Constraint Programming

Zdeněk Hanzálek, Jan Kelbel
hanzalek@fel.cvut.cz

CTU in Prague

May 24, 2017
1. Inspiration - Sudoku

2. Constraint Satisfaction Problem (CSP)
   - Search and Propagation
   - Arc Consistency
     - AC-3 Algorithm
   - Global Constraints
What is Constraint Programming?

- Sudoku is Constraint Programming
Assign digits to blank fields such that:

digits distinct per row, column, block
Assign digits to blank fields such that:
digits distinct per row, column, block
Sudoku

Assign digits to blank fields such that:
digits distinct per row, column, block
Sudoku

Assign digits to blank fields such that:
digits distinct per row, column, block
No blank field in the block can have a value of 3,6,8
### Sudoku - Propagation in the Lower Left Block

No blank field in the block can have a value of 3, 6, 8
- propagate to all blank fields
Use the same propagation for rows and columns

<table>
<thead>
<tr>
<th></th>
<th>8</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1,2,4,5,7,9</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>1,2,4,5,7,9</td>
<td>1,2,4,5,7,9</td>
<td></td>
</tr>
</tbody>
</table>
Prune digits from the fields such that:
digits distinct per row, column, block
Prune digits from the fields such that:
digits distinct per row, column, block
Sudoku - Propagation in One Field

Prune digits from the fields such that:
digits distinct per row, column, block
Prune digits from the fields such that:
digits distinct per row, column, **block**
Sudoku - Iterated Propagation

Iterate propagation for rows, columns and blocks

- When to stop?
- What if more assignments exist?
- What if no assignment exists?
Sudoku is Constraint Programming

Sudoku:
- **Variables** - fields
  - assign values - digits
  - maintain **domain** of variable - set of possible values
- **Constraints** - numbers in row, column and box must vary
  - relations among variables disable certain combinations of values

Constraint programming is **declarative programming**:
- **Model**: variables, domains, constraints
- **Solver**: propagation, searching
Constraint Satisfaction Problem (CSP) is defined by the triplet \((X, D, C)\), where:

- \(X = \{x_1, \ldots, x_n\}\) is a finite set of variables
- \(D = \{D_1, \ldots, D_n\}\) is a finite set of domains of variables
- \(C = \{C_1, \ldots, C_t\}\) is a finite set of constraints.

Domain \(D_i = \{v_1, \ldots, v_k\}\) is a finite set of all possible values of \(x_i\).

Constraint \(C_i\) is a couple \((S_i, R_i)\) where \(S_i \subseteq X\) and \(R_i\) is a relation over the set of variables \(S_i\). For \(S_i = \{x_{i_1}, \ldots, x_{i_r}\}\) is \(R_i \subseteq D_{i_1} \times \cdots \times D_{i_r}\).

CSP is an NP-complete problem.
Solution to Constraint Satisfaction Problem (CSP) is the complete assignment of values from the domains to the variables such that all constraints are satisfied
- it is a decision problem.

Constraint Satisfaction Optimization Problem (CSOP) is defined by \((X, D, C, f(X))\) where \(f(X)\) is the objective function. The search is not finished, when the first acceptable solution was found, but it is finished when the optimal solution was found (using branch&bound method for example).

Constraint Solving is defined by \((X, D, C)\) where \(D_i\) is defined on \(\mathbb{R}\) (e.g. the solution of the set of linear equations-inequalities).

Constraint Programming CP includes Constraint Satisfaction and Constraint Solving.
Example: \( x \in \{3, 4, 5\}, \ y \in \{3, 4, 5\}, \ x \geq y, \ y > 3 \)
Example: \( x \in \{3, 4, 5\}, \ y \in \{3, 4, 5\}, \ x \geq y, \ y > 3 \)

1. Propagate \( y > 3 \): \( x \in \{3, 4, 5\}, \ y \in \{4, 5\} \)
Example: $x \in \{3, 4, 5\}$, $y \in \{3, 4, 5\}$, $x \geq y$, $y > 3$

1. Propagate $y > 3$: $x \in \{3, 4, 5\}$, $y \in \{4, 5\}$
2. Propagate $x \geq y$: $x \in \{4, 5\}$, $y \in \{4, 5\}$
Example: $x \in \{3, 4, 5\}, \ y \in \{3, 4, 5\}, \ x \geq y, \ y > 3$

1. Propagate $y > 3$: $x \in \{3, 4, 5\}, \ y \in \{4, 5\}$

2. Propagate $x \geq y$: $x \in \{4, 5\}, \ y \in \{4, 5\}$

3. Propagation alone is not enough
   - the product of the domains (including infeasible $x = 4, \ y = 5$) is a superset of the solution
   - the search helps - we create subproblems
How it Works - Search and Propagation

Example: \( x \in \{3, 4, 5\}, \ y \in \{3, 4, 5\}, \ x \geq y, \ y > 3 \)

1. propagate \( y > 3: \) \( x \in \{3, 4, 5\}, \ y \in \{4, 5\} \)
2. propagate \( x \geq y: \) \( x \in \{4, 5\}, \ y \in \{4, 5\} \)
3. propagation alone is not enough
   - the product of the domains (including infeasible \( x = 4, y = 5 \)) is a superset of the solution
   - the search helps - we create subproblems
4. in the subproblems we use the propagation again

- The search can be driven by various means (order of the variables, division of domain/domains).
- By the propagation of the constraints we filter the domains of the variables.
In both cases we deal with declarative programming

Performance differs from problem to problem

CSP allows one to formulate **complex constraints**
(ILP uses inequalities only, CSP uses an arbitrary relation - e.g. a binary relation may be given by a set of compatible tuples)
  - CSP is more flexible, formulation is easier to understand

it is difficult to represent continuous problems by CSP
  - domains of real variables can be bypassed by using hybrid approaches
    - e.g. combination with LP

CP is new technique, it is more open
Complete search (for example Depth First Search):

\[ x_1 \in \{2, 3\}, \ x_2 \in \{1, 2, 3\}, \ x_3 \in \{4, 5\} \]

\[ x_1 > x_2 \quad \text{and} \quad x_1 + x_2 = x_3 \]
Initial propagation of constraints:

\[ x_1 \in \{2, 3\}, \ x_2 \in \{1, 2, 3\}, \ x_3 \in \{4, 5\} \]

\[ x_1 > x_2 \quad \text{and} \quad x_1 + x_2 = x_3 \]
Choose $x_1 = 2$ and propagate constraints:

$x_1 \in \{2, 3\}, \ x_2 \in \{1, X, X\}, \ x_3 \in \{X, X\}$

$x_1 > x_2 \quad x_1 + x_2 = x_3$
Choose $x_1 = 3$ and propagate constraints:

\[
x_1 \in \{2, 3\}, \quad x_2 \in \{1, 2, x\}, \quad x_3 \in \{4, 5\}
\]

\[
x_1 > x_2 \quad \quad x_1 + x_2 = x_3
\]
Choose $x_2 = 1$ and propagate constraints:

\[ x_1 \in \{2, 3\}, \quad x_2 \in \{1, 2, 3\}, \quad x_3 \in \{4, 5\} \]

\[ x_1 > x_2 \quad \quad x_1 + x_2 = x_3 \]
Choose $x_2 = 2$ and propagate constraints:

$x_1 \in \{2, 3\}, \quad x_2 \in \{1, 2, 4\}, \quad x_3 \in \{A, 5\}$

$x_1 > x_2 \quad x_1 + x_2 = x_3$
We will continue to consider only **binary CSP**, where every constraint is a binary relation

- general (n-ary) CSP can be converted to binary CSP
- binary CSP can be represented by **digraph** $G$
  - nodes are variables
  - if there is a constraint involving $x_i, x_j$, then the nodes $x_i, x_j$ are interconnected by oriented arcs $(x_i, x_j)$ and $(x_j, x_i)$

**Arc consistency is an essential method for propagation.**

- Arc $(x_i, x_j)$ is **Arc Consistent (AC)** iff for each value $a \in D_i$ there exists a value $b \in D_j$ such that the assignment $x_i = a, x_j = b$ meets all binary constraints for the variables $x_i, x_j$.
- A **CSP is arc consistent** if all arc are arc consistent.
- Note that AC is **oriented** - the consistence of arc $(x_i, x_j)$ does not guarantee the consistence of arc $(x_j, x_i)$.

There are other local consistencies (path consistency, k-consistency, singleton arc consistency,...). Some of them are stronger, some are weaker.
REVISE Procedure

From domain $D_i$ delete any value $a$, which is not consistent with domain $D_j$.

```plaintext
procedure REVISE
Input: Domain $D_i$ to be revised. Domain $D_j$. Set of constraints $C$.
Output: Binary variable deleted indicating deletion of some value from $D_i$. Revised domain $D_i$.

deleted := 0;
for $a \in D_i$ do
    if there is no $b \in D_j$ ; $x_i = a$, $x_j = b$ satisfies all constraints on $x_i$, $x_j$
        then
            $D_i := D_i \setminus a$; // delete $a$ from $D_i$
            deleted := 1;
        end
    end
end
```

Z. Hanzálek (CTU)
Example: Application of REVISE

CSP with variables \( X = \{x_1, x_2, x_3\} \), constraints \( x_1 > x_2 \), \( x_2 \neq x_3 \), \( x_2 + x_3 > 4 \), and domains \( D_1 = \{1, 2, 3\} \), \( D_2 = \{1, 2, 3\} \), \( D_3 = \{2, 3\} \).

After revision, some of the arcs are still **nonconsistent**

- the reason is that some of the domains have been reduced
- continue in the revision until all the arc are consistent (without consistence check - see AC-3)
Arc Consistency - AC-3 Algorithm

Maintain a queue of arcs to be revised (the arc is put in the queue only if it’s consistency could have been affected by the reduction of the domain).

**procedure AC-3**

**Input:** $X, D, C$ and graph $G$.

**Output:** Binary variable $fail$ indicating no solution in this part of the state space. The set of the revised domains $D$.

\[
\begin{align*}
fail &= 0; Q := E(G); & \quad \text{// initialize $Q$ by arcs of $G$} \\
\text{while } Q \neq \emptyset \text{ do} \\
& \quad \text{select and remove arc } (x_k, x_m) \text{ from } Q; \\
& \quad (\text{deleted}, D_k) = \text{REVISE}(D_k, D_m, C); \\
& \quad \text{if } \text{deleted} \text{ then} \\
& \quad \quad \text{if } D_k = \emptyset \text{ then } fail = 1 \text{ and EXIT ;} \\
& \quad \quad \text{else } Q := Q \cup \{(x_i, x_k) \text{ such that } (x_i, x_k) \in E(G) \text{ and } i \neq m\}; \\
& \quad \end{align*}
\]

The revision of $(x_k, x_m)$ does not change the arc consistency of $(x_m, x_k)$.
Example: Iteration of AC-3

CSP with variables $X = \{x_1, x_2, x_3\}$, constraints $x_1 = x_2$, $x_2 + 1 = x_3$ and domains $D_1 = \{1, 2, 3\}$, $D_2 = \{1, 2, 3\}$, $D_3 = \{1, 2, 3\}$.

Initialization: $Q = \{(x_1, x_2), (x_2, x_1), (x_2, x_3), (x_3, x_2)\}$

revise $(x_1, x_2)$
$D_1 = \{1, 2, 3\}$, $D_2 = \{1, 2, 3\}$, $D_3 = \{1, 2, 3\}$
$Q = \{(x_2, x_1), (x_2, x_3), (x_3, x_2)\}$

revise $(x_2, x_1)$
$D_1 = \{1, 2, 3\}$, $D_2 = \{1, 2, 3\}$, $D_3 = \{1, 2, 3\}$
$Q = \{(x_2, x_3), (x_3, x_2)\}$

revise $(x_2, x_3)$
$D_1 = \{1, 2, 3\}$, $D_2 = \{1, 2\}$, $D_3 = \{1, 2, 3\}$
$Q = \{(x_3, x_2), (x_1, x_2)\}$

revise $(x_3, x_2)$
$D_1 = \{1, 2, 3\}$, $D_2 = \{1, 2\}$, $D_3 = \{2, 3\}$
$Q = \{(x_1, x_2)\}$

revise $(x_1, x_2)$
$D_1 = \{1, 2\}$, $D_2 = \{1, 2\}$, $D_3 = \{2, 3\}$
$Q = \emptyset$
Global Constraints

Global constraint

- capture **specific structure** of the problem
- use this structure for efficient propagation using **specialized propagation algorithm**

Example: On set $X = \{x_1, \ldots, x_n\}$ we apply constraint $x_i \neq x_j \ \forall i \neq j$

- This can be formulated by many inequalities.
- The second option is the global constraint **alldifferent**, which uses a matching algorithm in a bipartite graph, where one side represents the variables and the other side represents the values.

Other examples of global constraints:

- scheduling (edge-finder)
- graph algorithms (clique, cycle)
- finite state machine
- bin-packing
Tools for Solving CSP

Proprietary:

- SICStus Prolog
- IBM CP, CP Optimizer (C++)
- IBM OPL Studio (OPL)
- Koalog (Java)

Open source:

- ECLiPSe (Prolog)
- Gecode (C++)
- Choco Solver (Java)
- Python constraints
Roman Barták.
Programování s omezujícími podmínkami.

Rina Dechter.
Constraint Processing.

Christian Schulte.
Constraint programming.
http://www.informatik.uni-osnabrueck.de/knust/class/, 2010.