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

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

[Answer Set Programming ("related paradigm")
http://hakank.org/asp/ 84 models]

G12 MiniZinc

- http://www.g12.cs.mu.oz.au/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

- http://g12.research.nicta.com.au/zinc_current/index_home_zinc.php
- http://www.hakank.org/minizinc/zinc.html
- 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

- http://sourceforge.net/projects/jacop-solver/
- http://hakank.org/jacop/jacop_scala.html
- 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).

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

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).

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

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).

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

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)

	Approach	Source Avail.	1/N/all Sols	Lang.	Set vars	Float vars	Strength	Heur.
- G12/FD	CP	N	N/All	Mercury	Y	N	3-4	Y
- G12/LazyFD	Lazy	N	N/All	Mercury	Y	N	4-5	Y
- G12/CPX	Lazy	N	1/All	C++	Y	N	3-4	Y
- G12/CBC	MIP	N	1	Mercury?	N	Y	2	?
- Gecode	CP	Y	N/All	C++	Y	N	5	Y
- JaCoP	CP	Y	N/All	Java	Y	N	4-5	Y
- SICStus	CP	Y	N/All	Prolog	N	N	4	Y
- ECLiPSe/ic	CP	Y	N/All	Prolog	Y	Y	3-4	Y
- ECLiPSe/fd	CP	Y	N/All	Prolog	Y	N	3-4	Y
- ECLiPSe/eplex	MIP	Y	1	Prolog	Y?	Y	2	?
- fzn2smt	SMT	N	1	C++	Y	N	4	N
- fzntini	SAT	N	1	C++(?)	Y	N	3-4	N
- Chuffed	Lazy	N	N/All	C++	N	N	5	Y
- BProlog	CP	Y	1/All	Prolog	N	N	2-3	Y
- (Choco)	CP	Y	(1)	Java	Y	N	1-3	Y
- (SCIP)	MIP	Y	1	C++	N	Y	1-2	N

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)  ->  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  =>  x[i] != x[j]);
}
```

all_different_except_0: decomposition

Choco

```
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

```
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

```
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

```
alldifferent_except_0(Xs) :-
        dim(Xs, [Len]),
        labeling(Xs),
        ( for(I, 1, Len) * for(J, 1, Len), param(Xs) do
              ( I \= J, Xs[I] #\= 0, Xs[J] #\= 0 )
        ->
          Xs[I] #\= Xs[J]
        ;
          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) -> x[i] != 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

```
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#

```
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).

% alldifferent 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.
```