G53CLP
Constraint Logic Programming

Dr Rong Qu

Constraint Propagation - Consistency Enforcing
Some Definitions

- Constraint satisfaction techniques
  - Constraint propagation
    - Basic idea: remove \textit{values} from domains and tighten constraints
    - Use the current information on constraints to derive new constraints
    - Can be used to fully solve the problems
  - Search techniques
    - ...
Some Definitions

- Constraint Graph (Constraint Network)
  - Binary CSPs
    - If all the constraints of a CSP affect two variables
  - The variables and constraints can be represented in a *constraint graph (constraint network)*
    - nodes represent variables
    - edges represent constraints
Arc Consistency – definition

- The arc \( \{x, y\} \) is arc consistent if
  - For each value \( a \) in the domain of \( x \)
  - There is a value \( b \) in the domain of \( y \)
  - Assignment \( x = a, y = b \) satisfy constraint \( C_{xy} \)

\[ \forall a \in X, \exists b \in Y, (a, b) \in C_{x, y} \]
Arc Consistency – definition

Question:

\{x, y\} is arc consistent \rightarrow \{y, x\} is arc consistent?

\forall a \in X, \exists b \in Y, (a, b) \in Cx, y

\forall b \in Y, \exists a \in X, (b, a) \in Cy, x
Arc Consistency – definition

A CSP is arc consistent iff every arc in its constraint graph is consistent

\[ \forall a \in X, \exists b \in Y, (a, b) \in Cx, y \]
Arc Consistency – example

x = \{\text{red, green, blue}\}

\begin{itemize}
  \item y = \{\text{red}\}
  \item z = \{\text{green, blue}\}
\end{itemize}

variables \(x, y\) and \(z\)

domains as given

constraints: not the same colour (value)

\{x, y\}, \{y, z\}, and \{x, z\}

\{y, x\}, \{z, y\}, and \{z, x\}
Arc Consistency – algorithms

- **Aim to**
  - Effectively remove many inconsistent labellings (values of variables) before the search, or at early stage of the search
  - Produce a CSP that is equivalent to the original one

- **So that**
  - The size of CSPs is reduced so as easier to solve
  - Constraint propagation, problem reduction ...
Arc Consistency – algorithms

- Implementation
  - repeatedly restrict the domains of the variables until the property holds true
  - examine each arc in turn and delete the values of the first node that does not match to any value of the second variable

- the deletion may change the situation of the whole graph, so arcs may need to be examined again
Arc Consistency – example

variables
x, y and z
domains
as given
constraints: not the same colour (value)
{x, y}, {y, z}, and {x, z}
{y, x}, {z, y}, and {z, x}
Arc Consistency – example

Variables: \( x, y \) and \( z \)

Domains: as given

Constraints: not the same colour (value)
- \( \{ x, y \}, \{ y, z \}, \) and \( \{ x, z \} \)
- \( \{ y, x \}, \{ z, y \}, \) and \( \{ z, x \} \)

\( x = \{ \text{red}, \text{green}, \text{blue} \} \)

\( y = \{ \text{red} \} \)

\( z = \{ \text{green}, \text{blue} \} \)

\( \{ x, y \} \) consistent

\( \{ x, z \} \) consistent

\( \{ y, z \} \) consistent?
Arc Consistency – AC-1

PROCEDURE AC-1(Z, D, C) // D: domains, C: constraints
BEGIN

Achieve node consistency

Construct the constraint list Q

REPEAT

Changed ← False;

FOR each item in Q

    Changed ← Revise(x→y, (Z,D,C)) // Revise deletes all values from Dx if they are not compatible with Dy; D may be reduced (arc consistency on {x,y})

UNTIL NOT Changed

Return (Z, D, C)

END

- Foundations of CS, Tsang, 2003
Arc Consistency – AC-1

Construct the constraint list \( Q \)

\[
Q \leftarrow \{ x \rightarrow y \mid C_{x,y} \in C \}
\]

\{x, y\}: every arc in the problem

If \( C_{x,y} \) is a constraint in the problem
both \( x \rightarrow y \) and \( y \rightarrow x \) are added to \( Q \)
Arc Consistency – AC-1

- AC-1 algorithm*
  - Not efficient to execute Revise
  - Removing any value will cause all items (even those not affected) in constraint list $Q$ to be re-examined
  - Very time consuming

*Mackworth (1977)
Arc Consistency – AC-3

- Improved: AC-3 algorithms*
  - Only those constraints which could be affected will be re-examined
  - If for arc \((x, y)\), any value \(v\) of \(x\) is removed
    - Domain of any third variable \(z (z, x)\) needs to be checked
      - Value \(v\) may support some values in \(z\)

*Mackworth (1977)
Arc Consistency – AC-3

PROCEDURE AC-3(Z, D, C)
BEGIN
  Achieve node consistency
  Construct the constraint list Q
  WHILE (Q is not empty) DO
    Delete item \( x \rightarrow y \) from \( Q \);
    IF Revise(\( x \rightarrow y, (Z,D,C) \)) THEN
      Update \( Q \) to include item \( z \rightarrow x \)
      //Include any 3\textsuperscript{rd} variable which is constrained by \( x \)
  Return (\( Z, D, C \))
END

- Foundations of CS, Tsang, 2003
Arc Consistency – AC-4

- AC-4*
  - Needs special data structure to remember individual pairs of variable-values
  - Avoid checking certain variables repeatedly
  - Use the ides of support

*Mohr & Henderson (1986)
Arc Consistency – AC-4

- AC-4
  - If a value \( v \) is removed from the domain of \( x \)
    - Not necessary to examine all binary constraints \( C_{x,y} \)
    - Ignore those values in \( D_y \) which do not reply on \( v \) for support
    - Those values in \( D_y \) rely on other values in \( D_x \) for support rather than \( v \)
Arc Consistency – algorithms

- AC-1 algorithm
  - Any value removed from a variable
  - All constraints in the constraint graph are re-checked
Arc Consistency – algorithms

- AC-3 algorithm
  - Any value removed from a variable
  - Only affected constraints are re-checked
Arc Consistency – algorithms

- AC-4 algorithm
  - Any value $v$ removed from a variable
  - Only those values that are supported of affected variables are re-checked
Arc Consistency – generalise

- Arc consistency is 2-consistency
  - Binary constraint
  - Example
    - Map colouring and 8-queen are arc consistent

- Node consistency is 1-consistency
  - Unary constraints
  - Example
    - $x = \{1, 2, 3\}$, $x$ must be even
Arc Consistency – generalise

- Node consistency algorithm
  - Simply remove inconsistent values from the domain of variables that do not satisfy the unary constraint
Arc Consistency – generalise

- Node consistency algorithm
  - Go through each variable
  - Check if the values satisfy the unary constraint of the variable
  - Delete all values which violate the constraints from the domains

- $a$: maximum size of domains; $n$: number of variables
  - $O(an)^*$
Arc Consistency – generalise

- *Big O
  - Notation in complexity theory
  - How the size of input affect the algorithm’s computational resource (time or memory)
  - Complexity of algorithms

- Self study: what is the complexity of AC-1 algorithm?
Arc Consistency – generalise

- Achieving arc and node consistency
  - Does not guarantee to find a solution, or
  - Does not prove there is a solution exist

- Extend the consistency check to more than two variables
  - Path consistency
Arc Consistency – generalise

- Path consistency
  - We can generalise arc consistency to problems concerning 3 variables
  - For values $a$ and $b$ for any two variables $x$ and $y$
  - There must be value $c$ for variable $z$, such that
  - Assignment $x = a$, $y = b$, $z = c$ satisfy the constraint $C_{xyz}$

- Path consistency algorithms
  - remove more inconsistencies
Arc Consistency – generalise

- Achieving path consistency
  - does not guarantee to find a solution, or
  - prove there is a solution exist

\[
\begin{align*}
\{1,2,3\} & \quad \{1,2,3\} \\
\{1,2,3\} & \quad \{1,2,3\} \\
\{1,2,3\} & \quad \{1,2,3\}
\end{align*}
\]

Constraint:
\[\neq\text{ for each edge}\]
$k$-consistency

- $k$-consistency
  - If one picks up $k$ variables and assign $(k-1)$ of them any values, the $k^{th}$ node can be assigned a value that is consistent with the previous values, satisfying the constraints between the $k^{th}$ and those $(k-1)$ variables

- $k$-consistent $\rightarrow (k-1)$-consistent?
$k$-consistency

- Strongly $k$-consistency
  - $j$-consistency for all $j \leq k$

That is

- A CSP is strong $k$-consistent if it is 1-, 2-, ... up to $k$-consistent
Consistency and Backtracking

- A solution can be found without backtracking if the constraint graph is strong $k$-consistent.
  - If a constraint graph is $j$-consistency $j < k$, backtracking still cannot be avoided.

- However
  - the algorithm of obtaining $k$-consistency is computational expensive.
Consistency and Backtracking

- A solution can be found without backtracking

1. If the constraint graph is a tree
   - Each node (except root) has at most one parent node
   - Each node may have zero or more child nodes

2. If the constraint graph is both arc-consistent and node-consistent
Consistency and Backtracking

- **Arc consistency vs. $k$-consistency**
  - Achieving stronger consistency checks
    - takes more time
    - reduces more branches

- **In practice**
  - We can find a smallest $k$, problem can be solved without backtracking
  - Deciding an appropriate level of consistency is an empirical science
Summary

- Constraint propagation algorithms
  - Arc consistency
    - Definition
    - Example
    - Arc consistency algorithms (AC-1, AC-3, AC-4)
  - $k$-consistency generalisation
    - Strong $k$-consistency
    - Consistency vs. backtracking free