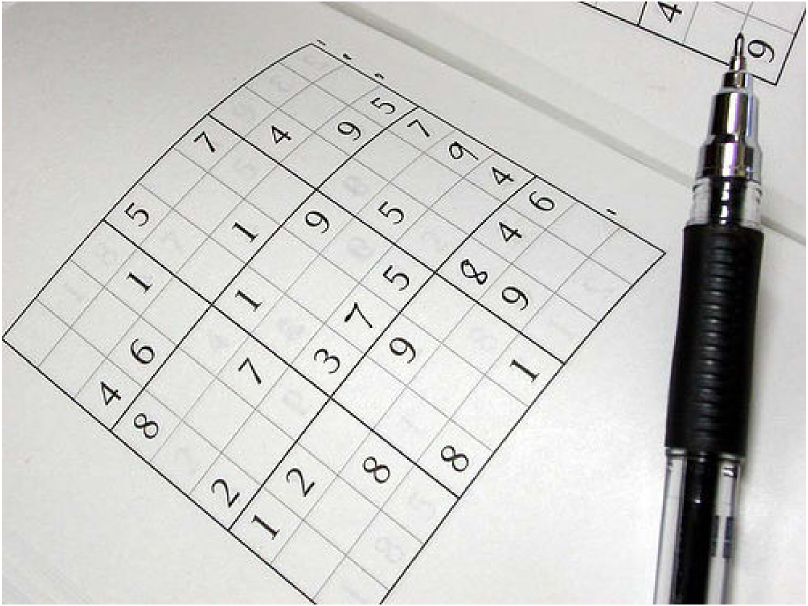
Constraint Programming

Mikael Z. Lagerkvist

Tomologic

October 2016

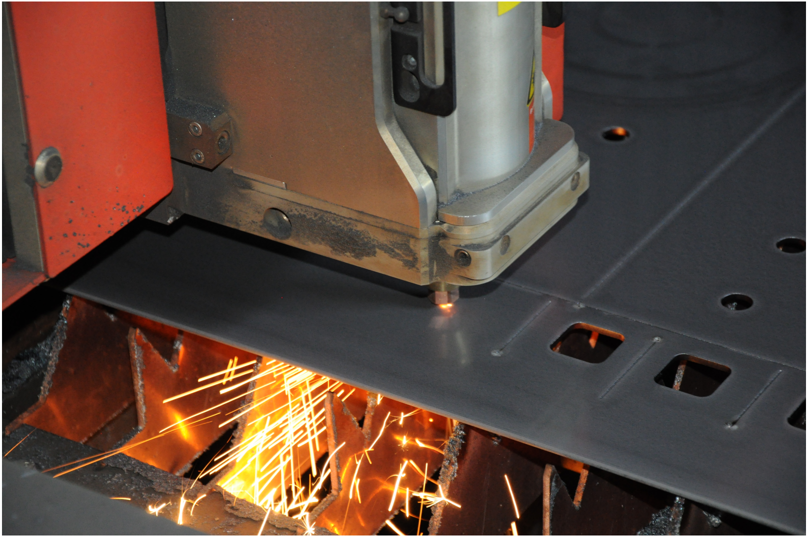


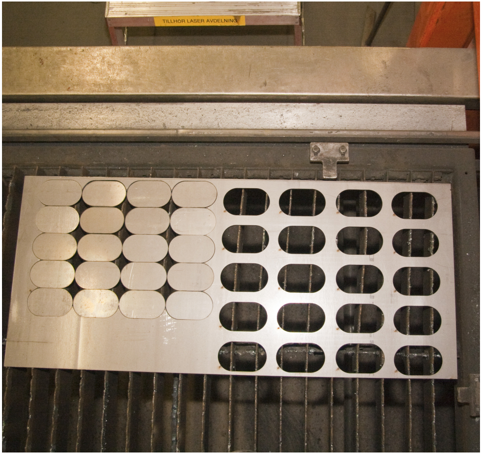
[flickr.com/photos/santos/]

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

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
- 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 ];
```

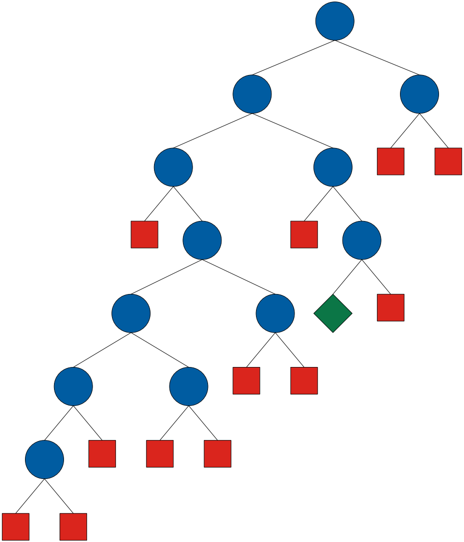
Sudoku - Solution

```
$ mzn-gecode -a -s sudoku.mzn
    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 interesting for solving practical problems
- Not my area

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 and rostering
  - Manufacturing, Railways, Air-line plane assignments, 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
  - Compilation, 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 $\sum_i a_i * x_i \leq d$, $\sqrt{x} = y$, $x \cdot y > z$, ...
  - Element/array indexing ($x[y] = z$)
- Logical constraints
  - $\wedge_i b_i = c$, $a \oplus b = c$, ...
  - $x + y = z \Leftrightarrow b$ (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

Send More Money

- SEND + MORE = MONEY
- One digit per letter
- Each letter different digit
- Don't start with zeroes

SMM - Variables

```
set of int: Digits = 0..9;
var Digits: S;
var Digits: E;
var Digits: N;
var Digits: D;
var Digits: M;
var Digits: O;
var Digits: R;
var Digits: Y;
array[1..8] of var int : letters =
  [S,E,N,D,M,O,R,Y];
```

SMM - Constraints

```
constraint all_different(letters);

constraint  1000*S + 100*E + 10*N + D  +
            1000*M + 100*O + 10*R + E  =
  10000*M + 1000*O + 100*N + 10*E + Y;

constraint S > 0;

constraint M > 0;
```

SMM - Search and output a solution

```
solve satisfy;

output [
   "S:", show(S),  " E:", show(E),
   " N:", show(N), " D:", show(D),
   " M:", show(M), " O:", show(O),
   " R:", show(R), " Y:", show(Y),
   "\n\n",
   "    ", show(S), show(E), show(N), show(D),"\n",
   " +  ", show(M), show(O), show(R), show(E),"\n",
   " = ",  show(M), show(O), show(N), show(E), show(Y),
   "\n"
];
```

SMM - Full program

```
include "globals.mzn";

set of int: Digits = 0..9;

var Digits: S;
var Digits: E;
var Digits: N;
var Digits: D;
var Digits: M;
var Digits: O;
var Digits: R;
var Digits: Y;

array[1..8] of var int : letters =
  [S,E,N,D,M,O,R,Y];

constraint all_different(letters);

constraint  1000*S + 100*E + 10*N + D  +
            1000*M + 100*O + 10*R + E  =
  10000*M + 1000*O + 100*N + 10*E + Y;

constraint S > 0;

constraint M > 0;

solve satisfy;

output [
   "S:", show(S), " E:", show(E), " N:", show(N), " D:", show(D),
   " M:", show(M), " O:", show(O), " R:", show(R), " Y:", show(Y),
   "\n\n",
   "    ", show(S), show(E), show(N), show(D),"\n",
   " +  ", show(M), show(O), show(R), show(E),"\n",
   " = ",  show(M), show(O), show(N), show(E), show(Y), "\n"
];
```

SMM - Solution

```
$ mzn-gecode -a -s sendmoremoney.mzn
S:9 E:5 N:6 D:7 M:1 O:0 R:8 Y:2

    9567
 +  1085
 = 10652
----------
==========
%%  runtime:       0.000 (0.507 ms)
%%  solvetime:     0.000 (0.133 ms)
%%  solutions:     1
%%  variables:     8
%%  propagators:   2
%%  propagations:  20
%%  nodes:         7
%%  failures:      3
%%  restarts:      0
%%  peak depth:    1
```

SMM - Alternative constraints

```
...

var int: SEND;
var int: MORE;
var int: MONEY;

constraint            1000*S + 100*E + 10*N + D =  SEND;
constraint            1000*M + 100*O + 10*R + E =  MORE;
constraint 10000*M + 1000*O + 100*N + 10*E + Y = MONEY;

constraint SEND + MORE = MONEY;

...
```

SMM - Alternative model solution

```
$ mzn-gecode -a -s sendmoremoney-alternative.mzn
S:9 E:5 N:6 D:7 M:1 O:0 R:8 Y:2

    9567
 +  1085
 = 10652
----------
==========
%%  runtime:       0.012 (12.622 ms)
%%  solvetime:     0.002 (2.018 ms)
%%  solutions:     1
%%  variables:     11
%%  propagators:   5
%%  propagations:  147
%%  nodes:         11
%%  failures:      5
%%  restarts:      0
%%  peak depth:    1
```

Send Most Money

- SEND + MOST = MONEY
- One digit per letter
- Each letter different digit
- Don't start with zeroes
- Maximize the amount of money

Most Money - Variables

```
set of int: Digits = 0..9;
var Digits: S;
var Digits: E;
var Digits: N;
var Digits: D;
var Digits: M;
var Digits: O;
var Digits: T;
var Digits: Y;
array[1..8] of var int : letters =
  [S,E,N,D,M,O,T,Y];
```

Most Money - Constraints

```
constraint all_different(letters);

constraint  1000*S + 100*E + 10*N + D  +
            1000*M + 100*O + 10*S + T  =
  10000*M + 1000*O + 100*N + 10*E + Y;
constraint S > 0;
constraint M > 0;

var int: value =
    10000*M + 1000*O + 100*N + 10*E + Y;
```

Most Money - Search and output

```
solve maximize value;

output [
   "S:", show(S),  " E:", show(E),
   " N:", show(N), " D:", show(D),
   " M:", show(M), " O:", show(O),
   " T:", show(T), " Y:", show(Y),
   "\n\n",
   "    ", show(S), show(E), show(N), show(D),"\n",
   " +  ", show(M), show(O), show(S), show(T),"\n",
   " = ",  show(M), show(O), show(N), show(E), show(Y),
   "\n"
];
```

Most Money - Full program

```
include "globals.mzn";

set of int: Digits = 0..9;

var Digits: S;
var Digits: E;
var Digits: N;
var Digits: D;
var Digits: M;
var Digits: O;
var Digits: T;
var Digits: Y;

array[1..8] of var int : letters =
  [S,E,N,D,M,O,T,Y];

constraint all_different(letters);

constraint  1000*S + 100*E + 10*N + D  +
            1000*M + 100*O + 10*R + E  =
  10000*M + 1000*O + 100*N + 10*E + Y;
constraint S > 0;
constraint M > 0;

var int: value =
    10000*M + 1000*O + 100*N + 10*E + Y;

solve maximize value;

output [
   "S:", show(S), " E:", show(E), " N:", show(N), " D:", show(D),
   " M:", show(M), " O:", show(O), " T:", show(T), " Y:", show(Y),
   "\n\n",
   "    ", show(S), show(E), show(N), show(D),"\n",
   " +  ", show(M), show(O), show(S), show(T),"\n",
   " = ",  show(M), show(O), show(N), show(E), show(Y), "\n"
];
```

Most Money - Solution

```
$ mzn-gecode -a -s sendmostmoney.mzn
...
S:9 E:6 N:7 D:5 M:1 O:0 T:3 Y:8

    9675
 +  1093
 = 10768
----------
S:9 E:7 N:8 D:3 M:1 O:0 T:2 Y:5

    9783
 +  1092
 = 10875
----------
S:9 E:7 N:8 D:4 M:1 O:0 T:2 Y:6

    9784
 +  1092
 = 10876
----------
==========
```

Most Money - Statistics

```
%%  runtime:       0.018 (18.043 ms)
%%  solvetime:     0.003 (3.437 ms)
%%  solutions:     8
%%  variables:     9
%%  propagators:   3
%%  propagations:  154
%%  nodes:         41
%%  failures:      13
%%  restarts:      0
%%  peak depth:    4
```

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

Bonus example - Social golfers

- Classic scheduling example
- Club with golfers playing tournament
- Each week golfers play in new groups
- All golfers play every week
- Goal: find schedule for p players in g groups of size p/g for w weeks

Social golfers example

- Golfers: Alice, Bob, Cara, David, Erica, Finley, Gretchen, Hamley, Ingrid
- Three person per group, four weeks of play
- $\{A, B, C\}$ $\{D, E, F\}$ $\{G, H, I\}$
- $\{C, D, H\}$ $\{B, F, G\}$ $\{A, E, I\}$
- $\{C, E, G\}$ $\{B, D, I\}$ $\{A, F, H\}$
- $\{C, F, I\}$ $\{B, E, H\}$ $\{A, D, G\}$

Set variables

- Set of integers from some universe
- $S \subseteq \{1, 2, \ldots, n\}$
- Simple constraints $x \in S$, $|S| = y$
- Set relations $S \cup T \subset U, \ldots$
- Global set constraints (all disjoint, range, roots, value precedence, partition, ...)
- Higher level modeling

Golfers - Problem definition

Giving names to the amounts and sets

```
int: weeks;
int: groups;
int: group_size;
int: golfers = groups * group_size;

set of int: Weeks = 1..weeks;
set of int: Groups = 1..groups;
set of int: Golfers = 1..golfers;
set of int: GroupSize = 1..group_size;
```

Golfers - Data file

Stored in separate file golf443.dzn

```
% Problem instance
weeks = 4;
groups = 4;
group_size = 3;
```

Golfers - Variables

Matrix of variables representing the groups.

```
array[Weeks,Groups] of
     var set of Golfers:
        schedule;
```

Golfers - Right size groups

Each group must have right cardinality

```
constraint
forall (week in Weeks, group in Groups) (
   card(schedule[week,group]) = group_size
);
```

Golfers - Play once per week

Two groups in one week share no players

```
constraint
forall (week in Weeks) (
   forall (group1, group2 in Groups
           where group1 < group2) (
      schedule[week,group1]
        intersect
      schedule[week,group2]
      = {}
   )
);
```

Golfers - All players every week

Every week all players get to play

```
constraint
forall (week in Weeks) (
   partition_set(
      [schedule[week,group] | group in Groups],
      Golfers
   )
);
```

Golfers - No replays

Two groups from different weeks share at most one player

```
constraint
forall (week1, week2 in Weeks
        where week1 < week2) (
   forall (group1, group2 in Groups) (
      card(
         schedule[week1,group1]
            intersect
         schedule[week2,group2]
      ) <= 1
   )
);
```

Golfers - Search and output

Standard search and simple output (one line per week)

```
solve satisfy;

output
[ if group == 1
      then "Week " ++ show(week) ++ ": "
      else "" endif ++
  show(schedule[week,group]) ++ " " ++
  if group == groups then "\n" else "" endif
  | week in Weeks, group in Groups
];
```

Golfers - First solution

```
$ mzn-gecode -s social.mzn golf443.dzn
Week 1: {5,11,12} {3,4,8} {2,7,10} {1,6,9}
Week 2: {6,7,11} {3,5,10} {2,4,9} {1,8,12}
Week 3: {6,10,12} {3,9,11} {2,5,8} {1,4,7}
Week 4: {8,10,11} {7,9,12} 4..6 1..3
----------
%%  runtime:       29.833 (29833.078 ms)
%%  solvetime:     29.820 (29820.188 ms)
%%  solutions:     1
%%  variables:     248
%%  propagators:   220
%%  propagations:  74940759
%%  nodes:         762337
%%  failures:      381153
%%  restarts:      0
%%  peak depth:    41
```

Golfers - Parallel search

```
$ mzn-gecode -s -p 4 social.mzn golf443.dzn
Week 1: {5,7,12} {3,4,9} {2,6,10} {1,8,11}
Week 2: {5,9,10} {3,11,12} {2,4,8} {1,6,7}
Week 3: {6,9,12} {3,7,8} {2,5,11} {1,4,10}
Week 4: {8,10,12} {7,9,11} 4..6 1..3
----------
%%  runtime:       1.138 (1138.283 ms)
%%  solvetime:     1.126 (1126.825 ms)
%%  solutions:     1
%%  variables:     248
%%  propagators:   220
%%  propagations:  4959569
%%  nodes:         52228
%%  failures:      26077
%%  restarts:      0
%%  peak depth:    39
```

Symmetry breaking

- Social golfers has symmetries
  - Solutions can be transformed simply
  - Only care about *some* solution
- Golfer names symmetric
- Order of groups in week is symmetric
- Order of weeks is symmetric

Golfer names symmetric

- Base solution
  - $\{A, B, C\}$ $\{D, E, F\}$ $\{G, H, I\}$
  - $\{C, D, H\}$ $\{B, F, G\}$ $\{A, E, I\}$
  - $\{C, E, G\}$ $\{B, D, I\}$ $\{A, F, H\}$
  - $\{C, F, I\}$ $\{B, E, H\}$ $\{A, D, G\}$
- Switched A and F
  - $\{F, B, C\}$ $\{D, E, A\}$ $\{G, H, I\}$
  - $\{C, D, H\}$ $\{B, A, G\}$ $\{F, E, I\}$
  - $\{C, E, G\}$ $\{B, D, I\}$ $\{F, A, H\}$
  - $\{C, A, I\}$ $\{B, E, H\}$ $\{F, D, G\}$

Group order symmetry

- Base solution
  - $\{A, B, C\}$ $\{D, E, F\}$ $\{G, H, I\}$
  - $\{C, D, H\}$ $\{B, F, G\}$ $\{A, E, I\}$
  - $\{C, E, G\}$ $\{B, D, I\}$ $\{A, F, H\}$
  - $\{C, F, I\}$ $\{B, E, H\}$ $\{A, D, G\}$
- Re-ordered first two groups
  - $\{D, E, F\}$ $\{A, B, C\}$ $\{G, H, I\}$
  - $\{C, D, H\}$ $\{B, F, G\}$ $\{A, E, I\}$
  - $\{C, E, G\}$ $\{B, D, I\}$ $\{A, F, H\}$
  - $\{C, F, I\}$ $\{B, E, H\}$ $\{A, D, G\}$

Week order symmetry

- Base solution
  - $\{A, B, C\}$ $\{D, E, F\}$ $\{G, H, I\}$
  - $\{C, D, H\}$ $\{B, F, G\}$ $\{A, E, I\}$
  - $\{C, E, G\}$ $\{B, D, I\}$ $\{A, F, H\}$
  - $\{C, F, I\}$ $\{B, E, H\}$ $\{A, D, G\}$
- Re-ordered first and last week
  - $\{A, B, C\}$ $\{D, E, F\}$ $\{G, H, I\}$
  - $\{C, D, H\}$ $\{B, F, G\}$ $\{A, E, I\}$
  - $\{C, E, G\}$ $\{B, D, I\}$ $\{A, F, H\}$
  - $\{C, F, I\}$ $\{B, E, H\}$ $\{A, D, G\}$

Golfers - Group order symmetry

Idea: Groups in week can be ordered

```
constraint
forall (week in Weeks) (
   forall (group1, group2 in Groups
           where group1 < group2) (
      schedule[week, group1]
         >
      schedule[week, group2]
   )
);
```

Golfers - Group order symmetry

```
$ mzn-gecode -s social.mzn golf443.dzn
Week 1: {1,7,11} {2,5,10} {3,6,9} {4,8,12}
Week 2: {1,6,10} {2,4,9} {3,7,8} {5,11,12}
Week 3: {1,4,5} {2,6,7} {3,10,12} {8,9,11}
Week 4: 1..3 {4,10,11} {5,6,8} {7,9,12}
----------
%%  runtime:       1.090 (1090.510 ms)
%%  solvetime:     1.086 (1086.382 ms)
%%  solutions:     1
%%  variables:     248
%%  propagators:   244
%%  propagations:  2639492
%%  nodes:         24363
%%  failures:      12172
%%  restarts:      0
%%  peak depth:    38
```

Golfers - First week symmetry

Idea: First week can be fixed statically

```
constraint
forall (group in Groups, i in GroupSize) (
   ((group-1)*group_size + i)
      in
   schedule[1,group]
);
```

Golfers - First week symmetry

```
$ mzn-gecode -s social.mzn golf443.dzn
Week 1: 1..3 4..6 7..9 10..12
Week 2: {6,9,10} {3,4,8} {2,7,11} {1,5,12}
Week 3: {6,7,12} {3,5,11} {2,4,9} {1,8,10}
Week 4: {6,8,11} {3,9,12} {2,5,10} {1,4,7}
----------
%%  runtime:       0.177 (177.424 ms)
%%  solvetime:     0.166 (166.398 ms)
%%  solutions:     1
%%  variables:     248
%%  propagators:   213
%%  propagations:  412250
%%  nodes:         4071
%%  failures:      2025
%%  restarts:      0
%%  peak depth:    27
```

Golfers - Search annotation

Idea: Search on first group per week first

```
solve ::
   set_search(
      [schedule[week,1] | week in Weeks] ++
      [schedule[week,group]
       | week in Weeks, group in Groups],
      input_order, indomain_min, complete)
satisfy;
```

Golfers - Search annotation

```
$ mzn-gecode -s -p 4 social.mzn golf443.dzn
Week 1: {1,10,11} {2,5,7} {3,6,12} {4,8,9}
Week 2: {1,6,7} {2,4,12} {3,8,11} {5,9,10}
Week 3: {1,4,5} {2,6,11} {3,7,9} {8,10,12}
Week 4: 1..3 {4,7,10} {5,6,8} {9,11,12}
----------
%%  runtime:       0.036 (36.828 ms)
%%  solvetime:     0.030 (30.782 ms)
%%  solutions:     1
%%  variables:     248
%%  propagators:   244
%%  propagations:  101909
%%  nodes:         1030
%%  failures:      493
%%  restarts:      0
%%  peak depth:    35
```

Golfers - All features

```
$ mzn-gecode -s social.mzn golf443.dzn
Week 1: 1..3 4..6 7..9 10..12
Week 2: {1,4,7} {2,5,10} {3,9,11} {6,8,12}
Week 3: {1,5,8} {2,6,11} {3,7,12} {4,9,10}
Week 4: {1,6,9} {2,4,12} {3,8,10} {5,7,11}
----------
%%  runtime:       0.016 (16.187 ms)
%%  solvetime:     0.004 (4.421 ms)
%%  solutions:     1
%%  variables:     248
%%  propagators:   231
%%  propagations:  4625
%%  nodes:         63
%%  failures:      21
%%  restarts:      0
%%  peak depth:    23
```

Golfers - All features and parallel

```
$ mzn-gecode -s -p 4 social.mzn golf443.dzn
Week 1: 1..3 4..6 7..9 10..12
Week 2: {1,4,7} {2,5,10} {3,9,11} {6,8,12}
Week 3: {1,5,8} {2,6,11} {3,7,12} {4,9,10}
Week 4: {1,6,9} {2,4,12} {3,8,10} {5,7,11}
----------
%%  runtime:       0.008 (8.116 ms)
%%  solvetime:     0.002 (2.617 ms)
%%  solutions:     1
%%  variables:     248
%%  propagators:   231
%%  propagations:  4693
%%  nodes:         64
%%  failures:      22
%%  restarts:      0
%%  peak depth:    23
```

Social golfers conclusions

- High level modeling with set variables
- Suprisingly hard problem
- Parallel search cheap to test
- Symmetry breaking is crucial but hard
  - Symmetry breakings might interfere with each other
  - Wrong order for group order symmetry would make first week fixing invalid
- Search is an art
  - Experimentation often required