Comparison of CP Systems

Håkan Kjellerstrand (hakank@gmail.com)
Private/Independent Researcher, Malmö

http://www.hakank.org/

My Constraint Programming Blog:
http://www.hakank.org/constraint_programming_blog/
Comparison of CP systems, for learning CP/CP system

Focus on:
* how easy is it to learn a system
* the modelling aspect

Criteria:
* syntax, element constraint, reification
* defining predicates (decompositions)
* community
* number of examples, documentation, etc

In this talk:
- strong/weak features (“likes”/”dislikes”)

Disclaimer: There are other criteria for selecting a CP system.
Common constraint problems

http://hakank.org/common_cp_problems/

- My “learning problems”
  I always start testing/learning a CP system with the same
  about 18 (or 30) problems.
  (And: report bugs/opinions to the developers.)

- Same approach → one can compare implementations
  in different CP systems.

- As of now:
  * 276 problems that is implemented in >= 2 systems
    (in total 1260 implemented models)
  * 73 problems in >= 6 systems.
  * 44 problems in >= 8 systems
  * 15 problems in >= 10 systems
### Constraint Programming systems

<table>
<thead>
<tr>
<th>System</th>
<th>Website</th>
<th>Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>MiniZinc</td>
<td>hakank.org/minizinc/</td>
<td>890</td>
</tr>
<tr>
<td>JaCoP</td>
<td>hakank.org/JaCoP/</td>
<td>18</td>
</tr>
<tr>
<td>Choco</td>
<td>hakank.org/choco/</td>
<td>20</td>
</tr>
<tr>
<td>Comet</td>
<td>hakank.org/comet/</td>
<td>166</td>
</tr>
<tr>
<td>Gecode</td>
<td>hakank.org/gecode/</td>
<td>162</td>
</tr>
<tr>
<td>Gecode/R</td>
<td>hakank.org/gecode_r/</td>
<td>29</td>
</tr>
<tr>
<td>ECLiPSe CLP</td>
<td>hakank.org/eclipse/</td>
<td>176</td>
</tr>
<tr>
<td>Tailor/Essence'</td>
<td>hakank.org/tailor/</td>
<td>29</td>
</tr>
<tr>
<td>SICStus Prolog</td>
<td>hakank.org/sicstus/</td>
<td>151</td>
</tr>
<tr>
<td>Zinc</td>
<td>hakank.org/minizinc/zinc.html</td>
<td>39</td>
</tr>
<tr>
<td>JaCoP/Scala</td>
<td>hakank.org/jacop/jacop_scala.html</td>
<td>38</td>
</tr>
<tr>
<td>or-tools/Python</td>
<td>hakank.org/or-tools/#python</td>
<td>203</td>
</tr>
<tr>
<td>or-tools/Java</td>
<td>hakank.org/or-tools/#java</td>
<td>36</td>
</tr>
<tr>
<td>or-tools/C#</td>
<td>hakank.org/or-tools/#csharp</td>
<td>122</td>
</tr>
</tbody>
</table>

[Answer Set Programming (“related paradigm”)](http://hakank.org/asp/) 84 models
G12 MiniZinc

- http://www.hakank.org/minizinc/
- Version 1.5.1

- FlatZinc, many solvers: G12 FD/LazyFD/CPX, Gecode, ECLiPSe, SICStus Prolog, JaCoP, Fzn2smt, Chuffed, fzntini, (beta: Choco, Bprolog), MIP solvers.
- ease of modelling: very high level, element: m[i,j], reification, logical operators
- documentation: specification + tutorial
- examples: many examples + benchmarks (some are mine), many test cases
- constraints: many (decompositions and solver dependent)
- community: not very active. Mailing list, Wiki, and SVN for public examples
- input/output: no support except data files
- search strategies: fixed (~ subset of ECLiPSe's), cannot write own
- other features:
  + float vars (solver dependent)
  + exists
- cannot write own propagators in MiniZinc (experimental: Search Combinators)
- no recursion, no “external loops”
- “fancy” output can be (very) slow
JaCoP

- http://jacop.osolpro.com/
- http://www.hakank.org/JaCoP/
- version 3.2

- ease of modelling: Java, Element can be tricky, "minimalistic" approach
- documentation: Nice tutorial, API from source code
- examples: many (6 are mine)
- constraints: many
- community: SourceForge forum, active
- input/output: Java
- search strategies: yes
- other features:
  + great class structure (easy to find)
  + FlatZinc
  + Scala interface (see separate entry)
  + Constraints: Geost, NetworkFlow, SoftAlldifferent, SoftGCC
- verbose (being a Java system)
Choco

- http://www.emn.fr/z-info/choco-solver/
- http://www.hakank.org/choco/
- version 2.1

- ease of modelling: Java, Element can be tricky
- documentation: great site, introduction material
- examples: many (but undocumented)
- constraints: many, nicely documented with example code
- community: active forum
- input/output: Java
- search strategies: yes
- other features:
  + FlatZinc (beta version)
  + SVN
  + float vars
  + Constraints: Geost, Tree, StretchPath
- class structure “cluttered” (hard to find constraints and methods)
Comet

- http://dynadec.com/
- http://www.hakank.org/comet/
- version: Update: 2.1.1 (from 2010)

- solvers: FD, LP/MIP, local search (slightly different syntax)
- ease of modelling: very high level (OPL like), element: m[i,j], reification, boolean operators, C++ like objects, etc
- documentation: great tutorial (also: OPL book & Comet book), examples: many documented examples
- constraints: many
- community: not very active forum nowadays
- input/output: C++ like
- search strategies: easy to writing own
- other features:
  + GUI (for local search)
  + debugger: text and GUI
  + scheduling (OPL like)
- license: commercial, academic or evaluation
- no new version since spring 2010
Gecode

- http://www.gecode.org/
- http://www.hakank.org/gecode/
- version: 3.7.3

- ease of modelling: C++, logical operators overloaded, Element sometimes tricky
- documentation: great site (searchable), great introduction
- examples: many (with bells & whistles, can be hard to read)
- constraints: many (overridden with same name)
- community: active
- input/output: C++, great command line options
- search strategies: many different built in
- other features:
  + very fast CP solver
  + Gist (interactive search tree)
  + great FlatZinc support
  + regular expressions
  + Constraints: Element for matrices, Cumulatives
- gotcha: Matrix view of integers m(cols, rows)
Gecode/R

- http://gecoder.rubyforge.org/
- http://www.hakank.org/gecode_r/
- version: For Gecode 2.2

- ease of modelling: Ruby, nice short cuts
- documentation: well structured site, API, introduction
- examples: not many (12 of which 3 are mine)
- constraints: not many, some missing from Gecode (e.g. gcc)
- community: not active
- input/output: Ruby
- search strategies: some of Gecode's
- other features:
  + Ruby's ease of handling arrays
  + regular expressions
- matrices can be tricky
- reifications can be tricky
- error messages can be confusing
- not updated with later Gecode versions
- discontinued?
ECLiPSe CLP

- http://www.eclipseclp.org/
- http://www.hakank.org/eclipse/
- version: 6.0

- ease of modelling: Prolog (CLP) with extensions: do-loops and arrays
- documentation: 2 books, great tutorials, reference, guides etc.
- examples: many
- constraints: quite many
- community: active
- input/output: Prolog
- search strategies: many, extensible
- other features:
  + two FD-solvers with set/float: ic and fd, MIP solver: eplex
  + FlatZinc (ic, fd, eplex)
  + GUI/shell for interactive debugging and tracing propagations
  + Propia: Generalized Propagation
- Prolog can be tricky (though do-loops/arrays makes it easier)
- element constraint can be tricky sometimes
Tailor/Essence'

- http://www.cs.st-andrews.ac.uk/~andrea/tailor/
- http://www.hakank.org/tailor/
- version: 0.3.2 (discontinued)

- translates Essence' models to other solvers: Minion, Gecode, FlatZinc
- ease of modelling: very high level (about as MiniZinc, Comet)
- documentation: basic tutorial
- examples: about 23 examples (some are mine)
- constraints: not many
- community: not active
- input/output: just data files
- search strategies: N/A (see below)
- other features:
  + "array slice", e.g. x[i, ...]
  - don't support predicates
  - cannot state search strategies for solvers
  - no full support of FlatZinc
  - discontinued
SICStus Prolog

- http://www.sics.se/isl/sicstuswww/site/index.html
- http://www.hakank.org/sicstus/
- version: 4.2.1

- ease of modelling: Prolog (with do-loops)
- documentation: extensive
- examples: many examples
- constraints: many
- community: active
- input/output: Prolog
- search strategies: many (can declare own)
- other features:
  + Fast CLP solver
  + FlatZinc support
  + SPIDER (GUI)
  + Constraints: smt, case, cumulatives, geost, automaton
- Prolog can be tricky (do-loops helps)
- license to use (evaluation versions exists)
G12 Zinc

- version: 2.0

- “big brother” of G12 MiniZinc
- ease of modelling: very high level
- documentation: not very much
- examples: not many Zinc specific examples (can use MiniZinc)
- constraints: may use MiniZinc's constraints
- community: not public active
- input/output: Zinc (MiniZinc) data files
- search strategies: many, cannot declare own direct (but more flexible)
- other features:
  + more data structures than MiniZinc: tuples, records, enum, “hash table”
  + more built-in predicates/functions than MiniZinc, can define functions
  + more flexible search strategies than MiniZinc
- no “external” loop constructs
- slow (compiling and running)
- no external solvers
JaCoP/Scala

- version: 1.0(?)

- Scala interface to JaCoP
- ease of modelling: High Level, Scala (many operators overloaded)
- documentation: not much
- examples: not many examples (21)
- constraints: All JaCoP's constraints
- community: not public active
- input/output: Scala I/O
- search strategies: using JaCoP's
- other features:
  + nice use of Scala features (overloading etc)
  - reification needs extra BoolVar
  - element must be stated explicitly (i.e. not overloaded)
Google or-tools/Python

- https://code.google.com/p/or-tools/
- http://www.hakank.org/or-tools/#python
- version: (SVN)

- Python interface to or-tools' C++ engine
- ease of modelling: Python with some extra sugar
- documentation: not much (doc project started Winter 2012)
- examples: many examples (most are mine)
- constraints: many
- community: active
- input/output: Python (+ command line parameter via gflags)
- search strategies: C++, cannot create own direct via Python
- other features
  + profiling, CPViz
  + supports CP and LP/MIP (GLPK and SCIP)
  + Constraints: graph/network algorithms
- element can be tricky
- reifications must be handled explicit (MIP trickery needed)
- no set vars
Google or-tools/Java

- https://code.google.com/p/or-tools/
- http://www.hakank.org/or-tools/#java
- version: (SVN)

- Java interface to or-tools' C++ engine
- ease of modelling: Java with some extra sugar
- documentation: not much (doc project started Winter 2012)
- examples: many examples (most are mine)
- constraints: many (from the underlying C++ code)
- community: active
- input/output: Java
- search strategies: C++, cannot create own direct via Java
- other features:
  + support CP and LP/MIP (GLPK and SCIP)
  + Profiling, CPViz
  + Constraints: graph/network algorithms, network routing, scheduling
- Element and reifications can be tricky (MIP trickery needed)
- not much syntactic sugar (being Java), more verbose code
- no set vars
Google or-tools/C#

- https://code.google.com/p/or-tools/
- http://www.hakank.org/or-tools/#csharp
- version: (SVN)

- C# interface to or-tools' C++ engine
- ease of modelling: high level, C# with quite much extra sugar
- documentation: not much (doc project started Winter 2012)
- examples: many examples (most are mine)
- constraints: many (from the underlying C++ code)
- community: active
- input/output: C#
- search strategies: C++, cannot create own direct via C#
- other features:
  + great interface to C#, much syntactic sugar
  + supports CP and LP/MIP (GLPK and SCIP)
  + Profiling, CPViz
  + Constraints: graph/network algorithms, network routing, scheduling
- Element and reifications can sometimes be tricky (MIP trickery needed)
- no set vars
## Comparison of the systems, subjective feature matrix

Range: 1..5 where 5 is very good, 1 is not good (or N/A).

<table>
<thead>
<tr>
<th>Feature</th>
<th>MiniZinc</th>
<th>Comet</th>
<th>Choco</th>
<th>JaCoP</th>
<th>Gecode</th>
<th>Gecode/R</th>
</tr>
</thead>
<tbody>
<tr>
<td># my models</td>
<td>890</td>
<td>166</td>
<td>20</td>
<td>18</td>
<td>162</td>
<td>29</td>
</tr>
<tr>
<td>ease of modelling</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>documentation, site</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>num. examples</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>num. constraints</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>active community</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>input/output</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>command line option</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>reification</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>propagators etc</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Element</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>set var</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>float var</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>debugging</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>CVS/SVN</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>open source</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>FlatZinc</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>host language</td>
<td>-</td>
<td>-</td>
<td>Java</td>
<td>Java</td>
<td>C++</td>
<td>Ruby</td>
</tr>
</tbody>
</table>

Note: These are features (and my subjected grades) for ease of learning/modeling.
## Comparison of the systems, subjective feature matrix

Range: 1..5 where 5 is very good, 1 is not good (or N/A).

<table>
<thead>
<tr>
<th>Feature</th>
<th>ECLiPSe</th>
<th>Tailor/Essence'</th>
<th>JaCoP/Scala</th>
<th>SICStus</th>
<th>Zinc</th>
<th>AnsSet</th>
<th>Prog</th>
</tr>
</thead>
<tbody>
<tr>
<td># my models</td>
<td>176</td>
<td>29</td>
<td>38</td>
<td>151</td>
<td>39</td>
<td>84</td>
<td></td>
</tr>
<tr>
<td>ease of modelling</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>documentation, site</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>num. examples</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>num. constraints</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>active community</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>input/output</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>command line option</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>reification</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>propagators etc</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Element</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>set var</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>float var</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>debugging</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>CVS/SVN</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>open source</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>FlatZinc</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>-</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>host language</td>
<td>Prolog</td>
<td>-</td>
<td>Scala</td>
<td>Prolog</td>
<td>-</td>
<td>ASP</td>
<td></td>
</tr>
</tbody>
</table>

Note: These are features (and my subjected grades) for ease of learning/modeling.
Comparison of the systems, subjective feature matrix

Range: 1..5 where 5 is very good, 1 is not good (or N/A).

<table>
<thead>
<tr>
<th>Feature</th>
<th>or-tools</th>
<th>Python</th>
<th>Java</th>
<th>C#</th>
</tr>
</thead>
<tbody>
<tr>
<td># my models</td>
<td>203</td>
<td>36</td>
<td>122</td>
<td></td>
</tr>
<tr>
<td>ease of modelling</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>documentation, site</td>
<td>3(forthcom.)</td>
<td>3(forthcom.)</td>
<td>3(forthcoming)</td>
<td></td>
</tr>
<tr>
<td>num. examples</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>num. constraints</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>active community</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>input/output</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>command line option</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>reification</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>propagators etc</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Element</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>set var</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>float var</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>debugging</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>CVS/SVN</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>open source</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>FlatZinc</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>host language</td>
<td>Python</td>
<td>Java</td>
<td>C#</td>
<td></td>
</tr>
</tbody>
</table>

Note: These are features (and my subjected grades) for ease of learning/modeling.
Constraint Programming systems (forthcoming)

Here are some CP systems that has been beta tested but not updated to latest/final version:

- JSR-331 : ~ 36 models
  Java JSR-331 API for Constraint Programming

- Numberjack : ~ 53 models
  Python interface to Mistral (CP solver) and SCIP (MIP solver)

- AIMMS : ~ 28 models
  Tested the CP extension (beta version)

TODO (perhaps):
  - Scampi (Scala CP solver)
  - or-tools/C++
  - gecode-python (Python interface to Gecode)
  - MS Solver Foundation (the C# interface)
  - OPL
  - Mozart/Oz
## MiniZinc solvers (that I use)

<table>
<thead>
<tr>
<th>Approach</th>
<th>Source</th>
<th>1/N/all</th>
<th>Lang. Set</th>
<th>Set vars</th>
<th>Float</th>
<th>Strength Heur.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Avail.</td>
<td>Sols</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- G12/FD</td>
<td>CP</td>
<td>N</td>
<td>N/All</td>
<td>Mercury</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>- G12/LazyFD</td>
<td>Lazy</td>
<td>N</td>
<td>N/All</td>
<td>Mercury</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>- G12/CPX</td>
<td>Lazy</td>
<td>N</td>
<td>1/All</td>
<td>C++</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>- G12/CBC</td>
<td>MIP</td>
<td>N</td>
<td>1</td>
<td>Mercury?</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>- Gecode</td>
<td>CP</td>
<td>Y</td>
<td>N/All</td>
<td>C++</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>- JaCoP</td>
<td>CP</td>
<td>Y</td>
<td>N/All</td>
<td>Java</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>- SICStus</td>
<td>CP</td>
<td>Y</td>
<td>N/All</td>
<td>Prolog</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>- ECLiPSe/ic</td>
<td>CP</td>
<td>Y</td>
<td>N/All</td>
<td>Prolog</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>- ECLiPSe/fd</td>
<td>CP</td>
<td>Y</td>
<td>N/All</td>
<td>Prolog</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>- ECLiPSe/eplex</td>
<td>MIP</td>
<td>Y</td>
<td>1</td>
<td>Prolog</td>
<td>Y?</td>
<td>Y</td>
</tr>
<tr>
<td>- fzn2smt</td>
<td>SMT</td>
<td>N</td>
<td>1</td>
<td>C++</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>- fzntini</td>
<td>SAT</td>
<td>N</td>
<td>1</td>
<td>C++(?)</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>- Chuffed</td>
<td>Lazy</td>
<td>N</td>
<td>N/All</td>
<td>C++</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>- BProlog</td>
<td>CP</td>
<td>Y</td>
<td>1/All</td>
<td>Prolog</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>- (Choco)</td>
<td>CP</td>
<td>Y</td>
<td>(1)</td>
<td>Java</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>- (SCIP)</td>
<td>MIP</td>
<td>Y</td>
<td>1</td>
<td>C++</td>
<td>N</td>
<td>Y</td>
</tr>
</tbody>
</table>

MIP solvers downgraded in strength since they are restricted in what they can solve. SCIP is even more restricted. Choco is a beta version.
Breaking news: My first CP related academic papers

2 accepted papers this spring on the same topic: Cell Design Problem
My part: 5 MiniZinc models + benchmarking different solvers.

* Short conference paper in *IEA/AIE 2012*
  (International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems)

*Ricardo Soto, Hakan Kjellerstrand, Juan Gutiérrez, Alexis López, Broderick Crawford, and Eric Monfroy:*
**Solving Manufacturing Cell Design Problems using Constraint Programming**

* Journal paper in *Expert Systems with Applications*

*Ricardo Soto, Hakan Kjellerstrand, Orlando Durán, Broderick Crawford, Eric Monfroy, Fernando Paredes:*
**Cell formation in group technology using constraint programming and Boolean satisfiability**
Thank you!

- Questions?

Hakan Kjellerstrand
http://www.hakank.org/index_eng.html
Answer Set Programming

- http://potassco.sourceforge.net/
- http://hakank.org/answer_set_programming/
- version:

- not CP, but interesting to compare to CP ("related paradigm")
- have tested the Potassco system most (other lparse/smodels, etc)
- ease of modelling: Very High Level (Prolog like), different modelling approach
- documentation: much (scattered), no modelling book
- examples: many examples (scattered)
- constraints: no global constraints
- community: active
- input/output: data files
- search strategies: N/A
- other features:  
  + supports optimization  
  + some modelling constructs can be stated easy, e.g. transitive closures  
  + experimental merge with CP  
  + there are many different ASP grounders/solvers
- grounding can take very long  
- has to create decompositions for all (global) constraints
My “about 18” Learning Problems

- SEND+MORE=MONEY / N-Queens: running a model, what to expect in output
- Least Diff: minimize the difference ABCDE - FGHIJ (distinct digits)
- Diet: how to interact with integer arrays and variable arrays
- Seseman: generate one or all solutions, handling of matrices
- Coins grid: Tony Hubermann's grid puzzle, minimize distances, MIP
- Simple map colouring: using graph/matrix, optimization
- de Bruijn sequence: a personal favourite, command line options, all solutions
- alldifferent_except_0: (decomposition of a) global constraint, reification
- Furniture Moving: scheduling, cumulative
- Minesweeper: more advanced example, problems from a file
- Quasigroup Completion: alldifferent on rows/columns, matrices
- Survo puzzle: alldifferent, reading instances from file
- Young Tableaux and partitions: combinatorial problem
- Send Most Money in any base: first optimize, then generating all solutions
- xkcd: simple problem, knapsack / set covering
- Crosswords: simple (from Apt etc), strings/chars, non-trivial Element
- Word square: another non-trivial Element, how to read a file (word list)
- Who killed Agatha: logical problem, non-trivial Element, reification

- Nowadays: about 30 models before blogging...
all_different_except_0: decomposition in different CP systems

Implementation of all_different_except_0 in different CP systems:

- proxy for “ease of modelling”
- overloading of operators
- logical operators
- reification

(Not the only way to encode this constraint.)
all_different_except_0: decomposition

**G12 MiniZinc**

forall(i,j in 1..length(x) where i != j) (  
    (x[i] > 0 \ x[j] > 0) -&gt; x[i] != x[j]  
)

**Comet**

int n = x.getSize();
forall(i in 1..n, j in i+1..n) {
    m.post(x[i] > 0 & x[j] > 0  =&gt;  x[i] != x[j]);
}
all_differenct_except_0: decomposition

Choco

```java
for(int i = 0; i < v.length; i++) {
    for(int j = i+1; j < v.length; j++) {
        m.addConstraint(ifThenElse(
            and(
                gt(v[i], 0),
                gt(v[j], 0)
            ),
            neq(v[i], v[j]),
            TRUE
        ));
    }
}
```
all_different_except_0: decomposition

JaCoP

```java
for(int i = 0; i < v.length; i++) {
    for(int j = i+1; j < v.length; j++) {
        m.impose(new IfThen(
            new And(
                new XneqC(v[i], 0),
                new XneqC(v[j], 0)
            ),
            new XneqY(v[i], v[j])
        )
    });
}
all_different_except_0: decomposition

JaCoP/Scala

for (i <- 0 until y.length; j <- 0 until i) {
  val b = new_BoolVar("b")
  b <=> AND((y(i) #\= 0), (y(j) #\= 0));
  b => (y(i) #\= y(j))
}
all_different_except_0: decomposition

Gecode

for(int i = 0; i < x.size(); i++) {
    for(int j = i+1; j < x.size(); j++) {
        rel(space,
            (x[i] != 0 && x[j] != 0) >> (x[i] != x[j]),
            icl);
    }
}
all_different_except_0: decomposition

Gecode/R

```ruby
n = x.length
b1_is_an bool_var_matrix(n,n)
b2_is_an bool_var_matrix(n,n)
b3_is_an bool_var_matrix(n,n)
n.times{|i|
  n.times{|j|
    if i != j then
      x[i].must_not.equal(0, :reify => b1[i,j])
      x[i].must_not.equal(0, :reify => b2[i,j])
      x[i].must_not.equal(x[j], :reify => b3[i,j])
      (b1[i,j] & b2[i,j]).must.imply(b3[i,j])
    else
      b1[i,j].must.true
      b2[i,j].must.true
      b3[i,j].must.true
    end
  }
}
```
all_different_except_0: decomposition

ECLiPSe CLP

\[
\text{alldifferent\_except\_0}(Xs) : - \\
\text{dim}(Xs, [Len]), \\
\text{labeling}(Xs), \\
( \text{for}(I, 1, \text{Len}) \times \text{for}(J, 1, \text{Len}), \text{param}(Xs) \text{ do} \\
\quad ( I \neq J, Xs[I] \neq 0, Xs[J] \neq 0 ) \\
\rightarrow \\
\quad Xs[I] \neq Xs[J] \\
; \\
\quad \text{true} \\
). 
\]
all_different_except_0: decomposition

SICStus Prolog

alldifferent_except_0(Xs) :-
    ( foreach(X,Xs) do indomain(X)),
    ( foreach(XI,Xs), count(I,1,_),
        param(Xs)
        do
            ( foreach(XJ,Xs), count(J,1,_),
                param(I,XI) do
                    I < J, XI #\= 0, XJ #\= 0
                    ->
                    XI #\= XJ
                    ;
                    true
                )
            )
    ).
all_different_except_0: decomposition

Tailor/Essence'

forall i,j : int(1..n) . ( (i != j) => ( ((x[i] != 0) \ (x[j] != 0)) => (x[i] != x[j]) ) )
all_different_except_0: decomposition

G12 Zinc

forall(i, j in index_set(x) where i != j) (  
  (x[i] > 0 \ x[j] > 0) \rightarrow x[i] \neq x[j]  
)
all_different_except_0: decomposition

or-tools/Python

n = len(a)
for i in range(n):
    for j in range(i):
        s.Add((a[i] != 0) * (a[j] != 0) <= (a[i] != a[j]))
all_different_except_0: decomposition

or-tools/Java

```java
int n = a.length;
for(int i = 0; i < n; i++) {
    for(int j = 0; j < i; j++) {
        IntVar bi = s.makeIsDifferentCstVar(a[i], 0);
        IntVar bj = s.makeIsDifferentCstVar(a[j], 0);
        IntVar bij = s.makeIsDifferentCstVar(a[i], a[j]);
        solver.addConstraint(
            s.makeLessOrEqual(
                s.makeProd(bi, bj).var(), bij));
    }
}
```
all_different_except_0: decomposition

or-tools/C#

```csharp
int n = a.Length;
for(int i = 0; i < n; i++) {
    for(int j = 0; j < i; j++) {
        s.Add((a[i] != 0) * (a[j] != 0) <= (a[i] != a[j]));
    }
}
```
all_different_except_0: decomposition

Answer Set Programming

#const n = 6.
#const m = 9.

values(0..m).
ix(1..n).

% unique indices of x, 1..n
1 { x(I, Val) : values(Val) } 1 :- ix(I).

% all_different except 0
%
% If Val > 0 then there must be 0..1 occurrences of Val in x.
{ x(I, Val) : ix(I) } 1 :- values(Val), Val > 0.