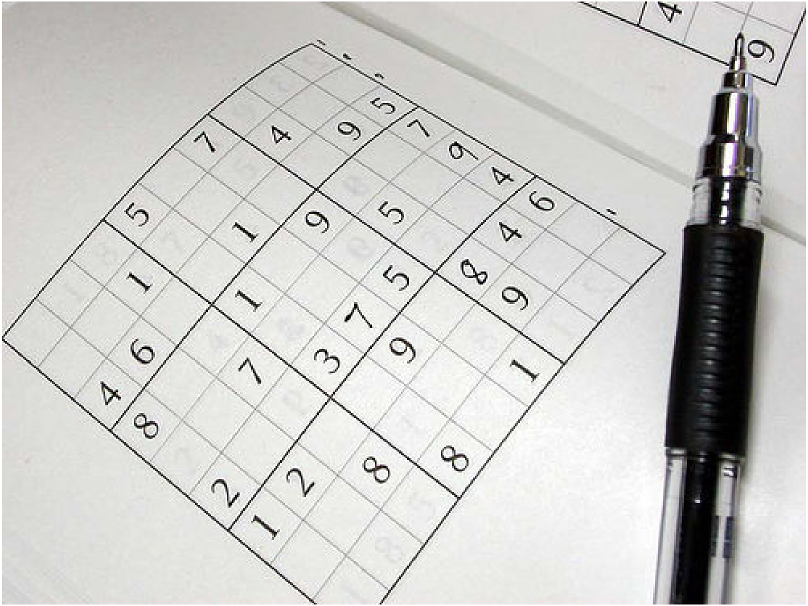
# Constraint Programming

Mikael Z. Lagerkvist

Tomologic

October 2015

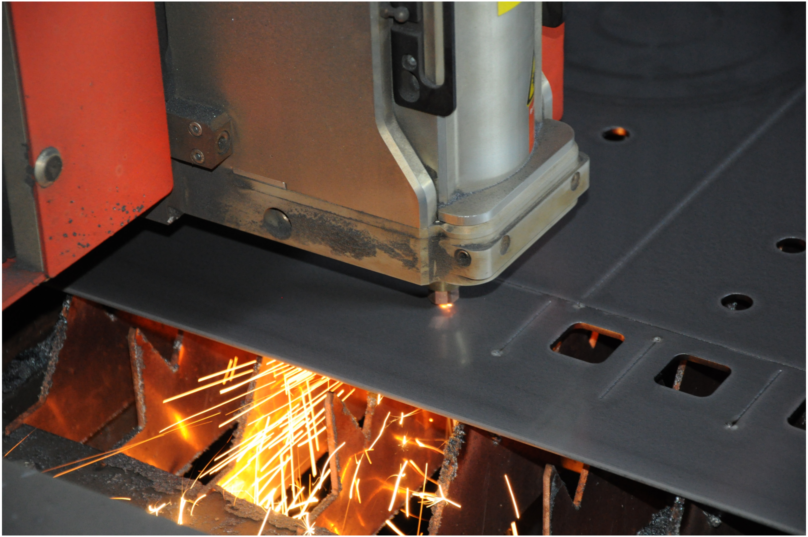


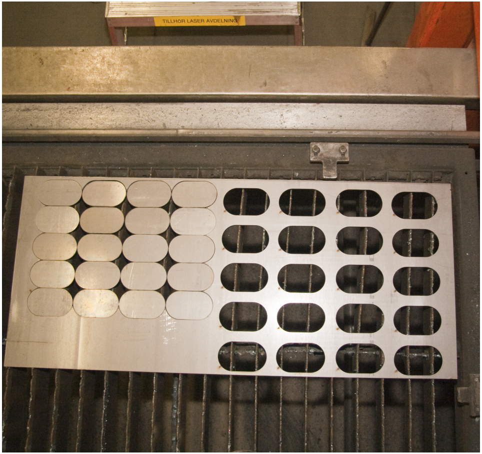
[flickr.com/photos/santos/]

# Who am I?

- Mikael Zayenz Lagerkvist
- Basic education at KTH 2000-2005
  - Datateknik
- PhD studies at KTH 2005-2010
  - Research in constraint programming systems
  - One of three core developers for Gecode, fast and well-known Constraint Programming (CP) system.

    http://www.gecode.org
- Senior developer R&D at Tomologic
  - Optimization systems for sheet metal cutting
  - Constraint programming for some tasks




TILLHOR LASER AVDELNING

# Tomologic

- Mostly custom algorithms and heuristics
- Part of system implemented using CP at one point
  - Laser Cutting Path Planning Using CP
    Principles and Practice of Constraint Programming 2013
    M. Z. Lagerkvist, M. Nordkvist, M. Rattfeldt
- Some sub-problems solved using CP
  - Ordering problems with side constraints
  - Some covering problems

1. Introduction
2. Sudoku example
3. Solving Sudoku with CP
4. Constraint programming basics
5. Constraint programming in perspective
   - Constraint programming evaluation
   - Constraint programming alternatives
6. Summary

# Sudoku - The Rules

- Each square gets one value between 1 and 9
- Each row has all values different
- Each column has all values different
- Each square has all values different

# Sudoku - Example

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | 3 | | 6 | |
| | | | | | | | 1 | |
| | 9 | 7 | 5 | | | | 8 | |
| | | | | 9 | | 2 | | |
| | | 8 | | 7 | | 4 | | |
| | | 3 | | 6 | | | | |
| | 1 | | | | 2 | 8 | 9 | |
| | 4 | | | | | | | |
| | 5 | | 1 | | | | | |

# Sudoku - Example

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | 3 | | 6 | |
| | | | | | | | 1 | |
| | 9 | 7 | 5 | | | | 8 | |
| | | | | 9 | | 2 | | |
| | | 8 | | 7 | | 4 | | |
| | | 3 | | 6 | | | | |
| | 1 | X | | | 2 | 8 | 9 | |
| | 4 | | | | | | | |
| | 5 | | 1 | | | | | |

$$X = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

# Sudoku - Example

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | 3 | | 6 | |
| | | | | | | | 1 | |
| | 9 | 7 | 5 | | | | 8 | |
| | | | | 9 | | 2 | | |
| | | 8 | | 7 | | 4 | | |
| | | 3 | | 6 | | | | |
| | 1 | X | | | 2 | 8 | 9 | |
| | 4 | | | | | | | |
| | 5 | | 1 | | | | | |

$$X = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

# Sudoku - Example

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | 3 | | 6 | |
| | | | | | | | 1 | |
| | 9 | 7 | 5 | | | | 8 | |
| | | | | 9 | | 2 | | |
| | | 8 | | 7 | | 4 | | |
| | | 3 | | 6 | | | | |
| | 1 | X | | | 2 | 8 | 9 | |
| | 4 | | | | | | | |
| | 5 | | 1 | | | | | |

$$X = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

# Sudoku - Example

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | 3 | | 6 | |
| | | | | | | | 1 | |
| | 9 | 7 | 5 | | | | 8 | |
| | | | | 9 | | 2 | | |
| | | 8 | | 7 | | 4 | | |
| | | 3 | | 6 | | | | |
| | 1 | X | | | 2 | 8 | 9 | |
| | 4 | | | | | | | |
| | 5 | | 1 | | | | | |

$$X = \{2, 3, 6, 7, 8, 9\}$$

# Sudoku - Example

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | 3 | | 6 | |
| | | | | | | | 1 | |
| | 9 | 7 | 5 | | | | 8 | |
| | | | | 9 | | 2 | | |
| | | 8 | | 7 | | 4 | | |
| | | 3 | | 6 | | | | |
| | 1 | X | | | 2 | 8 | 9 | |
| | 4 | | | | | | | |
| | 5 | | 1 | | | | | |

$$X = \{2, 3, 6, 7, 8, 9\}$$

# Sudoku - Example

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | 3 | | 6 | |
| | | | | | | | 1 | |
| | 9 | 7 | 5 | | | | 8 | |
| | | | | 9 | | 2 | | |
| | | 8 | | 7 | | 4 | | |
| | | 3 | | 6 | | | | |
| | 1 | X | | | 2 | 8 | 9 | |
| | 4 | | | | | | | |
| | 5 | | 1 | | | | | |

$$X = \{2, 3, 6, 7, 8, 9\}$$

# Sudoku - Example

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | 3 | | 6 | |
| | | | | | | | 1 | |
| | 9 | 7 | 5 | | | | 8 | |
| | | | | 9 | | 2 | | |
| | | 8 | | 7 | | 4 | | |
| | | 3 | | 6 | | | | |
| | 1 | X | | | 2 | 8 | 9 | |
| | 4 | | | | | | | |
| | 5 | | 1 | | | | | |

$$X = \{3, 6, 7\}$$

# Sudoku - Example

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | 3 | | 6 | |
| | | | | | | | 1 | |
| | 9 | 7 | 5 | | | | 8 | |
| | | | | 9 | | 2 | | |
| | | 8 | | 7 | | 4 | | |
| | | 3 | | 6 | | | | |
| | 1 | X | | | 2 | 8 | 9 | |
| | 4 | | | | | | | |
| | 5 | | 1 | | | | | |

$$X = \{3, 6, 7\}$$

# Sudoku - Example

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | 3 | | 6 | |
| | | | | | | | 1 | |
| | 9 | 7 | 5 | | | | 8 | |
| | | | | 9 | | 2 | | |
| | | 8 | | 7 | | 4 | | |
| | | 3 | | 6 | | | | |
| | 1 | X | | | 2 | 8 | 9 | |
| | 4 | | | | | | | |
| | 5 | | 1 | | | | | |

$$X = \{3, 6, 7\}$$

# Sudoku - Example

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | 3 | | 6 | |
| | | | | | | | 1 | |
| | 9 | 7 | 5 | | | | 8 | |
| | | | | 9 | | 2 | | |
| | | 8 | | 7 | | 4 | | |
| | | 3 | | 6 | | | | |
| | 1 | X | | | 2 | 8 | 9 | |
| | 4 | | | | | | | |
| | 5 | | 1 | | | | | |

$$X = \{6\}$$

# Sudoku - Example

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | 3 | | 6 | |
| | | | | | | | 1 | |
| | 9 | 7 | 5 | | | | 8 | |
| | | | | 9 | | 2 | | |
| | | 8 | | 7 | | 4 | | |
| | | 3 | | 6 | | | | |
| | 1 | 6 | | | 2 | 8 | 9 | |
| | 4 | | | | | | | |
| | 5 | | 1 | | | | | |

# Sudoku - Example

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | 3 | | 6 | |
| | | | | | | | 1 | |
| | 9 | 7 | 5 | | | | 8 | |
| | | | | 9 | | 2 | | |
| | | 8 | | 7 | | 4 | | |
| | | 3 | | 6 | | | | |
| | 1 | 6 | | | 2 | 8 | 9 | |
| | 4 | | | | | | | |
| | 5 | | 1 | | | | | |

# Sudoku - Example

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 8 | | | | 3 | | 6 | |
| | 3 | | | | | | 1 | |
| | 9 | 7 | 5 | | | 3 | 8 | |
| | | | | 9 | | 2 | | |
| | | 8 | | 7 | | 4 | | |
| | | 3 | | 6 | | | | |
| | 1 | 6 | | | 2 | 8 | 9 | |
| | 4 | 9 | | | | | 2 | |
| | 5 | 2 | 1 | | 9 | | 4 | |

# Sudoku - Solving with CP

- Solving Sudoku using CP
- Defining the variables
- Defining the constraints

# Sudoku - Solving with CP

- Solving Sudoku using CP
- Defining the variables
- Defining the constraints
- I will use the MiniZinc modelling language
  - `http://www.minizinc.org/`
  - High level modelling for CP
  - Model in MiniZinc solvable using many different systems

# Sudoku - Defining the variables

```
array[1..9,1..9] of var 1..9 :
    puzzle = [|
        _, _, _, _, _, 3, _, 6, _|
        _, _, _, _, _, _, _, 1, _|
        _, 9, 7, 5, _, _, _, 8, _|
        _, _, _, _, 9, _, 2, _, _|
        _, _, 8, _, 7, _, 4, _, _|
        _, _, 3, _, 6, _, _, _, _|
        _, 1, _, _, _, 2, 8, 9, _|
        _, 4, _, _, _, _, _, _, _|
        _, 5, _, 1, _, _, _, _, _|
    |];
```

# Sudoku - Rules for rows and columns

```
% In all columns, all values different
constraint forall (col in 1..9) (
           all_different (row in 1..9)
              (puzzle[row, col])
      );

% In all rows, all values different
constraint forall (row in 1..9) (
           all_different (col in 1..9)
              (puzzle[row, col])
      );
```

# Sudoku - Rules for squares

```
% In all squares, all values different
constraint forall (row,col in {1,4,7}) (
           all_different (i,j in 0..2)
               (puzzle[row+i, col+j])
      );
```

# Sudoku - Search and output a solution

```
solve satisfy;

output [ show(puzzle[i,j]) ++
         if j = 9 then "\n"
         else " "
         endif
       | i,j in 1..9 ];
```

# Sudoku - Full program

```
include "globals.mzn";
array[1..9,1..9] of var 1..9 : puzzle = [|
        _, _, _, _, _, 3, _, 6, _|
        _, _, _, _, _, _, _, 1, _|
        _, 9, 7, 5, _, _, _, 8, _|
        _, _, _, _, 9, _, 2, _, _|
        _, _, 8, _, 7, _, 4, _, _|
        _, _, 3, _, 6, _, _, _, _|
        _, 1, _, _, _, 2, 8, 9, _|
        _, 4, _, _, _, _, _, _, _|
        _, 5, _, 1, _, _, _, _, _|
    |];
% In all columns, all values different
constraint forall (col in 1..9) (
           all_different (row in 1..9) (puzzle[row, col]) :: domain
       );
% In all rows, all values different
constraint forall (row in 1..9) (
           all_different (col in 1..9) (puzzle[row, col]) :: domain
       );
% In all squares, all values different
constraint forall (row,col in {1,4,7}) (
           all_different (i,j in 0..2) (puzzle[row+i, col+j]) :: domain
       );
solve satisfy;
output [ show(puzzle[i,j]) ++ if j = 9 then "\n" else " " endif
       | i,j in 1..9 ];
```

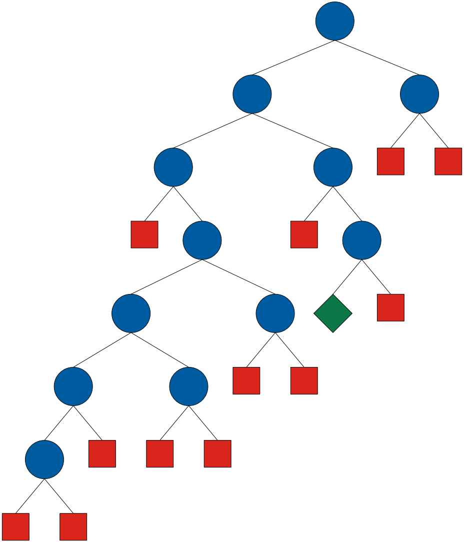
# Sudoku - Solution

```
1 8 5 9 2 3 7 6 4
2 3 4 6 8 7 5 1 9
6 9 7 5 1 4 3 8 2
4 7 1 3 9 8 2 5 6
9 6 8 2 7 5 4 3 1
5 2 3 4 6 1 9 7 8
3 1 6 7 4 2 8 9 5
7 4 9 8 5 6 1 2 3
8 5 2 1 3 9 6 4 7
----------
...
```

# Sudoku - Statistics

```
...
%%  runtime:        0.001 (1.412 ms)
%%  solvetime:      0.000 (0.411 ms)
%%  solutions:      1
%%  variables:      81
%%  propagators:    27
%%  propagations:   379
%%  nodes:          12
%%  failures:       5
%%  peak depth:     4
%%  peak memory:    100 KB
```

# Sudoku - Search tree



# What is Constraint programming?

- A way to model combinatorial (optimization) problems
- Systems for solving such problems
- A programming paradigm
- Theoretical model used in complexity

# Constraint programming for modelling

- Modelling combinatorial (optimization) problems
- Problems are typically complex (NP-hard)
- Language for describing models and instances
- Solving using different techniques
  - dedicated constraint programming systems most common

# Constraint programming systems

- Uses modeled structure for smart search
- Problems are typically complex (NP-hard)
  A CP system is no silver bullet
- Often implemented as libraries
  - Commercial: IBM CP Optimizer (C++), Xpress Kalis, Opturion CPX (C++), Sicstus Prolog (C), ...
  - Free: Gecode (C++), Choco (Java), OscaR (Scala), Google or-tools (C++), Minion (C++), Jacop (Java), ...

# Constraint programming as a paradigm

- Constraint Logic Programming
  - Mix of libraries and language
  - Search through Prolog search
  - Sicstus Prolog, ECLiPSe, BProlog, ...
- Constraint Handling Rules
  - Language for expressing constraints
- Other languages such as Mozart/Oz
  - Constraints embedded into base language
  - For example, used as synchronization mechanism between threads

# Constraint programming as theoretical model

- Using theoretical models of constraint problems for complexity research
- Not as interesting for solving practical problems
- Not my area, Per Austrin knows this better

# Constraint programming in my view

- Both models and systems
- Strong and structured way for modelling problems
- Systems turn-key solution in many cases
- Algorithmic middleware connecting smart independent components
- Base for implementing custom solutions
- Interesting research topic

# Constraint programming use cases

- Combinatorial problems
  - Puzzles, combinatorial design problems, ...
- Scheduling
  - Manufacturing, Railways, Air-line plane assignments, ...
- Personel rostering
  - Air-line crews, Hospital staff, ...
- Planning
  - Vehicle routing, Laser cut path planning, Disaster evacuation, ...
- Bioinformatics
  - Protein folding, Gene sequencing, ...
- Testing
  - Hardware verification test planning, Covering sets, Abstract interpretation, ...
- Various
  - Wine blending, TV-schedule selection, Music composition, ...

# Constraint programming basics

- Define variables
- Define constraints
- Draw conclusions from constraints and current values
- When all conclusions are made
  - Make a guess
  - Draw new conclusions
  - When inconsistency detected, backtrack

# Constraint programming - variables

- Finite set of variables
- Variable represents an unknown value from a set
- Finite domain variables
  - Integers
    E.g., $x$ some value from $\{2, 3, 5, 7, 11, 13, 17, 19, 23, 29\}$
  - Sets
    E.g., $y$ subset of $\{4, 8, 15, 16, 23, 42\}$
  - Boolean, Tasks, Graphs, Relations, Strings, ...
- Continous domain
  - Float variables
  - Upper and lower bound, solve to certain precision

# Constraint programming - constraints

- Basic constraints
  - Domain $x \in S$, $x \neq v$
  - Arithmetic $\sum_i a_i * x_i \leq d$, $\sqrt{x} = y$, $x \cdot y > z$, $\ldots$
  - Element/array indexing ($x[y] = z$)
- Logical constraints
  - $\wedge_i b_i = c$, $a \oplus b = c$, $\ldots$
  - $x + y = z \Leftrightarrow b$ (reified constraints)
- Structural (global) constraints
  - All different ($\forall_{i,j|i \neq j} x_i \neq x_j$)
  - Global cardinality, binpacking, regular language, disjunctive and cumulative resource usage, no overlap, hamiltonian cycle, matching, minimum spanning tree, extensional ...
  - Encapsulates re-occuring sub-structures
  - 350+ defined in Global Constraints Catalogue

# Constraint programming - constraints

- Constraints are implemented as propagators
- Propagators look at current domains, and make deductions
- Example: Linear in-equality
  - $x + 2 \cdot y < z \quad x, y \in \{2..10\}, z \in \{4..10\}$
  - Propagation deduces $x \in \{2..6\}, y \in \{2..4\}, z \in \{6..10\}$
- Example: All different
  - $\mathrm{alldifferent}(x, y, z, v)$
  - $x \in \{1, 2\}, y \in \{1, 2\}, z \in \{1, 2, 3, 4\}, v \in \{2, 4\}$
  - $x \in \{1, 2\}, y \in \{1, 2\}, z \in \{\cancel{1}, \cancel{2}, 3, 4\}, v \in \{\cancel{2}, 4\}$
  - $x \in \{1, 2\}, y \in \{1, 2\}, z \in \{3, 4\}, v \in \{4\}$
  - $x \in \{1, 2\}, y \in \{1, 2\}, z \in \{3, \cancel{4}\}, v \in \{4\}$
  - $x \in \{1, 2\}, y \in \{1, 2\}, z \in \{3\}, v \in \{4\}$

# Constraint programming - constraints

- Express high-level intent
- In the best case, propagation removes all variables not in any solution.
- Global constraints encapsulate smart algorithms
  - All different uses bipartite matching and strongly connected components
  - Global cardinality uses flow algorithms
  - Symmetric all different uses general matching
  - Cumulative uses edge-finding, time-tabling, and not-first/not-last reasoning
  - Binpacking uses dynamic programming
  - ...

# Constraint programming - search

- Search = Branching + Exploration order
- Branching is heuristic choice
  - Defines shape of search tree
  - Smallest domain, minimum regret, smallest domain/accumulated failure count, activity based, ...
  - Custom problem specific heuristics
- Exploration order
  - Explore search tree induced by branching
  - Depth first, Limited discrepancy, Best first, Depth bounded, ...
  - Sequential, Restarts, Parallel, ...
  - Large neighborhood search

# Constraint programming benefits

- Concise and natural models
- Often good performance
- Custom search and hybridizations
- Describe the problem, the computer figures out how to solve it.

# Constraint programming draw backs

- Complex behaviour, small changes may result in large differences
- The best model is not the simplest model
- Automatic search and symmetry breaking just starting
  - New features makes systems more complex
- Debugging constraint models is hard
- When more specialized systems work, they are often more efficient

# Constraint programming alternatives

- Constraint programming as modelling language is very expressive
- Systems must handle generality
- Limiting expressiveness can lead to more effective systems
- Using CP style models translating to base language
  - Simpler models than in base language
  - Easier modelling and debugging
  - Not widely used

# Linear Programming

- Restrictions
  - Only float variables
  - Only linear in-equality constraints ($\sum_i a_i * x_i \leq d$)
- Mathematical optimization
- Most common method in industry and research
- Algorithms are polynomial
- Systems are *very* good
  - Commercial: IBM CPLEX, Gurobi, Mosek, ...
  - Free: COIN-OR, lp_solve, glpk, ...
- Some things are hard to express, leading to very large models
- More information: KTH Course SF1841 Optimization

# Mixed Integer Programming

- Restrictions
  - Float variables, some restricted to be integers
  - Only linear in-equality constraints ($\sum_i a_i * x_i \leq d$)
- Also very common in industry and research
- Algorithms are *not* polynomial
- Linear relaxation (disregarding integer requirements) guides search
- Same systems as for Linear Programming
- Huge increase in performance last 15 years ($>$ x1000)

# Difference between CP and MIP models

- Consider $n$ variables $x$ from 1 to $m$ and an all different constraint.
- Constraint programming

$$x = \langle x_1, x_2 \ldots, x_n \rangle, x_i \in \{1, \ldots, m\}$$
$$\mathrm{alldifferent(x)}$$

- Mixed Integer Programming

$$x = \langle x_{11}, x_{12}, \ldots, x_{1m}, x_{21}, \ldots, x_{nm} \rangle, x_{ij} \in \{0, 1\}$$

$$\forall j \in \{1..m\} \sum_{i=1}^{n} x_{i,j} = 1 \qquad \forall i \in \{1..n\} \sum_{j=1}^{m} x_{i,j} \leq 1$$

# SAT

- Restrictions
  - Only Boolean variables
  - Only simple or clauses
- Systems are highly optimized
- Suitable for some types of problems
- Common in industry and research
- Algorithms are *not* polynomial
- Explanations for failures, cheap restarts, activity based search
- More information: Print and read source of MiniSat (`http://minisat.se/`)

# Satisfiability Modulo Theories

- Restrictions
  - Boolean variables and simple or clauses
  - Algebraic theory (equality, arithmetic, ...)
- Makes SAT systems more usable
- Middle ground between SAT and CP
- Common in industry and research
- Algorithms are *not* polynomial
- Extensively used at Microsoft (https://github.com/Z3Prover/z3)

# Local Search

- Move between (potentially invalid) solutions
  - Local moves and evaluation
  - Meta heuristics: Simulated annealing, Tabu search, ...
  - Population based: genetic, ant colony, particle swarm, ...
- No guarantees that solution will be found
- Very often ad-hoc and unprincipled
- Can be very effective
- Constraint Based Local Search combines modelling of CP with local search
- CBLS Systems: Comet, OscaR

# Summary

- Constraint programming is a way to model and solve combinatorial optimization problems
- Even if CP systems are not used, modelling abstractions are important
- Fun way to solve puzzles
- Useful both in research and in industry
- More information: KTH Course ID2204 Constraint Programming