counter}[y,1,z] = 1 \\
\texttt{counter}[x,2,y] = 3 & \texttt{counter}[y,2,x] = 2 & \texttt{counter}[y,2,z] = 1 \\
\texttt{counter}[x,3,y] = 2 & \texttt{counter}[y,3,x] = 3 & \texttt{counter}[y,3,z] = 0 \\
\texttt{counter}[x,4,y] = 1 & \texttt{counter}[y,4,x] = 4 & \texttt{counter}[y,4,z] = 1 \\
 & & \texttt{counter}[z,3,y] = 3
\end{array}$$

$$\begin{array}{ll}
S[x,1] = \{(y,1),(y,2),(y,3),(y,4)\} & S[y,1] = \{(x,1),(z,3)\} \\
S[x,2] = \{(y,2),(y,3),(y,4)\} & S[y,2] = \{(x,1),(x,2),(z,3)\} \\
S[x,3] = \{(y,3),(y,4)\} & S[y,3] = \{(x,1),(x,2),(x,3)\} \\
S[x,4] = \{(y,4)\} & S[y,4] = \{(x,1),(x,2),(x,3),(x,4),(z,3)\} \\
 & S[z,3] = \{(y,1),(y,2),(y,4)\}
\end{array}$$

AC6

Binary normalized problems

$S[x_j, v_j]$: list of values (x_i, v_i) currently having (x_j, v_j) as their first support

```
function AC6(in X: set): Boolean ;
  begin
    /* initialization */;
    Q ← ∅; S[x_j, v_j] = 0, ∀v_j ∈ D(x_j), ∀x_j ∈ X;
    foreach x_i ∈ X, c_ij ∈ C, v_i ∈ D(x_i) do
        v_j ← smallest value in D(x_j) s.t. (v_i, v_j) ∈ c_ij;
        if v_j exists then add (x_i, v_i) to S[x_j, v_j];
        else remove v_i from D(x_i) and add (x_i, v_i) to Q;
        if D(x_i) = ∅ then return false ;
    /* propagation */;
    while Q ≠ ∅ do
        select and remove (x_j, v_j) from Q;
        foreach (x_i, v_i) ∈ S[x_j, v_j] do
            if v_i ∈ D(x_i) then
                v'_j ← smallest value in D(x_j) greater than v_j s.t. (v_i, v_j) ∈ c_ij;
                if v'_j exists then add (x_i, v_i) to S[x_j, v'_j];
                else
                    remove v_i from D(x_i); add (x_i, v_i) to Q;
                    if D(x_i) = ∅ then return false ;
    return true ;
  end
```

$O(ed^2)$ time
$O(ed)$ space

AC6
Example

$$\mathcal{P} = \langle X = (x, y, z), \mathcal{DE} = \{D(x) = D(y) = \{1, 2, 3, 4\}, D(z) = \{3\}\}, \mathcal{C} = \{C_1 \equiv x \leq y, C_2 \equiv y \neq z\}\}\rangle$$

$$S[x, 1] = \{(y, 1), (y, 2), (y, 3), (y, 4)\}$$
$$S[x, 2] = \{\}$$
$$S[x, 3] = \{\}$$
$$S[x, 4] = \{\}$$

$$S[y, 1] = \{(x, 1), (z, 3)\}$$
$$S[y, 2] = \{(x, 2)\}$$
$$S[y, 3] = \{(x, 3)\}$$
$$S[y, 4] = \{(x, 4)\}$$
$$S[z, 3] = \{(y, 1), (y, 2), (y, 4)\}$$

Reverse2001

Binary case — optimal coarse-grained

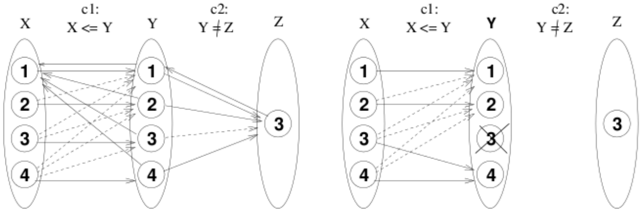
Last(x_i, v_i, x_j): pointer to store v_j, the smallest support in c_{ij}

```
function Revise2001(in x_i: variable; c_ij: constraint): Boolean ;
  begin
    CHANGE ← false;
    foreach v_i ∈ D(x_i) s.t. Last(x_i, v_i, x_j) ∉ D(x_j) do
      v_j ← smallest value in D(x_j) greater than Last(x_i, v_i, x_j) s.t.
      (v_i, v_j) ∈ c_ij;
      if v_j exists then Last(x_i, v_i, x_j) ← v_j;
      else
        remove v_i from D(x_i);
        CHANGE ← true;
    return CHANGE ;
  end
function AC3/GAC3(in X: set): Boolean ;
  begin
    /* initalisation */;
    Q ← {(x_i, c) | c ∈ C, x_i ∈ X(c)};
    /* propagation */;
    while Q ≠ ∅ do
      select and remove (x_i, c) from Q;
      if Revise(x_i, c) then
        if D(x_i) = ∅ then return false ;
        else Q ← Q ∪ {(x_j, c') | c' ∈ C ∧ c' ≠ c ∧ x_i, x_j ∈ X(c') ∧ j ≠ i};
    return true ;
  end
```

$O(ed^2)$ time
$O(ed)$ space

Reverse2001
Example

$$\mathcal{P} = \langle X = (x, y, z),\ \mathcal{DE} = \{D(x) = D(y) = \{1,2,3,4\}, D(z) = \{3\}\},$$
$$\mathcal{C} = \{C_1 \equiv x \leq y, C_2 \equiv y \neq z\}\}\rangle$$



$\texttt{Last}[x,1,y] = 1$ $\texttt{Last}[y,1,x] = 1$ $\texttt{Last}[y,1,z] = 3$
$\texttt{Last}[x,2,y] = 2$ $\texttt{Last}[y,2,x] = 1$ $\texttt{Last}[y,2,z] = 3$
$\texttt{Last}[x,3,y] = 3$ $\texttt{Last}[y,3,x] = 1$ $\texttt{Last}[y,3,z] = nil$
$\texttt{Last}[x,4,y] = 4$ $\texttt{Last}[y,4,x] = 1$ $\texttt{Last}[y,4,z] = 3$
$\texttt{Last}[z,3,y] = 1$

Limitation of Arc Consistency

Example

$$\langle x < y, y < z, z < x; x, y, z \in \{1..100000\}\rangle$$

is inconsistent.

Proof: Apply revise to $(x, x < y)$

$$\langle x < y, y < z, z < x; x \in \{1..99999\}, y, z \in \{1..100000\}\rangle,$$

ecc. we end in a fail.

- Disadvantage: large number of steps.
  Run time depends on the size of the domains!
- Note: we could prove fail by transitivity of $<$.
  $\rightsquigarrow$ Path consitency involves two constraints together

References

Bessiere C. (2006). **Constraint propagation**. In *Handbook of Constraint Programming*, edited by F. Rossi, P. van Beek, and T. Walsh, chap. 3. Elsevier. Also as Technical Report LIRMM 06020, March 2006.