Consistency Enforcing and Constraint Propagation: Node and Arc Consistency
Summary

- Node consistency and Arc Consistency
Solution Techniques for Constraint Network

Solving Constraint Networks

- **Inference:**
  - Infer new constraints based on existing ones
  - Eliminate values from variables that do not meet constraints

- **Search:**
  - Look for a solution trying different values of variables
  - Backtracking and similar approaches
  - Local search
Backtracking

**general ideas**

- Choose a variable $x$
- list its domain values
- for each value add a constraint $x = v$ and recursively evaluate the rest of the problem
Local Consistency

**general ideas**

- Partial assignments can lead to constraint violations
  - We can evaluate a constraint as soon as all variables in its scope are assigned
- We can backtrack as soon as a constraint is not locally consistent
Inference and constraint propagation

Example (inference)

- Variables: \{A, B, C\}
- Domain: \{0, 1\} or true,false
- Constraint: \{A \rightarrow B, C \rightarrow A, C\}
- Propagating the constraints we can infer \{A, B\}
- Similar reasoning if we know \{\neg B\} holds
Consistency

Consistency Methods

- Approximation of inference
  - arc, path and i-consistency
- Generate tighter networks
- Partial assignments can be discarded earlier
Example

Example (Consistency)

- n-Queen problem
- Minimal network is tighter than original network
- On minimal network finding the solution is easier
Tightness and search space

restricting the searchspace

- Given two equivalent network $\mathcal{R}$ and $\mathcal{R}'$
- if $\mathcal{R}' \subset \mathcal{R}$ ($\mathcal{R}'$ is tighter than $\mathcal{R}$)
- then searching for a solution on $\mathcal{R}'$ is more efficient than searching on $\mathcal{R}$
- $\mathcal{R}'$ has a smaller search space
Complete inference

finding solution with no dead end

- We can deduce constraints until:
  - an inconsistency is found
  - or we can derive a solution with depth-first and no backtracking
- but we might need to introduce an exponential number of constraint
- usually it is preferable to introduce a bounded amount of constraints
Consistency approaches

- Given a partial solution of length \( i - 1 \) we extend the solution to one more variable
- Consistency enforcing:
  - any partial solution of a subnetwork extensible to a surrounding network
  - size of the subnetwork defines different approaches
- Arc-Consistency: from 1 variable to 2
- Path-Consistency: from 2 variables to 3
- \( l \)-consistency: from \( i-1 \) to \( i \)
Extending solutions

consistency and solution extension

- i-1 consistency:
  - for any legal value for i-1 variables
  - we can find a legal value for any other connected variables.

- A network that is i-consistent for \( i = 1, \cdots, n \) is globally consistent
Consistency and computational issues

### consistency and computation

- The higher is $i$ the better a search algorithm will behave.
- Time and space cost to ensure $i$-consistency is exponential in $i$.
- Trade-off addressed with experimental evaluation.
Example (consistency)

- Variables: \{X, Y, T, Z\}, \( D_i = 1, 2, 3 \)
- Constraints: \( X < Y, Y = Z, T < Z, X < T, X < 4 \)
Node consistency

- Variable $x_i$, Domain $D_i$
- $x_i$ is node consistent if every value of its domain satisfy every unary constraint
- $\forall v \in D_i \forall C = \{< x >, R_{x_i}\} a \in R_{x_i}$
Constraint Propagation

- We modify the constraint network so that:
  - local consistency is satisfied (enforcing consistency)
  - solutions do not change (maintaining equivalence)
Constraint propagation for node consistency

CP for node consistency

- If a variable $x_i$ is not node consistent:
  - remove all values from $D_i$ that do not satisfy all unary constraints
  - $D'_i = D_i \setminus \{v | \exists C = \{< x_i >, R_{x_i} \} \land v \not\in R_{x_i} \}$

- $D'_i$ contains only values that satisfy all unary constraints (enforcing consistency)

- all removed values could not be part of any solution (maintaining equivalence)
Arc Consistency

Example (Arc consistency)

- Variables $x, y$ with domains $D_x = D_y = \{1, 2, 3\}$.
- $C = \{< x, y >, R_{x,y} = x < y\}$
- $D_x$ and $D_y$ are not arc consistent with $R_{x,y}$
- $D'_x = \{1, 2\}, D'_y = \{2, 3\}$ are arc consistent
- $D''_x = \{1\}, D''_y = \{2\}$ are arc consistent but...
Constraint propagation for arc consistency

CP for arc consistency

- If a variable $x_i$ is not arc consistent w.r.t. $x_j$:
  - remove all values from $D_i$ that does not have a matching value in $x_j$
- $D_i'$ contains only values that satisfy binary constraints (enforcing consistency)
- all removed values could not be part of any solution (maintaining equivalence)
Arc Consistency

- Network $\mathcal{R} = \langle X, D, C \rangle$
- $x_i, x_j \in X$
- $x_i$ arc consistent w.r.t. $x_j$ iff
  - $\forall a_i \in D_i \exists a_j \in D_j | (a_i, a_j) \in R_{x_i, x_j}$
- $R_{x_i, x_j}$ is arc consistent iff $x_i$ arc consistent w.r.t. $x_j$ and $x_j$ arc consistent w.r.t. $x_i$
- $\mathcal{R}$ is arc consistent iff all its constraints are arc consistent
Revise Procedure

**Algorithm 1** Revise(((\(x_i\)), \(x_j\)))

**Require:** \(R_{x_i,x_j}, D_i, D_j\)

**Ensure:** \(D_i\) such that \(x_i\) is arc consistent w.r.t. \(x_j\)

for all \(a_i \in D_i\) do
  if \(\neg \exists a_j \in D_j | (a_i, a_j) \in R_{x_i,x_j}\) then
    delete \(a_i\) from \(D_i\);
  end if
end for

Equivalent to \(R_{xy} \leftarrow R_{xy} \cap \pi_{xy}(R_{xz}D_zR_{zy})\)
Revise Procedure for Networks

**Revise for Network**

```
for all \( x_i \in X \) do
  for all \( R_{x_i,x_j} \in C \) do
    Revise((\( x_i \),\( x_j \)));
    Revise((\( x_j \),\( x_i \)));
  end for
end for
```

- This algorithm does not work!
- Revising arc consistency on a variable might make another variable not-arc consistent
Revising Networks

Example (Revise for Network)

- Variables $x, y, z$ with domains
  $D_x = \{0, 1, 2, 3\}, D_y = \{1, 2\}, D_z = \{0, 1, 2\}$.

- $C_{x,y} = \{< x, y >, R_{x,y} = x < y\}$,
  $C_{z,x} = \{< z, x >, R_{z,x} = z < x\}$
An algorithm that does work!

AC-1

Require: \( \mathcal{R} = \langle X, D, C \rangle \)
Ensure: \( \mathcal{R}' \) the loosest arc consistent network for \( \mathcal{R} \)

repeat
  for all Pairs \( x_i, x_j \) that participate in a constraint do
    Revise((\( x_i \),\( x_j \));
    Revise((\( x_j \),\( x_i \));
  end for
until no domain is changed

This algorithm does work!
Inconsistent Networks

AC-1 always terminate

- If we do not change any domain then we stop and $\mathcal{R}$ is AC
- If we remove a value we make at least one domain smaller
- If a domain is empty the network is inconsistent: we cannot find any solution
Example

Variables: \( \{x, y, z\} \), domains \( D_x = D_y = D_z = \{1, 2, 3\} \)

Constraints \( \{x < y, y < z, z < x\} \)

apply AC-1
Computational complexity of AC-1

AC-1 is $O(nek^3)$

- $n$: nodes, $e$: edges, $k$: max number of values of a domain
- each cycle: $ek^2$ operations
- worst case we delete 1 element from one domain at each cycle
- we can have at most $nk$ cycles
AC-3

Require: \( R = \langle X, D, C \rangle \)

Ensure: \( R' \) the loosest arc consistent network for \( R \)

for all every pairs \((x_i, x_j)\) that participate in a constraint \( R_{x_i, x_j} \in \mathcal{R} \) do
  \( Q \leftarrow Q \cup \{(x_i, x_j), (x_j, x_i)\} \)
end for

while \( Q \neq \{\} \) do
  pop \((x_i, x_j)\) from \( Q \)
  \( \text{REVISE}((x_i), x_j) \)
  if \( D_i \) changed then
    \( Q \leftarrow Q \cup \{(x_k, x_i), k \neq i, k \neq j\} \)
  end if
end while
AC-3 Example

Example

AC-3

- Variables \( x, y, z \), domains \( D_x = D_z = \{2, 5\}, D_y = \{2, 4\} \)
- Constraints: \( R_{x,z} = \{a_x, a_z, |(a_x \mod a_z = 0)\} \)
  \( R_{y,z} = \{a_y, a_z, |(a_y \mod a_z = 0)\} \)
- Run AC-3
AC-3 Computational Complexity

**Comp. Complexity**

- $O(ek^3)$
- Revise for each couple is $O(k^2)$
- worst case we evaluate $2ek$
- because we can put back each couple at most $k$ times
Distributed Arc Consistency

AC-1 can be distributed

- Each node a computer, they can send messages to neighbours
- Each computer knows only its direct neighbours and shared constraints
- Revise = $D_i \leftarrow D_i \cap \pi_i(R_{ij} \Join D_j)$
- Node $j$ sends a message to $i$: $h_{j\rightarrow i} = \pi_i(R_{ij} \Join D_j)$
- Node $i$ computes $D_i \leftarrow D_i \cap h_{j\leftarrow i}$ for each message received by its neighbours
Consistency and Arc Consistency

Empty Domain and Arc Consistency

- Arc consistency + empty domain $\rightarrow$ inconsistent problem
- Arc consistent + all domains are not empty $\not\rightarrow$ consistent problem
- Arc consistency is not complete
  - It checks only single (binary) constraints and single domain constraint
Example: incompleteness of AC for consistency

Example

Binary Graph Colouring

- Variables: \( x, y, z \) Domain: \( D_i = \{R, Y\} \)
- Constraints: \( x \neq y, y \neq z, z \neq x \)
Exercise

Exercise 1

Consider the following network:

- Variables: \{X, Y, Z, W\}, Domain \(D_i = \{0, 1, 2\}\)
- Constraints: \(X < Y\), \(Z = X\), \(Z < W\), \(W < Y\)

describe an execution of AC-3. Is the resulting network arc consistent? Is the resulting network consistent? Motivate your answers.