Problem Reduction for Constraint Satisfaction

Motivations:

1. Reduce problem to easier problems
2. Detect unsatisfiability

Problem Reduction Overview

- Node-consistency (NC)
- Arc-consistency (AC)
- Path-consistency (PC)
- (strong) *k*-consistency
- Directional Arc-consistency (DAC)
- Other Consistency Properties (not covered)
  - Directional PC, Adaptive consistency, ...

Node-consistency

- Principle of NC:
  - If unary constraint C_x exists for variable *x*
  - Remove any value *v* from D_x if *v* does not satisfy C_x
- E.g. D_x: days of the week;
  C_x: this job must be done in weekdays
- Simple pre-processing strategy
- Cheap to compute: O(*n*)

Node-consistency (NC), Definition

- Notation: C_x – constraint on variable *x*
- A CSP is ***node-consistent*** iff for all variables all values in its domain satisfy the constraints on that variable
- Note:
  - The statement "***P implies Q***" is ***true*** if P is ***false***
  - Hence if all domains are empty, the problem is *node-consistent* (though *unsatisfiable*)

Arc-consistency (AC), Definition

- Notation C*x*,*y* : constraint on variables *x* & *y*
- An ***arc*** (*x*, *y*) is ***arc-consistent*** iff
  - for every value *a* in the D*x* which satisfies C_x,
  - there exists at least one value *b* in D*y*
  - such that <*y*, *b*> is compatible with <*x*, *a*>

  In this case, we say <*y*, *b*> *supports* <*x*, *a*>
- A ***problem*** is ***arc-consistent*** iff every arc in its constraint graph is arc-consistent

Maintaining AC, Example

- Variables: *x*, *y*, *z*
- Domains: {1, 2, 3, 4}
- Constraints:

  x < *y* ; *y* < *z*

x	1	2	~~3~~	~~4~~
y	~~1~~	2	3	~~4~~
z	~~1~~	~~2~~	3	4

- x<y means <x,4> not supported by y and <y,1> not supported by x
- y<z means <y,4> not supported by z and <z,1> & <z,2> not supported by y
- Re-check x<y would delete <x,3> as now (with <y,4> gone) it has no support from y

Maintaining Arc-consistency

- Principle of Arc-consistency:
  - If any variable *y* has no value to support <*x*, *v*>
  - Then remove *v* from D_x
- Naïve algorithms:
  - Repeat constraint propagation
  - Until no more values can be removed
- Advanced algorithms: ***record supports***
  - In order to focus on what to propagate
  - Complex data structure required

Arc-consistency Algorithms

- AC maintenance, lots of research
  - AC-1: Naïve but good for parallel processing
  - AC-4: Complex, using the concept of support
- Complexity manageable: $O(a^3ne)$ to $O(a^2e)$
- MAC: seen to be generally practical
  - Keep data structure to record *supports* permanently
  - No need to maintain AC from scratch

Algorithm AC-4

- Efficient algorithm for maintaining AC
  - Use data structure to reduce number of checks
- *Support* $S_{<x,a>}$ records the set of labels that <x,a> supports
  - When <x,a> is removed, all labels in $S_{<x,a>}$ lose support from <x,a>; they need re-examination
- *Counter*[(x, y), a] records the number of supports that y provides to <x,a>
  - When counter reduced to 0, remove <x,a>
- *M*[x,a] = 1 if <x,a> has been rejected; 0 otherwise

AC-4 Data Structure, Example

- Variables: *x*, *y*, *z*
- Domains: {1, 2, 3, 4}
- Constraints: *x* < *y* ; *y* < *z*
- Data Structure:
  - $S_{<x,3>}$ = {<y,4>}
  - $S_{<x,4>}$ ={}
  - $S_{<y,2>}$ = {<x,1>, <z,3>, <z,4>}
  - $S_{<y,4>}$ = {<x,1>, <x,2>, <x,3>}
  - …
- When <y,4> is removed:
  - $S_{<y,4>}$ points to <x,1>, <x,2>, <x,3>
  - Counter[(x,y),3] reduced to 0
  - <x,3> is removed

x	1	2	3	4
y	1	2	3	4
z	1	2	3	4

Counter[(x,y),1] = 3
Counter[(x,y),2] = 2
Counter[(x,y),3] = 1
Counter[(x,y),4] = 0
Counter[(y,x),4] = ? exercise

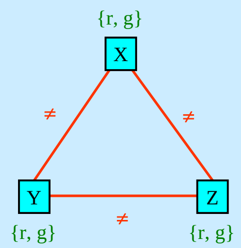
Remarks on AC

- AC has been maintained in this problem
- But it doesn't mean that all combinations of the remaining values are compatible with each other.
- All it means is every remaining value is supported by at least one value in every other variable

x	1	2	~~3~~	~~4~~
y	~~1~~	2	3	~~4~~
z	~~1~~	~~2~~	3	4

Example: AC but Unsatisfiable

- This problem is AC
- But unsatisfiable
- Can we detect unsatisfiability?



Path-consistency

- Principle:
  - Tightening constraints
  - By ***constraint composition***
- Algorithms: PC-1, …, PC-4
- Complexity manageable: $O(a^3n^3)$
- Is effort justifiable?

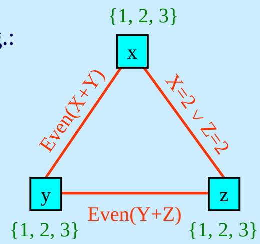
Path-consistency (PC), Definition

- Notation C_S – the set of all relevant constraints on the set of variables S
- An ***path*** (*x*, *y*, *z*) is ***path-consistent*** iff
  - for every 2-compound label (<*x*,*a*><*z*,*c*>) that satisfies $C_{\{x,z\}}$,
  - there exists a value *b* in D*y*
  - such that (<*x*,*a*><*y*,*b*><*z*,*c*>) satisfies $C_{\{x,y,z\}}$
- A ***problem*** is ***path-consistent*** iff every path in its constraint graph is path-consistent

Binary Constraint Representation

- A Binary constraint may be represented by a matrix, e.g.:
- $D_x = D_y = D_z = \{1, 2, 3\}$
- C_{xy} : *X* + *Y* must be even
- C_{xz} : at least one of *X* and *Z* must be equal to 2

$$C_{x,y} = C_{y,z} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \quad C_{x,z} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

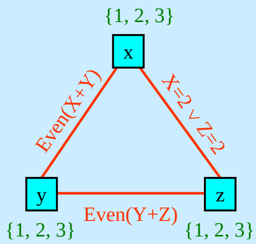


Constraint Composition

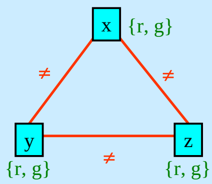
- Constraint C_{xy} & C_{yz} could tighten C_{xz}
- $C_{xz} = C_{xz} \wedge C_{xy} * C_{yz}$

$$= \begin{vmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{vmatrix} \wedge \begin{vmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{vmatrix} * \begin{vmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{vmatrix}$$

$$= \begin{vmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{vmatrix} \wedge \begin{vmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{vmatrix} = \begin{vmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{vmatrix}$$



PC Detecting Unsatisfiability



$$C_{x,y} = C_{y,z} = C_{x,z} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

- Given (<x,r><y,g>), which satisfies $C_{x,y}$
- It has no compatible value in z
- Hence not PC

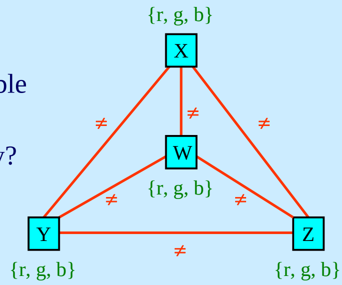
$$C_{x,y} = C_{x,y} \wedge C_{x,z} * C_{z,y}$$

$$= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \wedge \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \wedge \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Example: PC but Unsatisfiable

- Problem is PC
- But unsatisfiable
- Can we detect unsatisfiability?



k-consistency, Definition

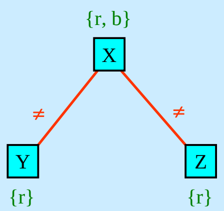
- A problem is ***k-consistent*** iff:
  - For all (k-1)-compound labels
    ($<x_1, v_1>$ $<x_2, v_2>$ … $<x_{k-1}, v_{k-1}>$)
  - that satisfies all constraints on $x_1, x_2, \ldots, x_{k-1}$
  - For every k^{th} variable x_k
  - There exists a value v_k such that all constraints on $x_1, x_2, \ldots, x_{k-1}, x_k$ are satisfied
- *Note: this is a definition, not every problem is k-C*

Special Cases of k-Consistency

- AC ≡ 2-Consistency
- For binary problems, PC ≡ 3-Consistency

Strong *k*-consistency

- A problem is ***strong k-consistent*** iff
  - it is 1-C, 2-C, …, *k*-C
- Complexity grows with *k*



This problem is 3-C but not 2-C!

Directional-arc-consistency

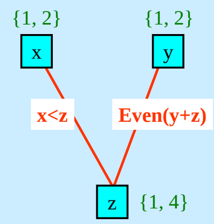
- Observations:
  - Variables ordering is often static in a search
  - AC is bi-directional
- Principle:
  - Given an ordering of the variables
  - only remove v_x from D_x when
  - support does not exist for any *future* variable *y*
- Practical, may sit in other algorithms

DAC Algorithm

- Given ordering of n variables, $x_1, x_2, \ldots, x_n$
- For $k = n$ to 1 by –1 DO
  - For each variable x_j where $j<k$ & $C_{k,j}$ DO
    - Remove from domain of x_j any value not supported by x_k
- Each constraint is checked once only
  - Normally cheaper than maintaining AC

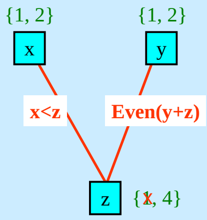
Maintaining DAC, Example 1

- Under order (x→y→z)
- Examine y→z (even sum)
  - <y,1> supported by <z,1>
  - <y,2> supported by <z,4>
- Examine x→z (x<z)
  - <x,1> supported by <z,4>
  - <x,2> supported by <z,4>
- Examine x→y
  - No constraint



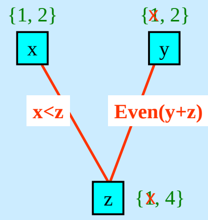
Maintaining DAC, Example 2

- Under order (z→y→x)
- Examine y→x
  No constraint
- Examine z→x (x<z)
  <z,1> not supported by x
  <z,4> supported by <x,1> and <x,2>
- Examine z→y (even sum)
  (<z,1> already deleted)
  <z,4> supported by <y,2>



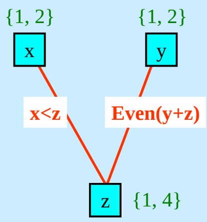
Maintaining DAC, Example 3

- Under order (y→z→x)
- Examine z→x (x<z)
  <z,1> not supported by x
  <z,4> supported by <x,1>, <x,2>
- Examine y→x (no constraint)
- Examine y→z (even sum)
  <y,1> not supported by z
  – as <z,1> has been removed
  <y,2> supported by <z,4>



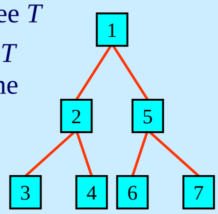
Maintaining DAC, Remarks

- To maintain AC, one must
  - Remove <z,1> (no support from x)
  - Remove <y,1> (no support from z after <z,1> is removed)
- Maintaining DAC under both (x→y→z) and (z→y→x) does not obtain AC
  - Only <z,1> is removed
- Maintaining DAC under the right order (y→z→x) matters



Tree-search Algorithm

- If the constraint graph is a tree *T*
- Ω: Ordering the variables in *T* such that parents always come before children
- Maintain DAC under Ω
- Then search under Ω
  - Such search is guaranteed ***backtrack-free***!



One possible Ω

Problem Reduction, Summary

- Aim: reduce problem to one that is easier to solve, or detect dead-ends
- Concept is simple, procedures could be complex
- General Concept: k-consistency
  - NC≡1-C, AC≡2-C, PC≡3-C for binary CSPs
- NC & AC potentially reduce domains
- PC potentially tightens binary constraints
- k-C for k>2 potentially tightens k-1 constraints