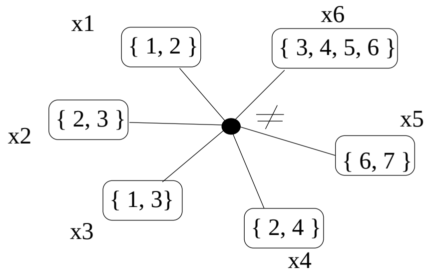
Title: A Filtering Algorithm for Constraints of Difference in CSPs
Author: J.-Ch. Régin
Proc.: AAAI
Pages: 362–367
Year: 1994

Theory and Practice of Constraint Satisfaction
CSCE 990-05, Fall 1999
www.cse.unl.edu/˜choueiry/CSE990-05/

All-diffs constraint

Constraint: C
Variables: $X_C = \{x_1, x_2, \ldots, x_6\}$



Context: finite CSPs

Goal: efficiency of arc consistency

Focus: All-diff constraints
(as for monotonic, functional, and non-functional constraints)

Result: efficient algorithm $\begin{cases} \text{Space} : \mathcal{O}(pd) \\ \text{Time} : \mathcal{O}(p^2 d^2) \end{cases}$
p: #vars, d: max domain size

Application: used in RESYN for subgraph isomorphism
(plan synthesis in organic chemistry)

Contributions

- An algorithm to establish arc consistency in an all-diff constraint
  $\rightarrow$ efficient
  $\rightarrow$ powerful pruning

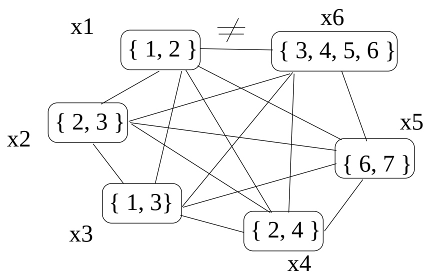
- An algorithm to propagate deletions among several all-diff constraints

- Illustration on the zebra problem

Why?

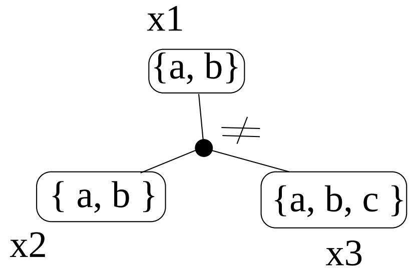
- GAC4 handles n-ary constraints
  $\rightarrow$ good pruning power
  $\rightarrow$ quite expensive:
    depends on size and number
    of all admissible tuples $= \frac{d!}{(d-p)!}$
    p: #vars, d: max domain size

- Replace n-ary by a set of binary constraints, then use AC-3 or AC-4
  $\rightarrow$ cheap
  $\rightarrow$ bad pruning



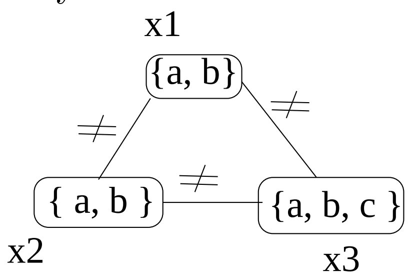
Example

- n-ary constraint



GAC4: rules out a, b for x_3

- Set of binary constraints



AC-3/4 ends with no filtering

Notations

CSP: $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$

$C \in \mathcal{C}$ defined on $X_C = \{x_{i_1}, x_{i_2}, \ldots, x_{i_j}\} \subseteq \mathcal{X}$

p: arity of C, $p = |X_C|$

d: max $|D_{x_i}|$

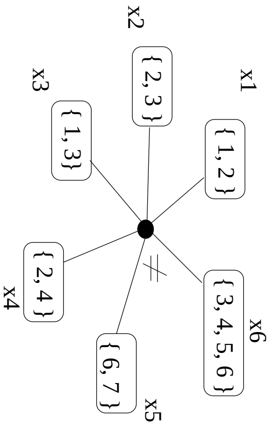
- A value a_i for x_i is consistent for C, if $\exists$ values for other all variables in X_C such that these values and a_i simultaneously satisfy C
- A constraint C is consistent, if all values for all variables X_C are consistent for C
- A CSP is arc-consistent, if all constraints (whatever their arity) are consistent
- A CSP is diff-arc-consistent iff all its all-diffs constraints are arc-consistent

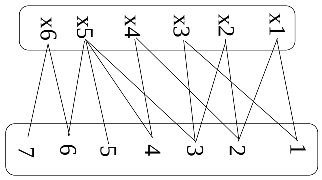
Value Graph

Given C, an all-diff constraint,
the value Graph of C is a bipartite graph

$$\mathrm{GV}(C) = (X_C, D(X_C), E)$$

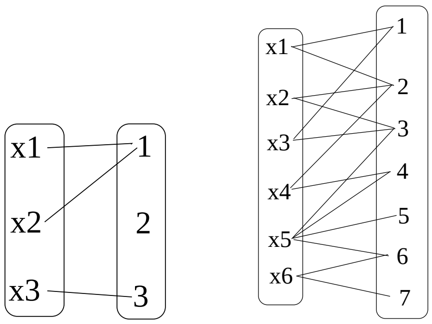
Vertices:	X_C	$=$	$\{x_{i_1}, x_{i_2}, \ldots, x_{i_j}\}$
Vertices:	$D(X_C)$	$=$	$\cup_{x \in X_C}(D_x)$
Edges:	(x_i, a)	iff	$a \in D_x$





Space complexity?
Draw GV of the 3-node coloring example

Definitions: matching



Matching: a subset of edges in G with no vertex in common

Max. matching biggest possible

Matching covers a set X: every vertex in X is an endpoint for an edge in matching

- Left: M that covers X_C is a max matching
- If every edge in GV(C) is in a matching that covers X_C, is consistent for C

Theorem 1

CSP: $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ is diff-arc-consistent iff

for every all-diff $C \in \mathcal{C}$

every edge GV(C) belongs to a matching that covers X_C in GV(C)

Task:

Repeat for each all-diff constraint,
- Build G ($\equiv$ GV) of all-diff constraint C
- Remove edges that do not belong to <u>any</u> matching covering X_C

Algorithm 1:

- Compute one M(G), maximal matching in G
- If M(G) does not cover X_C, then stop
- Using M(G), remove edges that do not belong...

→ Hopcroft & Karp: Efficient procedure for computing a matching covering X_C

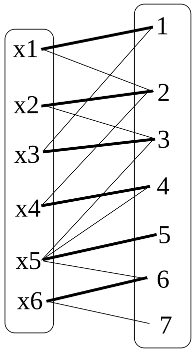
Our problem becomes

Given:

- an all-diff constraint C
- its value graph $G = (X, Y, E)$
- one maximum covering $M(G)$

Remove edges that belong to no matching covering X

Definitions



Given a matching M:

matching edge: an edge in M

free edge: an edge not in M

matched vertex: incident to a matching edge

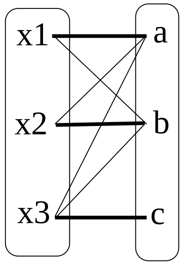
free vertex: otherwise

alternative path (cycle): a path (cycle) whose edges are alternatively matching and free

length of a path: number of edges in path

vital edge: belongs to every maximum matching

Questions

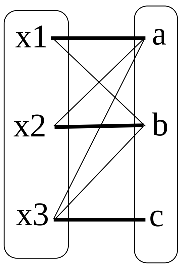


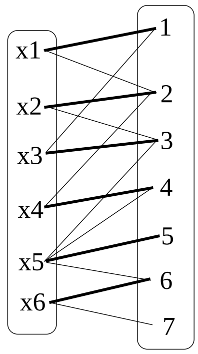
Indicate:

- matching edges
- free edges
- matched vertices
- a free vertex
- an alternative path, length?
- an alternative cycle, length?
- a vital edge

Property 1 (Berge)

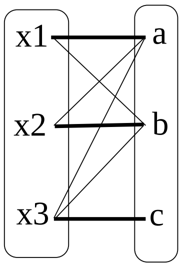
An edge belongs to some of but not all maximum matchings, iff for an arbitrary maximum matching M, it belongs to either:
- an even alternating cycle, or
- an even alternating path that begins at a free vertex

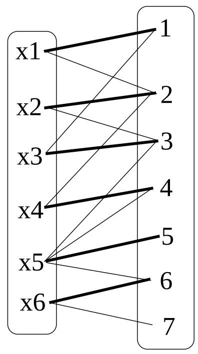




Thus:

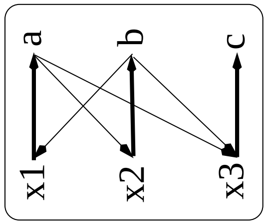
The edges to remove should not be in:
- all matchings (vital)
- an even alternating path starting at a free vertex
- an even alternating cycle

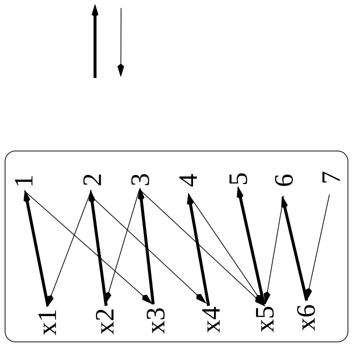




Given:

- $G = (X, Y, E)$
- a matching $M(G)$ covering X
- Build G_O, by orienting the edges





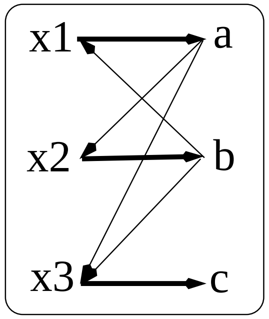
- every directed cycle in G_O corresponds to an even alternating cycle of G, and conversely
- every directed simple path in G_O, starting at a free vertex corresponds to an even alternating path of G starting at a free vertex, an conversely

Task:

Given G, and $M(G)$, remove edges that do not belong to any matching covering X_C

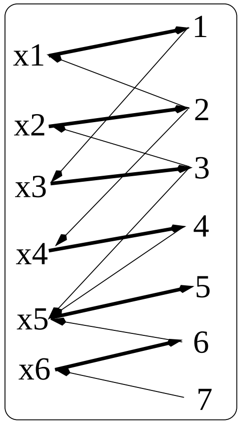
Algorithm 2

- Build G_O
- Mark all edges of G_O as unused
- Identify all edges starting at a free vertex by a breadth-first search, mark them as used
- Compute strongly connected components in G_O. Mark "used" any directed edge between two vertices in the same strongly connected component, as any such edge belongs to a directed cycle and conversely
- All remaining unused edges,
  if they are in $M(G)$, mark them as vital
  else put them in RE and remove them from G



Algorithm 2

- ...
- Identify all edges starting at a free vertex by a breadth-first search, mark them as used
- Compute strongly connected components in G_O. Mark "used" any directed edge between two vertices in the same strongly connected component, as any such edge belongs to a directed cycle and conversely
- All remaining unused edges,
  if they are in $M(G)$, mark them as vital
  else put them in RE and remove them from G



Algorithm 2

- ...
- Identify all edges starting at a free vertex by a breadth-first search, mark them as used
- Compute strongly connected components in G_O. Mark "used" any directed edge between two vertices in the same strongly connected component, as any such edge belongs to a directed cycle and conversely
- All remaining unused edges,
  if they are in $M(G)$, mark them as vital
  else put them in RE and remove them from G

So far..

Given C, remove edges that are not consistent for C

.. but,

A variable x may be in more than one all-diff constraints, *i.e.* x may be in X_{C_i} and X_{C_j}, with C_i and C_j two all-diff constraints

How to propagate the effect of filtering of C_i on C_j?

$\rightarrow$ start from scratch?

$\rightarrow$ propagate deletions more intelligently

use the fact that before deletion due to C_j,

a matching covering X_{C_i} was known in GV(C_i)

Assume we have C_i, C_j, and C_k involving a given variable

$$\text{Compute} \begin{cases} \text{RE}(C_i), \text{RE}(C_j), \text{RE}(C_k), \\ \text{G=GV}(C_i), \text{M(G), etc.} \end{cases}$$

Idea

Consider C_i

First remove from G deletions due to C_j, C_k

Second, try to extend the remaining edges in M(G) into a matching that covers X_{C_i}

Finally, apply **Algorithm 2**

... iterate

Consider C_i, G = GV(C_i), M(G)

Set RE $\leftarrow$ RE(C_i)

ER $\leftarrow$ RE(C_j) $\cup$ RE(C_k)

Include Algorithm 3 from paper

Example: the Zebra problem

5 houses of different colors
5 inhabitants, different nationalities, different pets, different drinks, different cigarettes

Include constraints from paper

Query: who drinks water?
who owns a zebra?

Zebra: formulation

25 variables:
$$\begin{cases} \text{5 house-color } C_1, C_2, \ldots, C_5 \\ \text{5 nationalities } N_1, N_2, \ldots, N_5 \\ \text{5 drinks } B_1, B_2, \ldots, B_5 \\ \text{5 cigarettes } T_1, T_2, \ldots, T_5 \\ \text{5 pets } A_1, A_2, \ldots, A_5 \end{cases}$$

Include variables from paper

Domain of each variable $= \{1, 2, 3, 4, 5\}$
$(\equiv \{h1, h2, h3, h4, h5\})$
Constraints 2–15?

Formulating Constraint 1:

1. Binary constraint between any pair in each cluster: binary CSP
2. Five 5-ary all-diff constraints: non-binary CSP
3. The 5-ary constraints are replaced with their GV. Space?

Results (I)

Formulation 1 solved with AC

Include results from paper

Formulation 2 solved with GAC-4

Include results from paper

Formulation 3 solved with the new technique.

Same results as 2.

Results (II)

a: # of binary constraints
p: size of a cluster
c: # of clusters
d: # of values in a domain
$\mathcal{O}(ad^2)$: complexity of AC on binary

Formulation 1 solved with AC

- number of binary constraint added is $\mathcal{O}(cp^2)$
- filtering complexity is $\mathcal{O}((a + cp^2)d^2)$

Formulation 2 solved with GAC-4

- filtering complexity is $\mathcal{O}(\frac{d!}{(d-p)!}p)$

Formulation 3 solved with the new technique

- arc-consistency is $\mathcal{O}(ad^2)$
- all-diff filtering is $\mathcal{O}(cp^2d^2)$
- total filtering is $\mathcal{O}(ad^2 + cp^2d^2)$

Extension

Improved bounds by J.-F. Puget (AAAI 99) for ordered domains (*e.g.*, time in scheduling).

Lesson

We can improve the performance of search by:

- identifying special structures in the constraint graph (*e.g.*, tree, biconnected components, DAG)
- identifying special types of constraints (*e.g.*, functional, anti-functional, monotonic, all-diffs)

Improved arc-consistency

Van Hentenryck et al. AIJ 92

Functional

A constraint C is <u>functional with respect to a domain D</u> iff for all $v \in D$ (respectively $w \in D$) there exists at most one $w \in D$ (respectively $v \in D$) such that $C(v, w)$.

Anti-functional

A constraint C is <u>anti-functional with respect to a domain D</u> iff $\neg C$ is functional with respect to D.

Monotonic

A constraint C is <u>monotonic with respect to a domain D</u> iff there exists a total ordering on D such that, for all values v and $w \in D$, $C(v, w)$ holds implies $C(v', w')$ holds for all values v' and $w' \in D$ such that $v' \leq v$ and $w' \leq w$.