Title: Synthesizing Constraint Expressions

Author: E.C. Freuder

Journal: Communications of the ACM

Volume: 21, No 11

Year: 1978

Foundations of Constraint Satisfaction

CSCE 990-005, Fall 2000

Berthe Y. Choueiry (Shu-we-ri)

Ferguson Hall, Room 104

`choueiry@cse.unl.edu`, Tel: (402)472-5444

Contributions

- Synthesis of **constraints**: constraints networks
- Levels of consistency
- Constraint synthesis to $\left\{ \begin{array}{l} \text{replace} \\ \text{complement} \end{array} \right\}$ backtrack search
- Two mechanisms: $\left\{ \begin{array}{l} \text{constraint synthesis} \\ \text{constraint propagation (by projection)} \end{array} \right.$

$\longrightarrow$ rule out inconsistencies in stages to control combinatorial explosion

$\longrightarrow$ find out all solutions

(Constraint = relation = predicate = property = feature = attribute = etc.)

Observation I

Given $\mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$, the goal is to find the **global** constraint that satisfies simultaneously all constraints in $\mathcal{C}$

Given 2^n **local** constraints:

$$
\begin{aligned}
C_{V_1} &= \{\,(a_1)\} \subseteq D_{V_1} \\
&\vdots \\
C_{V_i,V_j} &= \{\,(a_i, a_j)\} \subseteq D_{V_i} \times D_{V_j} \\
&\vdots \\
C_{V_k,V_l,V_m} &= \{\,(a_k, a_l, a_m)\} \subseteq D_{V_k} \times D_{V_l} \times D_{V_m} \\
&\vdots
\end{aligned}
$$

Synthesize **global** constraint:

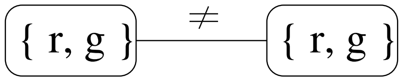
$$C_{V_1,V_2,\ldots,V_n} = \{\,(a_1, a_2, \ldots, a_n)\} \subseteq D_{V_1} \times D_{V_2} \ldots \times D_{V_n}$$

Observation II

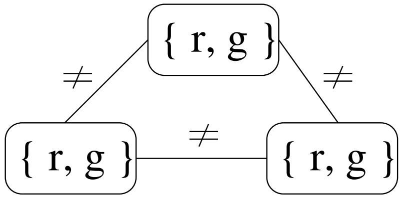
Consistency algorithms (e.g., Waltz algorithm, node, arc, path consistency) propagate the effects of **local** constraints **globally**

\+ important reduction of possibilities

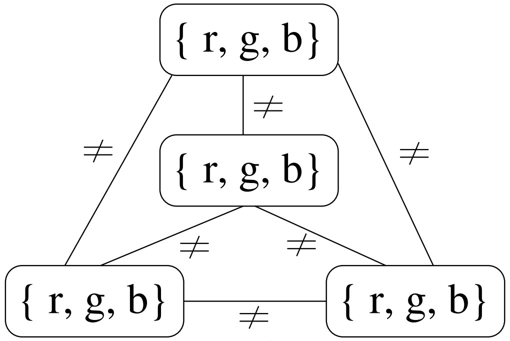
– do not necessarily $\begin{cases}\text{terminate with solutions}\\ \text{detect the non-existence of solutions}\end{cases}$



- arc-consistent
- not all combinations are acceptable



- arc-consistent
- insoluble



- arc-consistent
- path-consistent
- insoluble

$\longrightarrow$ search is still required

Idea

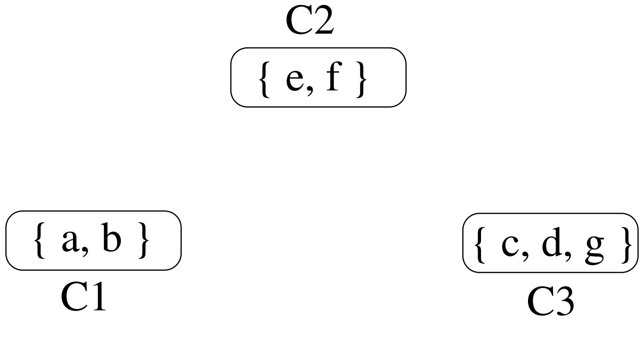
- Synthesize global constraint
- Use propagation to control (reduce) combinatorial complexity

Example

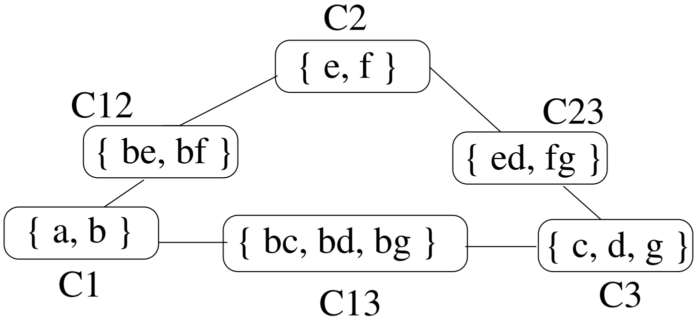
Variables:	$X_1,\ X_2,\ X_3$	
Constraints:	$C_1 = \{(a), (b)\}$	on X_1
	$C_2 = \{(e), (f)\}$	on X_2
	$C_3 = \{(c), (d), (g)\}$	on X_3
	$C_{12} = \{(be), (bf)\}$	on X_1, X_2
	$C_{13} = \{(bc), (bd), (bg)\}$	on X_1, X_3
	$C_{23} = \{(ed), (fg)\}$	on X_2, X_3

Start

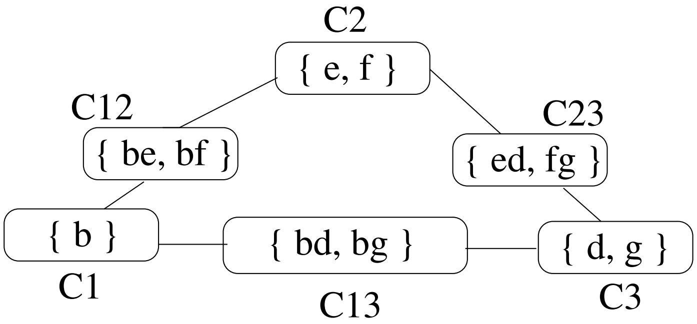
Step 1: Create nodes for unary constraints:



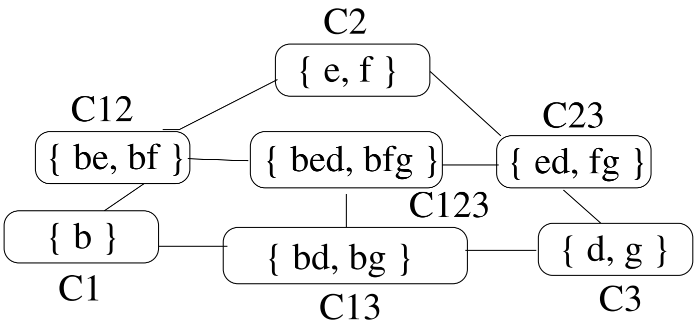
Step 2: Create nodes for binary constraints:



Propagate binary constraints



Step 3: Create nodes for ternary constraints:



Propagate ternary constraints on binary ones

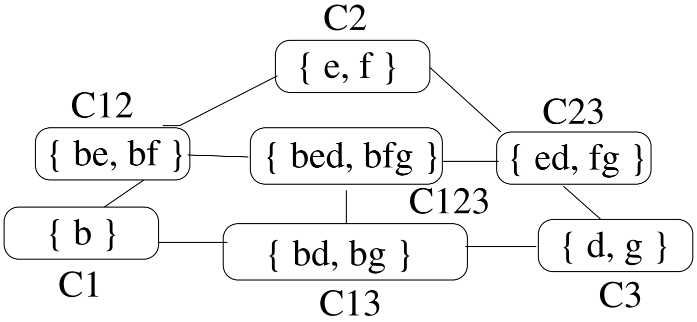
Propagate binary constraints on unary ones

Done

Notations and definitions (I)

$$
\begin{array}{lcl}
\mathcal{V} & = & \{X_1, X_2, \ldots, X_n\} \\
\text{universes} & = & \{U_1, U_2, \ldots, U_n\} \\
I & = & \{1, 2, \ldots, n\} \\
J & \subseteq & I \\
X_J & = & \{X_j\}_{j \in J} \qquad (\subseteq \mathcal{V})
\end{array}
$$

nonconstraint :	universal constraint
instantiation of X_i :	$X_i \leftarrow a_i \in U_i$
instantiation of X_J :	$X_J \leftarrow a_J = \{a_j\}_{j \in J}$
constraint C_J on X_J :	$\{a_J\}$ = set of inst. of X_J
a_J satisfies $C_J \Rightarrow$	$a_J \in C_J$

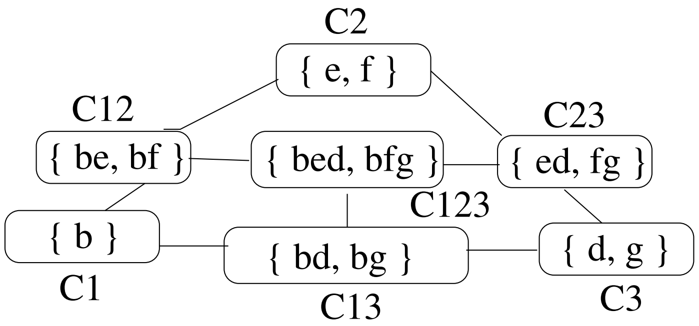


Notations and definitions (II)

Given C_H $(H \subseteq J)$,

a_J restricted to $C_H = \{a_j \in a_J\}_{j \in H}$

a_J satisfies C_H if "a_J restricted to C_H" $\in C_H$



Constraint network of order k in n variables $(k \leq n)$:

- constraints called **nodes**, N_J, for **each** $J \subseteq I$ and $|J| \leq k$
- links exist between N_J and N_H if $(H \subseteq J) \wedge (|H| = |J| - 1)$

$\rightarrow$ Full network: $k = n$

$\rightarrow$ Order of node N_J is the size of J

Notations and definitions (III)

Locally propagate N_J to a neighbor N_H

→ remove from N_H all a_H that do not satisfy N_J

Globally propagate N_J through a neighbor N_H

→ locally propagate N_J to N_H

→ if N_H modified, globally propagate N_H through all its neighbors except N_J

Propagate N_J

→ globally propagate N_J through all its neighbors

Relaxed network: when propagation causes no change

Synthesis algorithm

Step 1:

Build a network with nodes N_J

corresponding to all unary constraints

$k \leftarrow 1$

Step k+1:

For all $J \subseteq I$ with $|J| = k + 1$

Add a node N_J to the network

Link N_J to all N_H, $H \subseteq J$ and $|H| = k$

Locally propagate each N_H to N_J

Propagate N_J

$\longrightarrow$ The algorithm run n steps

$\longrightarrow$ results in a full constraint network

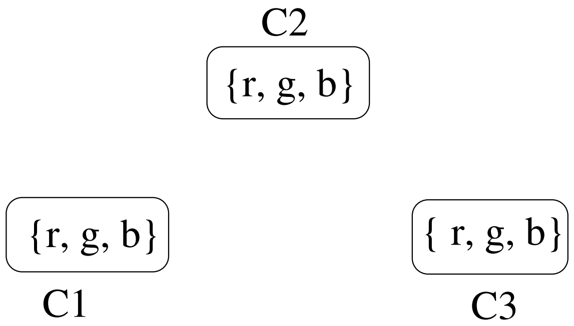
$\longrightarrow$ Proceeding by steps helps containing the combinatorial explosion (loop: add node, propagate)

$\longrightarrow$ Watch out for early terminate: empty node

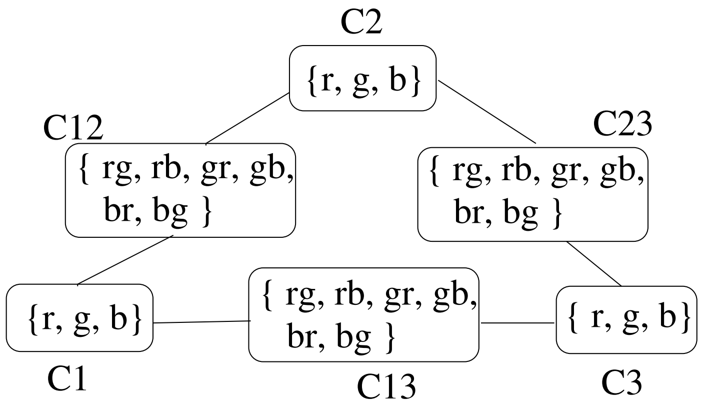
$\longrightarrow$ In the final network, every member of every N_J appears in some solution

Example: 3-variables, 3-colors

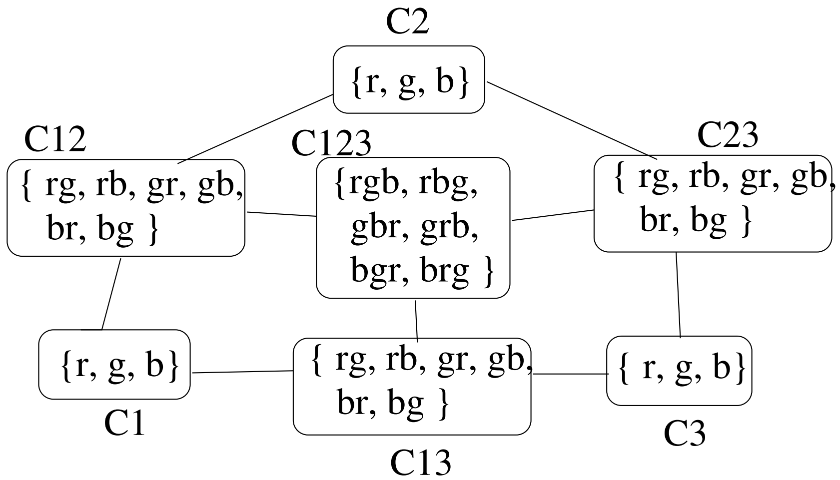
Step 1:



Step 2:

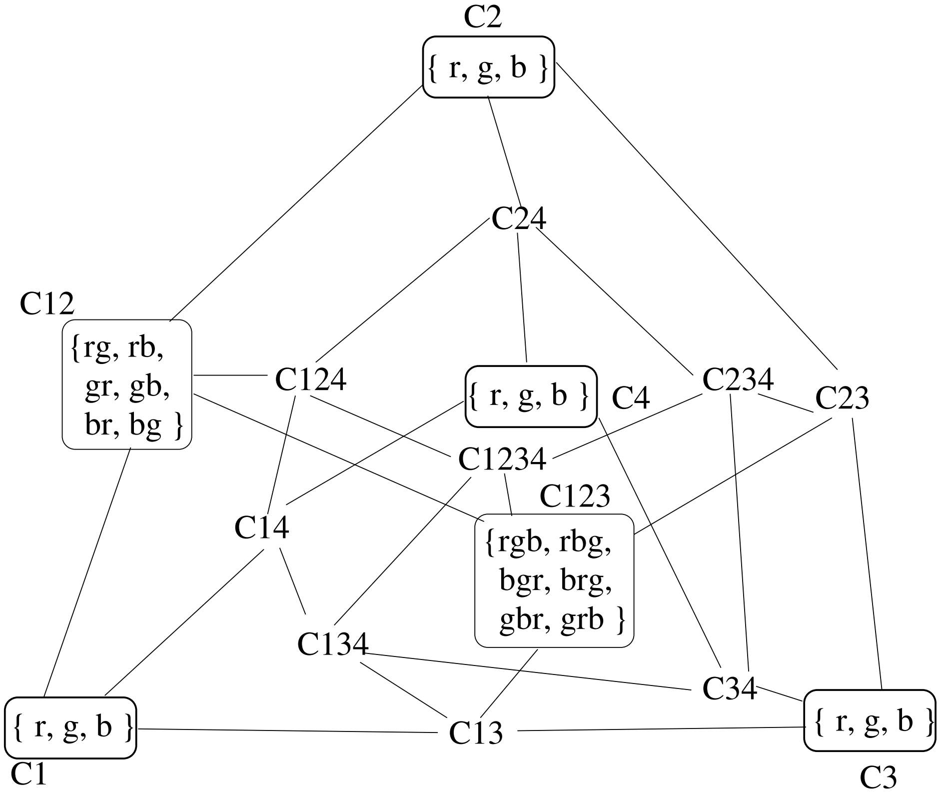


Step 3:



Example: 4-variables, 3-colors

Step 4:



C1234 = { rgbr, rgbg, rgbb, rbgr, rbgg, rbrg, etc. }

⟶ local propagation of ternary constraints
into C1234 empties C1234

⟶ problem is insoluble

More definitions:

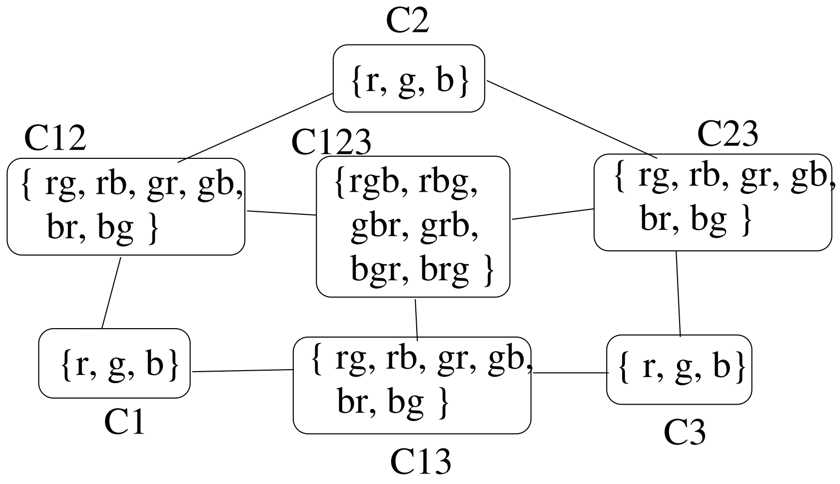
$C = \wedge_{J \in 2^I} C_J$: a constraint expression of order n

- k-satisfiability
- k-compatibility
- k-completeness
- k-consistency

Satisfiability

Given $|J| = k \leq n$, a_J <u>k-satisfies</u> C, if a_J satisfies the constraints C_H for all $H \subseteq J$

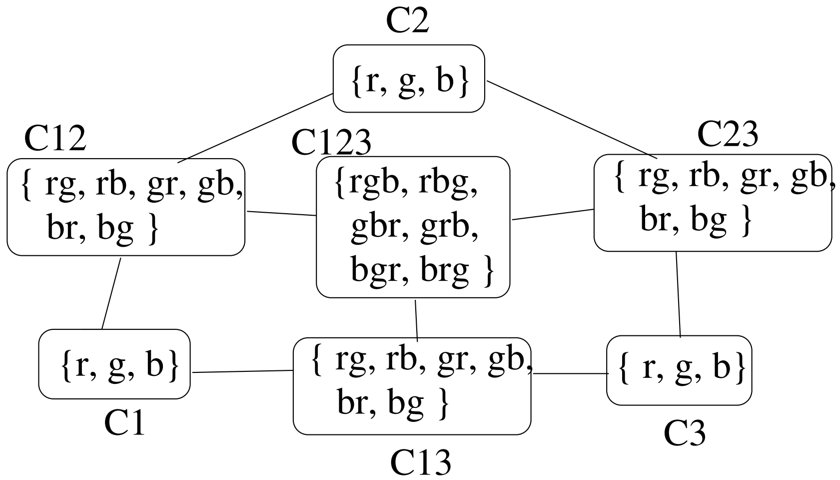
C is <u>k-satisfiable</u> if for all $J \subseteq I$ such that $|J| = k$, $\exists\, a_J$ that k-satisfies C



Compatibility

A node N_J of order k is <u>k-compatible</u> with C, if all members of N_J k-satisfy C

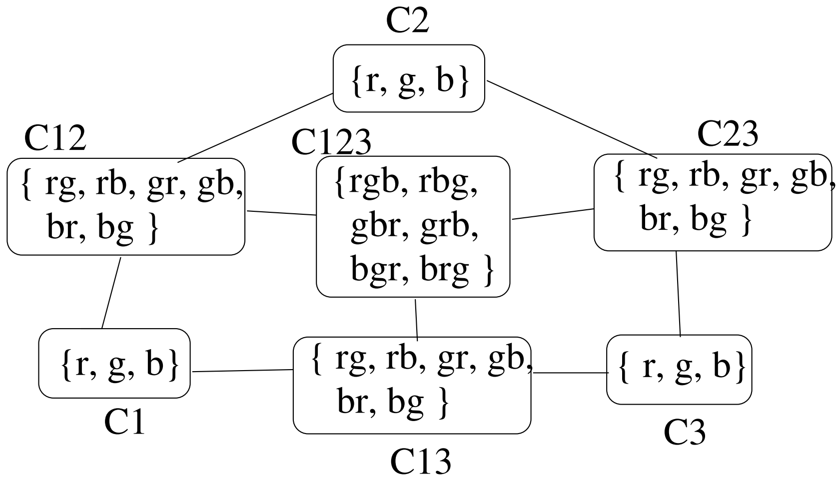
A constraint network of order $\geq k$ is <u>k-compatible</u> with C, if all nodes N_J are k-compatible with C



Completeness

A node N_J of order k is k-complete for C, if any a_J that k-satisfies C is member of N_J

A constraint network is k-complete for C, if every node of order k is k-complete for C



Consistency

A constraint network of order $\geq k$ in n variables is <u>k-consistent</u>:

if for any (k-1) variables, X_H

any instantiation a_H that satisfies X_H

and any choice of a k^{th} variable X_i

there exists a value for X_i that combined to a_H, k-satisfy $X_H \cup \{X_i\}$

1-consistency is node consistency

2-consistency is arc consistency

3-consistency is path consistency

Results

- At Step K, the synthesis algorithm achieves k-compatibility and k-completeness
- After K steps, the synthesis algorithm produces a network that is j-consistent for $j \leq k$
- The relaxation of the network is compatible and complete with respect to C
- N_I, the node of order I, contains the solutions. If N_I has no possible instantiations, C is not satisfiable, the problem is not soluble

Summary

- Constraints network: nodes are constraints
- Constraint synthesis by creation of higher-order nodes
- Synthesis algorithm: in stages, synthesize constraints and propagate their effects
- Use of lower-order consistencies to achieve higher-order consistency
- Definitions: k-satisfiability, k-compatibility, k-completeness, k-consistency. Introduced to justify theoretically the synthesis algorithm

$\longrightarrow$ higher-order constraint + arc consistency = higher-order consistency

$\longrightarrow$ Interesting perspective, but no complexity/space requirements considerations