Foundations of Constraint Processing
CSCE421/821, Fall 2005
www.cse.unl.edu/~choueiry/F05-421-821/

Berthe Y. Choueiry (Shu-we-ri)
Avery Hall, Room 123B
choueiry@cse.unl.edu, Tel: (402)472-5444

Images scanned from paper by Nimit Mehta
**Context:** finite CSPs

**Goal:** efficiency of arc consistency

**Focus:** All-diff constraints

**Result:** efficient algorithm

\[
\begin{align*}
\text{Space: } & \mathcal{O}(pd) \\
\text{Time: } & \mathcal{O}(p^2d^2)
\end{align*}
\]

\(p: \#\text{vars}, \ d: \text{max domain size}\)

**Application:** used in RESYN for subgraph isomorphism

(plan synthesis in organic chemistry)

---

**Contributions**

- An algorithm to establish arc consistency in an all-diff constraint
  - → efficient
  - → 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
    \( \frac{d!}{(d-p)!} \)
    \( p: \#\text{vars}, d: \max \text{domain size} \)

- 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
  \( x_1 \)
  \hline
  \( \{a, b\} \)
  \( \neq \)
  \( \{a, b, c\} \)
  \( x_2 \)
  \( x_3 \)
  GAC4: rules out \( a, b \) for \( x_3 \)

- Set of binary constraints
  \( x_1 \)
  \hline
  \( \{a, b\} \)
  \( \neq \)
  \( \{a, b, c\} \)
  \( x_2 \)
  \( x_3 \)
  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 if 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:

$ GV(C) = (X_C, D(C), E)$

where $X_C$ is the set of variables, $D(C)$ is the domain of each variable, and $E$ is the set of edges between variables and their domains.
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 $X_C$, $C$ is consistent

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 any 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...

\[
\text{Algorithm 1: DIFF-INITIALIZATION}(G)
\]
% returns false if there is no solution, otherwise true
% the function COMPUTEMAXIMUMMATCHING\((G)\) computes a maximum matching in the graph \( G \)
begin
1. Build \( G = (X_C, D(X_C), E) \)
2. \( M(G) \leftarrow \text{COMPUTEMAXIMUMMATCHING}(G) \)
   if \( |M(G)| < |X_C| \) then return false
3. \( \text{REMOVEDGESFROM}(G, M(G)) \)
   return true
end

\[ \rightarrow \text{Hopcroft & Karp: Efficient procedure} \]
for computing a matching covering \( X_C \)
\[ \rightarrow \text{Or, maximal flow in bipartite graph (less efficient)} \]

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

- alternating 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 alternating path, length?
- an alternating 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
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 directed edges that belong to a directed simple path 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

Algorithm 2: REMOVEEDGESFROMG(G,M(G))
% RE is the set of edges removed from G.
% M(G) is a matching of G which covers X
% The function returns RE
begin
1. Mark all directed edges in GO as “unused”.
   Set RE to θ.
2. Look for all directed edges that belong to
   a directed simple path which begins at a free
   vertex by a breadth-first search starting from
   free vertices, and mark them as “used”.
3. Compute the strongly connected components of GO.
   Mark as “used” any directed edge that joins two
   vertices in the same strongly connected component.
4. for each directed edge de marked as “unused” do
   set e to the corresponding edge of de
   if e ∈ M(G) then mark e as “vital”
   else
     RE ← RE U {e}
     remove e from G
return RE
end
Algorithm 2

- All remaining unused edges, if they are in $M(G)$, mark them as vital
- Identify all edges starting at a free vertex by a breadth-first search, mark them as used

Mark "used" any directed edge between two directed cycle and conversely directed component, as any such edge belongs to a component.

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 $C_j$?

→ start from scratch?

→ 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

Compute

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

**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

---

**Algorithm 2: Diff-Propagation \( G, M(G), \text{ER} \)**

Set \( \text{RE} \leftarrow \text{RE}(C_i) \cup \text{RE}(G) \)

Consider \( C_i, G = \text{GV}(C_i), M(G) \)

if \( \text{RE} \leftarrow \text{RE}(C_i) \cup \text{RE}(G) \)

end
**Example:** the Zebra problem

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

Consider the following facts:

1. The Englishman lives in the red house
2. The Spaniard has a dog
3. Coffee is drunk in the green house
4. The Ukrainian drinks tea
5. The green house is immediately to the right of the ivory house

6. The snail owner smokes Old-Gold

7. *etc.*

**Query:** who drinks water?
who owns a zebra?

**Zebra:** formulation

25 variables:

- 5 house-color $C_1, C_2, \ldots, C_5$
- 5 nationalities $N_1, N_2, \ldots, N_5$
- 5 drinks $B_1, B_2, \ldots, B_5$
- 5 cigarettes $T_1, T_2, \ldots, T_5$
- 5 pets $A_1, A_2, \ldots, A_5$

Domain of each variable = \{1, 2, 3, 4, 5\}
(\equiv \{h_1, h_2, h_3, h_4, h_5\})
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 (1)

Formulation 1 solved with AC

Formulation 2 solved with GAC-4

Formulation 3 solved with the new technique.
Same results as 2.
Results (II)

- $a$: number of binary constraints
- $p$: size of a cluster
- $c$: number of clusters
- $d$: number of values in a domain

$O(ad^2)$: complexity of AC on binary

**Formulation 1** solved with AC
- number of binary constraint added is $O(cp^2)$
- filtering complexity is $O((a + cp^2)d^2)$

**Formulation 2** solved with GAC-4
- filtering complexity is $O\left(\frac{d!}{(d-p)!p}\right)$

**Formulation 3** solved with the new technique
- arc-consistency is $O(ad^2)$
- all-diff filtering is $O(cp^2d^2)$
- total filtering is $O(ad^2 + cp^2d^2)$

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-diff)
**Improved arc-consistency**  Van Hentenryck et al. AIJ 92

**Functional**
A constraint $C$ is functional with respect to a domain $D$ 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 anti-functional with respect to a domain $D$ iff

$\neg C$ is functional with respect to $D$.

**Monotonic**
A constraint $C$ is monotonic with respect to a domain $D$ 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 $w' \leq w$. 