Constraint Satisfaction Problems

Constraint Networks

Stefan Wölfl, Christian Becker-Asano, and Bernhard Nebel

October 27, 2014
1 Constraint Networks

- Constraint networks
- Solution
- Normalized Constraint Networks
- Deduction
Constraint networks

Definition

A constraint network is a triple

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

where:

- \( V \) is a non-empty and finite set of variables;
- \( \text{dom} \) is a function that assigns to each variable \( v \in V \) a non-empty set \( \text{dom}(v) \) (\( \text{dom}(v) \) is called the domain of \( v \), elements of \( \text{dom}(v) \) are called values);
- \( C \) is a set of relations over variables of \( V \) (called constraints), i.e., each constraint is a relation \( R_{x_1,\ldots,x_m} \) over some scheme \( S = (x_1, \ldots, x_m) \) of variables in \( V \).

The set of constraint schemes \( \{ S_1, \ldots, S_t \} \) is called network scheme.
Constraint networks

If we assume some ordering of the variables in $V$, we can write networks more compactly:

**Definition**

A constraint network is a triple $N = \langle V, D, C \rangle$ where:

- $V = \{v_1, \ldots, v_n\}$ is a non-empty and finite ordered set of variables (assume order $(v_1, \ldots, v_n)$);
- $D = (D_1, \ldots, D_n)$ is a sequence of domains for $V$ (with $D_i$ the domain of variable $v_i$ and $D_N = D_1 \times \cdots \times D_n$ is the domain of $N$);
- $C$ is a set of constraints $(x, R)$ where $x = (v_{i_1}, \ldots, v_{i_m})$ is a scheme of variables in $V$ and $R \subseteq D_{i_1} \times \cdots \times D_{i_m}$.

**Notice:** any ordering of the variables suffices. Constraint network that differ only in the variable ordering are considered equal.
Constraint networks

- Note that we consider finitely many variables only, but (e.g., for theoretical studies) this could be relaxed.

- In the 2nd definition we assumed that the relations of the constraint are embedded in the domain of the network: \( R \subseteq D_{i_1} \times \cdots \times D_{i_m} \). Such networks are called embedded networks (see Bessiere, 2006).

- The definition does not require that constraint relations are given explicitly (in extension, i.e. by a set of its tuples; table constraint). A constraint relation \( R \) could be specified by any Boolean function \( f_R \): a tuple satisfies the constraint \( R \) iff \( f_R \) applied on the tuple gives 1.

- If not stated otherwise, we will always assume that the domains of the variables are given in extension.
Example: 4-queens problem

The 4-queens problem can be represented as a constraint network. For example, consider variables $v_1, \ldots, v_4$ (each associated to a column of the $4 \times 4$-chess board). Each variable $v_i$ has as its domain $D_i = \{1, \ldots, 4\}$ (conceived of as the row positions of a queen in column $i$).

<table>
<thead>
<tr>
<th></th>
<th>$v_1$</th>
<th>$v_2$</th>
<th>$v_3$</th>
<th>$v_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Define then binary constraints (thus encoding “non-attacking queen positions”):

\[
R_{v_1,v_2} := \{(1, 3), (1, 4), (2, 4), (3, 1), (4, 1), (4, 2)\}
\]

\[
R_{v_1,v_3} := \{(1, 2), (1, 4), (2, 1), (2, 3), (3, 2), (3, 4), (4, 1), (4, 3)\}
\]

\[
\ldots
\]
**Example: Graph colorability**

$k$-Colorability of a graph $G$ can be represented as a constraint network of the following form:

- $V = \{v_i : v_i \text{ is a vertex in } G\}$
- $D_i = \{1, \ldots, k\} \ (v_i \in V)$
- $C = \{((v_i, v_j), \neq) : \{v_i, v_j\} \text{ is an edge of } G\}$

Binary constraint networks can be represented by a directed labeled graph (or even: by an undirected graph if all constraints are symmetric).
Solution of a constraint network

Definition

A solution of a constraint network $N = \langle V, D, C \rangle$ is a (variable) assignment

$$a : V \rightarrow \bigcup_{i : v_i \in V} D_i$$

such that:

(a) $a(v_i) \in D_i$, for each $v_i \in V$,
(b) $(a(x_1), \ldots, a(x_m)) \in R$ for each constraint $R_{x_1, \ldots, x_m}$ in $C$.

$N$ is called satisfiable if $N$ has a solution.

$sol(N)$ denotes the set of all solutions of $N$. $sol(N)$ can also be written as:

$$sol(N) = \{(d_1, \ldots, d_n) \in D_1 \times \cdots \times D_n : \text{the assignment } v_1 \mapsto d_1, \ldots, v_n \mapsto d_n \text{ defines a solution}\}$$
Instantiation, partial solution

Let $N = \langle V, D, C \rangle$ be a constraint network.

**Definition**

(a) An **instantiation** of a subset $V' \subseteq V$ is an assignment $a : V' \rightarrow \bigcup_{i: v_i \in V'} D_i$ with $a(v_i) \in D_i$.

(b) An instantiation $a$ of $V'$ is called **partial solution** if $a$ satisfies each constraint $R_S$ in $C$ with $S \subseteq V'$. In this case $a$ is called **locally consistent**.

(c) Shortcut notation: for an instantiation $a$ of $V' = \{x_1, \ldots, x_m\}$ and constraint $R_S$ with scope $S \subseteq V'$, set

$$a[S] := (a(x_1), \ldots, a(x_m)).$$

Hence, a solution is an instantiation of all variables in $V$ that is locally consistent.
Note:

(a) An instantiation of $V' \subseteq V$, $a$, is a partial solution (locally consistent) iff

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

(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:

```
  4  q
  3  q
  2
  1  q

V_1 V_2 V_3 V_4
```
Let $N = \langle V, D, C \rangle$ be a constraint network.

**Definition**

An instantiation $a'$ of subset $V' \subseteq V$ is called a nogood (of $N$) if $a'$ cannot be extended to a (full) solution of $N$, i.e., there exists no solution $a: V \rightarrow \bigcup_i D_i$ such that $a|_{V'} = a'$.

Instantiations that are no nogoods are sometimes called **consistent** or **globally consistent** (to emphasize the difference to locally consistent assignments).

Later, we will also introduce the notion of **globally consistent networks**.
Normalized constraint network

Let $N = \langle V, D, C \rangle$ be a constraint network. Due 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. This results in a constraint network that contains at most one constraint for each subset of variables. Such a network is called a normalized constraint network.
Let $N$ and $N'$ be constraint networks on the same set of variables and on the same domains for each variable.

**Definition**

$N$ and $N'$ are called equivalent if they have the same set of solutions.

**Example:**
Tightness

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

**Definition**

$N$ is as tight as $N'$ if for each constraint $R_S$ of $N$,

(a) $N'$ has no constraint with the same scope as $R_S$, or

(b) $R \subseteq \pi_S(R'_S)$, where $R'_S$ is the constraint of $N'$ with the same scope as $R_S$. 

[Diagram showing the relationship between $N$ and $N'$ with variables $v_1$, $v_2$, and $v_3$.]
Intersection of networks

Definition

The intersection of $N$ and $N'$, $N \cap N'$, is the network defined by intersecting for each scope the constraints $R_S \in C$ and $R'_S \in C'$ with the same scope, i.e., modulo a suitable permutation of the constraint schemes,

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

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 $N$ and $N'$ are equivalent networks, then $N \cap N'$ is equivalent to both networks and as tight as both networks.
2 Constraint Networks and Graphs

- Primal Constraint Graphs
- Dual Constraint Graph
- Constraint Hypergraph
Primal constraint graphs

Let $N = \langle V, D, C \rangle$ be a (normalized) constraint network.

**Definition**

The primal constraint graph of a network $N = \langle V, D, C \rangle$ is the undirected graph

$$G_N := \langle V, E_N \rangle$$

where

$$\{u, v\} \in E_N \iff \{u, v\} \text{ is a subset of the scope of some constraint in } N.$$
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 explicit constraint in which both variables participate.
Definition

The **dual constraint graph** of a constraint network \( N = \langle V, D, C \rangle \) is the labeled graph

\[
D_N := \langle V', E_N, l \rangle
\]

with

- \( X \in V' \iff X \) is the scope of some constraint in \( N \)
- \( \{X, Y\} \in E_N \iff X \cap Y \neq \emptyset \)
- \( l : E_N \to 2^V, \quad \{X, Y\} \mapsto X \cap Y \)

In the example above, the dual constraint graph is:

\[ V_1, V_2, V_3 \quad \xrightarrow{V_3} \quad V_3, V_4, V_5 \]
Definition

The **constraint hypergraph** of a constraint network \( N = \langle V, D, C \rangle \) is the hypergraph

\[ H_N := \langle V, E_N \rangle \]

with

\[ X \in E_N \iff X \text{ is the scope of some constraint in } N. \]

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:

\[ E_N = \{ \{v_1, v_2, v_3\}, \{v_3, v_4, v_5\} \}. \]
3 Solving Constraint Networks
Simple solution strategy: Backtracking search

**Backtracking**: search systematically for locally 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, return to the previous variable.
Backtracking algorithm

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

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

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

if \(a\) is not locally consistent with \(N\):
    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 \(D_i\):
    \(a' := a \cup \{v_i \mapsto x\}\)
    \(a'' \leftarrow \text{Backtracking}(N, a')\)
    if \(a''\) is not “inconsistent”:
        return \(a''\)

return “inconsistent”
Rina Dechter. Constraint Processing, Chapter 2, Morgan Kaufmann, 2003