Constraint Satisfaction Problems
Constraint Networks

Bernhard Nebel and Stefan Wölfle
based on a slideset by
Malte Helmert and Stefan Wölfle
(summer term 2007)

Albert-Ludwigs-Universität Freiburg

October 26, 2009
Constraint Satisfaction Problems

October 26, 2009 — Constraint Networks

Constraint Networks
Solution
Normalized Constraint Networks
Deduction
Minimal Networks

Projection Networks

Constraint Networks and Graphs
Primal Constraint Graphs
Dual Constraint Graph
Constraint Hypergraph

Solving Constraint Networks
Constraint Networks

Definition
A constraint network is a triple

\[ C = \langle V, \text{dom}, C \rangle \]

where:

- \( V \) is a non-empty and finite set of variables.
- \( \text{dom} \) is a function that assigns a non-empty (value) set (domain) to each variable \( v \in V \).
- \( C \) is a set of relations over variables of \( V \) (constraints), i.e., each constraint is a relation \( R_{v_1, \ldots, v_n} \) over some variables \( v_1, \ldots, v_n \) in \( V \).

The set of scopes \( \{S_1, \ldots, S_t\} \) is called network scheme.
Example: 4-Queens Problem

Consider variables $v_1, \ldots, v_4$ (associated to the columns of a $4 \times 4$-chess board). Each of these variables $v_i$ has as its domain $\{1, \ldots, 4\}$ (conceived of as the row positions of a queen in column $i$).

![Chessboard diagram]

Define then binary constraints (thus encoding possible queen movements):

- $R_{v_1, v_2} := \{(1, 3), (1, 4), (2, 4), (3, 1), (4, 1), (4, 2)\}$
- $R_{v_1, v_3} := \{(1, 2), (1, 4), (2, 1), (2, 3), (3, 2), (3, 4), (4, 1), (4, 3)\}$

...
Graph Representation of Binary Constraint Networks

Constraint networks with binary constraints only can be represented by a directed labelled graph (even: an undirected graph if all constraints are symmetric).

Example: The constraint network defined by:

\[ V = \{v_1, v_2, v_3\}, \]
\[ \text{dom}(v_i) = \{\text{red}, \text{blue}\} \ (i = 1..3), \]
\[ C = \{((v_1, v_2), \neq), ((v_2, v_1), \neq), ((v_1, v_3), \neq)\} \]
**Solvability of Networks**

**Definition**
A constraint network is **solvable** (or: **satisfiable**) if there exists an assignment

\[ a: V \rightarrow \bigcup_{v \in V} \text{dom}(v) \]

such that

(a) \( a(v) \in \text{dom}(v) \), for each \( v \in V \),

(b) \( (a(v_1), \ldots, a(v_n)) \in R_{v_1,\ldots,v_n} \) for all constraints \( R_{v_1,\ldots,v_n} \).

A **solution** of a constraint network is an assignment that solves the network.
Instantiation, Partial Solution

Let $\mathcal{C} = \langle V, \text{dom}, \mathcal{C} \rangle$ be a constraint network.

Definition

(a) An instantiation of a subset $V'$ of $V$ is an assignment $a : V' \rightarrow \bigcup_{v \in V'} \text{dom}(v_i)$ with $a(v_i) \in \text{dom}(v_i)$.

(b) An instantiation $a$ is a partial solution if $a$ satisfies each constraint with scope $S \subseteq V'$. We also say: $a$ is consistent relative to $\mathcal{C}$.

(c) For an instantiation $a$ of a subset $V' = \{v_1, \ldots, v_n\}$ and a constraint $R$ with scope $S \subseteq V'$, let

$$\bar{a}[S] := (a(v_1), \ldots, a(v_n)).$$

Hence a solution is an instantiation of all variables in $V$ that is consistent relative to $\mathcal{C}$.
Instantiation, Solution

Note:

(a) An instantiation of variables in $V' \subseteq V$, $a$, is a partial solution (consistent relative to $C$) iff

$$\bar{a}[S] \in R, \quad \text{for each constraint } R \text{ with scope } S \subseteq V'.$$

(b) Not every partial solution is part of a (full) solution, i.e., there may be partial solutions of a constraint network that cannot be extended to a solution. For the 4-queens problem, for example,
Normalized Constraint Networks

Let $C = \langle V, \text{dom}, C \rangle$ be a constraint network. According to our definition it is possible that $C$ contains constraints

\[ R_{v_{i_1}, \ldots, v_{i_k}} \quad \text{and} \quad S_{v_{j_1}, \ldots, v_{j_k}} \]

where $(j_1, \ldots, j_k)$ is just a permutation of $(i_1, \ldots, i_k)$.

Without changing the set of solutions, we can simplify the network by deleting $S_{v_{j_1}, \ldots, v_{j_k}}$ from $C$ and rewriting $R_{v_{i_1}, \ldots, v_{i_k}}$ as follows:

\[ R_{v_{i_1}, \ldots, v_{i_k}} \leftarrow R_{v_{i_1}, \ldots, v_{i_k}} \cap \pi_{v_{i_1}, \ldots, v_{i_k}}(S_{v_{j_1}, \ldots, v_{j_k}}). \]

Given a fixed order on the set of variables $V$, we can systematically delete-and-refine constraints. The result is a constraint network that contains at most one constraint for each subset of variables. This network is referred to as a normalized constraint network.
**Equivalence**

Let $C$ and $C'$ be constraint networks on the same set of variables and on the same domains for each variable.

**Definition**

$C$ and $C'$ are **equivalent** if each solution of $C$ is a solution of $C'$, and vice versa.

**Example:**

```
  v_1
 / \  \
 red blue red blue
  \  /
   \=
  v_2 v_3

  v_1
 / \  \
 red blue red blue
  \   /
   \=  
  v_2 v_3
```
Tightness

Let $C$ and $C'$ be (normalized) constraint networks on the same set of variables and on the same domains for each variable.

**Definition**

$C$ is as tight as $C'$ if for each constraint $R$ of $C$ with scope $S$,

(a) $C'$ has no constraint with scope $S$, or

(b) $R \subseteq R'$, where $R'$ is the constraint of $C'$ with scope $S$.

**Notes:**

- Constraint tightness has a large influence on the solubility of constraint networks.
- Be warned: different concepts of tightness can be found in the literature
- Here: Tightness does not account for comparing constraints with different arities
Intersection of Networks

Let $C$ and $C'$ be constraint networks as above.

**Definition**

The intersection of $C$ and $C'$, $C \cap C'$, is the network defined by intersecting for each scope $S$ of constraints $R_S \in C$ and $R'_S \in C'$ the respective relations, i.e.,

$$R''_S := R_S \cap R'_S.$$

If for a scope $S$ only one of the networks contains a constraint, then we set:

$$R''_S := R_S \quad \text{(or := } R'_S, \text{ resp.)}$$

**Lemma**

If $C$ and $C'$ are equivalent networks, then $C \cap C'$ is equivalent to both networks and as tight as both networks.
Minimal Network

Definition
Let $C_0$ be a constraint network and let $C_1, \ldots, C_k$ be the set of all constraint networks (defined on the same set of variables and the same domains) that are equivalent to $C_0$.

\[ \bigcap_{1 \leq i \leq k} C_i \]

is the minimal network of $C_0$.

Lemma
The minimal network is equivalent to and as tight as all the constraint networks $C_i$. There is no network equivalent to $C_0$ that is tighter than the minimal network.
Projecting Relations

Let $R_S$ be a relation with scope $S = \{v_1, \ldots, v_k\}$ (we can think of $R_S$ as a constraint network . . .).

Definition
The projection network of $R_S$, $\text{Proj}(R_S)$, is the constraint network defined by:

$$
V := S \\
\text{dom}(v_i) := \pi_{v_i}(R_S) \\
R_{v_i,v_j} := \pi_{v_i,v_j}(R_S)
$$

Note: The projection network is an upper approximation by binary networks in the following sense:

Lemma
Any solution of $R_S$ (as a network) defines a solution of $\text{Proj}(R_S)$.
Binary Representation

Definition
A relation $R_S$ with scope $S$ has a binary representation if the relation (conceived of as a network) is equivalent to $\text{Proj}(R_S)$.

From the fact that a relation has a binary representation, it does not follow that all its projections have binary representations as well (Exercise!).

Definition
A relation $R_S$ with scope $S$ is binary decomposable if the relation itself and all its projections to subsets of $S$ (with at least 3 elements) have a binary representation.
Let $C = \langle V, \text{dom}, C \rangle$ be a (normalized) constraint network.

**Definition**

The **primal constraint graph** of a network $C = \langle V, \text{dom}, C \rangle$ is the undirected graph

$$G_C := \langle V, E_C \rangle$$

where

$$\{u, v\} \in E_C \iff \{u, v\} \text{ is a subset of the scope of some constraint in } C.$$
Primal Constraint Graph: Example

Consider a constraint network with variables $v_1, \ldots, v_5$ and two ternary constraints $R_{v_1, v_2, v_3}$ and $S_{v_3, v_4, v_5}$.

Then the primal constraint graph of the network has the form:

Absence of an edge between two variables/nodes means that there is no direct constraint between these variables.
Dual Constraint Graphs

Definition
The dual constraint graph of a constraint network \( C = \langle V, \text{dom}, C \rangle \) is the labeled graph

\[
D_C := \langle V', E_C, l \rangle
\]

with

\[
X \in V' \iff X \text{ is the scope of some constraint in } C
\]

\[
\{X, Y\} \in E_C \iff X \cap Y \neq \emptyset
\]

\[
l : E_C \rightarrow 2^V, \quad \{X, Y\} \mapsto X \cap Y
\]

In the example above, the dual constraint graph is:
Definition
The constraint hypergraph of a constraint network $C = \langle V, \text{dom}, C \rangle$ is the hypergraph

$$H_C := \langle V, E_C \rangle$$

with

$$X \in E_C \iff X \text{ is the scope of some constraint in } C.$$  

In the example above (constraint network with variables $v_1, \ldots, v_5$ and two ternary constraints $R_{v_1, v_2, v_3}$ and $S_{v_3, v_4, v_5}$) the hyperedges of the constraint hypergraph are:

$$E_C = \big\{ \{v_1, v_2, v_3\}, \{v_3, v_4, v_5\} \big\}.$$
Simple Solution Strategy: Guess and Check

**Backtracking**: search systematically for consistent partial instantiations in a depth-first manner:

- **forward phase**: extend the current partial solution by assigning a consistent value to some new variable (if possible)
- **backward phase**: if no consistent instantiation for the current variable exists, we return to the previous variable.
Backtracking Algorithm

**Backtracking**\((C, a)\):

*Input:* a constraint network \(C = \langle V, D, C \rangle\) and a partial assignment \(a\) of \(C\)
(e.g., the empty instantiation \(a = \{\}\))

*Output:* a solution of \(C\) or “inconsistent”

if \(a\) is not consistent with \(C\):
    return “inconsistent”

if \(a\) is defined for all variables in \(V\):
    return \(a\)

select some variable \(v_i\) for which \(a\) is not defined

for each value \(x\) from \(D_i\):
    \(a' := a \cup \{v_i \mapsto x\}\)
    \(a'' \leftarrow \text{Backtracking}(C, a')\)
    if \(a''\) is not “inconsistent”:
        return \(a''\)

return “inconsistent”
Literature

Rina Dechter.
Constraint Processing,
Chapter 2, Morgan Kaufmann, 2003