choco: an Open Source Java Constraint Programming Library

The choco team*
website: choco.emn.fr
contact: choco@emn.fr

École des Mines de Nantes
LINA CNRS
4 rue Alfred Kastler – BP 20722
F-44307 Nantes Cedex 3, France

Abstract. choco is a java library for constraint satisfaction problems (CSP) and constraint programming (CP). It is built on a event-based propagation mechanism with backtrackable structures. choco is an open-source software, distributed under a BSD licence and hosted by sourceforge.net. choco is mainly developped by people at École des Mines de Nantes (France) and is financially supported by Bouygues SA and Amadeus SA.

1 Introduction

choco originated in 1999 within the OCRE project, a French national initiative for an open constraint solver for both teaching and research involving researchers and practitionners from Nantes (École des Mines), Montpellier (LIRMM), Toulouse (INRA and ONERA) and Bouygues SA. Its first implementation was in CLAIRE [CJL02], Yves Caseau's language that compiled into C++. It has been used since then as a teaching tool and the main constraint programming developping tool in the Constraint Programming research group in Nantes.

In 2003, choco went through its premiere major modification when it has been implemented into the Java programming language. The objective was to ensure a greater portability and to ensure an easier takeover for newcomers. As for this time, choco really started its worldwide expansion.

In 2008, choco is being taken a step further. Thanks to the hiring of a full-time engineer on the solver (financed by École des Mines de Nantes, Bouygues SA and Amadeus SA), choco enters a new period of its development. As the v2 version is shipped on Sep. 10^{th}, 2008, it offers a clear separation between the model and the solving machinery (providing both modelling tools and innovative

* The choco team is composed of people from École des Mines de Nantes (including Narendra Jussien and Charles Prud'homme), Cork Constraint Computation Center (including Hadrien Cambazard), Bouygues e-lab (including Guillaume Rochart) and Amadeus SA (including François Laburthe).

solving tools), a complete refactoring improving its general performance, and a more user-friendly API for both newcomers and experienced CP practionners.

`choco` v2 is an open system distributed as a sourceforge project under a BSD license authorizing all possible usages. It is a glass box (all the sources are provided) for teaching (illustrating and implementing all major concepts of Constraint Programming), research (its open API allows an easy integration of personal state-of-the-art algorithms and concepts within the solver) and problem solving (it is now used in real-life contexts in several companies).

In a few words, `choco` is an efficient yet readable constraint system for research and development ; `choco` is a readable yet efficient constraint system for teaching.

2 `choco`'s general features

`choco` is a problem modeler and a constraint programming solver available as a Java library. Moreover, its architecture allows the plugin of other (non CP based) solvers.

2.1 A Problem Modeler

`choco` is a problem modeler able to manipulate a wide variety of variable types (all considered here as first-class citizens):

- integer variables;
- set variables representing sets of integer values;
- real variables representing variables taking their value in an interval of floats;
- expressions representing a integer- or real-based expression using operators such as *plus*, *mult*, *minus*, *scalar*, *sum*, etc.

`choco` 's modeler accepts over 70 constraints (provided that the called solver will be a CP-based solver):

- all classical arithmetical constraints (or integers or reals): equal, not equal, less or equal, greater or equal, etc.;
- reified constraints *i.e.* boolean operations between (possibly reified) constraints;
- table constraints defining the sets of tuples that (do not) verify the intended relation for a set of variables;
- a large set of useful classical global constraints including the `alldifferent` [Rég94] constraint, the `global cardinality` [Rég96] constraint, the `nvalue` [BHH+05] constraint, the `element` [BC94] constraint, the `cumulative` [AB93] constraint, etc.

Moreover, `choco` provides access to the most recent state-of-the-art implementations of global constraints produced in Nantes, France including the `tree` [BLF08] constraint and the `geost` [BCP+07] constraint.

Finally, the implementation of the `regular` [Pes04] and `cost-regular` [DPR06] constraints provide an automatic[1] access to all global constraints whose checker is either a deterministic finite automaton or a DFA with (array of) counters as described in the Global Constraint Catalog [BCDP07].

2.2 A Constraint Programming solver

`choco` is (as expected) a constraint programming solver. It provides:

- several implementations of the various domain types (*eg.* enumerated, bounded, list-based, ... integer variables);
- several algorithms for constraint propagation (state-of-the-art AC algorithms for table constraints, full and bound `alldifferent`, parameterized `cumulative`, etc.).

`choco` can either be used in satisfaction mode (computing one solution, all solutions or iterating them) or in optimization mode (maximisation and minimisation). Search can be parameterized using a set of predefined variable and value selection heuristics (including impact-based search [Ref04] and domain over weighted degree [BHLS04]). User's parameterization includes designing her own variable and/or value selectors, as well as precising which should be the decision variables and even designing cascading variable/value selectors for different sets of variables.

Finally, when converting the model into a solver-specific problem, `choco` can enter into a *pre-processor* mode that will perform some automatic improvements[2] in the model. `choco` is initially a solver working with intensional constraints and therefore the *pre-processor* attempts to use the intensional constraints available in choco whenever this is possible. It does the following operations:

- choose a level of consistency e.g : `alldifferent` or `boundAlldifferent`; arc-consistency or forward-checking for extensionnal constraints; arc-consistency on complex expressions or a weaker form of consistency resulting from the decomposition of the expression by introducing intermediate variables;
- compute maximal cliques in the constraint graph of binary differences or disjunctions to state the `alldifferent` global constraint or the `disjunctive` global constraint;
- recognize intentional constraints stated as expressions (or predicates) to state the appropriate intentional constraint : distances ($|x - y| < z$), linear equations, min/max constraints. Some constraints stated extensionnaly such as differences or equalities are also recognized;
- simple value symmetry breaking in case of pure coloring problems.

[1] This automatization will be available on the January 2009 release of `choco` .
[2] Those improvements were investigated during the solver competition.

3 choco's design

choco is a Java library that chose to provide a clear separation between modeling and solving. Figure 1 represents the overall architecture of the choco library. There are two separate parts:

- the first part (from the user's point of view) is devoted to expressing the problem. The idea is to manipulate variables and relations to be verified for these variables (constraints) disregarding their potential implementation (either from the variable point of view or the constraint point of view). A complete API is provided to be able to state a problem in a way as user-friendly as possible.
- the second part is devoted to actually solve the problem. In Figure 1, only CP related information is provided. Solving includes specific memory management for tree-based search (as in CP).

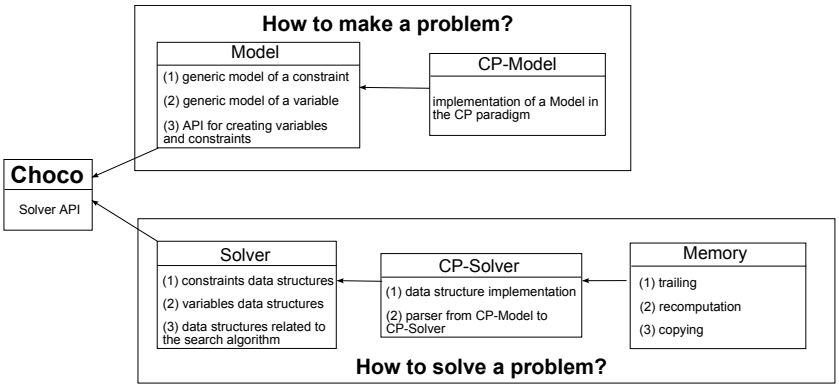

Fig. 1. choco 's general architecture. The separate parts are clearly identified: a modelling part for stating the problem and a solving part (here only the CP related information is described) for actually solving the modelled problem.

This clear separation between model and solver has been introduced to ease the usage of constraint programming to newcommers. This architecture is meant to let newcommers focus on the modeling part of their problem and rely on the *pre-processor* of choco that will take over the user to translate its model into a more CP-like model to be automatically solved by the solver. However, any CP practitionner or CP specialists is left the right to:

- make annotations within the model to force the *pre-processor* to use specific implentations and ways of handling constraints (for example when considering expressions) ;

– use the solver's API to directly program or modify the default behavior of the solver.

choco in its new version has been designed for allowing tree search related solvers to be integrated within the platform. For example, we are currently porting PaLM [JB00], an explanation-based constraint solver which does not rely on a tree-based exploration of the search space. The idea is to provide to the end user of choco a natural and effortless way to use a given constraint model in different contexte (explanation-based constraint programming, local search, etc.)

4 choco in practice

Here is a few lines of code to get the essence of using choco in practice. Notice the use of annotations when building variables (cp:enum) and the explicit separation between Model and Solver.

```
//1- Create the model
Model m = new CPModel();
int n = 6;
//2- declaration of variables
IntegerVariable[] vars = makeIntVarArray("v", n, 0, 5, "cp:enum");
IntegerVariable obj = makeIntVar("obj",0,100,"cp:bound");

//3- add some constraints
String regexp = "(1|2)(3*)(1|4|5)";
m.addConstraint(regular(regexp, vars));
m.addConstraint(neq(vars[0], vars[5]));
m.addConstraint(eq(scalar(new int[]{2,3,1,-2,8,10}, vars), obj));

//4- Create the solver
Solver s = new CPSolver();

//5- read the model and solve it
s.read(m);
s.solve();
if (s.isFeasible()) {
    do {
        for (int i = 0; i < n; i++) {
            System.out.print(s.getVar(vars[i]).getVal());
        }
        System.out.println("");
    } while (s.nextSolution());
}
//6- Print the number of solutions found
```

```
System.out.println("Nb_sol : " + s.getNbSolutions());
```

Which gives the following output :

```
133334 72
133335 82
233331 44
233334 74
233335 84
Nb_sol : 5
```

5 choco as a teaching and research tool

choco is used in many different places for teaching. For example, in France, the universities of Nantes, Montpellier, Rennes, Toulouse, Clermont-Ferrand; the engineering schools of École des Mines de Nancy, École des Mines de Nantes, École Nationale Supérieure des Sciences et Techniques Appliquées, etc. all use choco for teaching constraint programming. choco is not necessarily the only solver that is presented but one of its asset is that it is an open solver whose source code can be browsed and understood easily.

choco is also used in R&D divisions in several companies including Bouygues SA, Amadeus SA but also Dassault Aviation; research agencies such as ONERA and even NASA. It is worth noticing that a company has been created in France which exclusively uses choco as its optimization tool: KLS optim (http://klsoptim.com/).

6 Conclusion

choco is an open, user-oriented constraint solver which provides a clear separation between model and solver. It paves the way to provide a general problem solving library not necessarily dedicated to constraint programming. It is improving every day and eager to integrate user improvements, new constraints, new solvers, propositions, etc.

Visit choco.emn.fr for the latest news, the current version, teaching material, documentation, etc. about choco

Acknowledgments

choco would not exist without its founding fathers: François Laburthe (Amadeus, SA) and Narendra Jussien (École des Mines de Nantes), its core team: Hadrien Cambazard (4C, Cork) and Guillaume Rochart (Bouygues SA), its new generation of developers and contributors: Charles Prud'homme (EMN – project

management), Xavier Lorca (EMN – teaching, training), Guillaume Richaud (EMN – development), Julien Menana (EMN – development), Arnaud Malapert (EMN and University of Montreal – development) and its funding fathers: École des Mines de Nantes which hosts the choco web site, servers, project manager, Bouygues SA and Amadeus SA.

References

[AB93] A. Aggoun and N. Beldiceanu. Extending CHIP in order to solve complex scheduling and placement problems. *Mathl. Comput. Modelling*, 17(7):57–73, 1993.

[BC94] N. Beldiceanu and E. Contejean. Introducing global constraints in CHIP. *Math. Comput. Modelling*, 20(12):97–123, 1994.

[BCDP07] Nicolas Beldiceanu, Mats Carlsson, Sophie Demassey, and Thierry Petit. Global constraint catalogue: Past, present and future. *Constraints*, 12(1):21–62, 2007.

[BCP+07] Nicolas Beldiceanu, Mats Carlsson, Emmanuel Poder, R. Sadek, and Charlotte Truchet. A generic geometrical constraint kernel in space and time for handling polymorphic k-dimensional objects. In *CP*, pages 180–194, 2007.

[BHH+05] Christian Bessiere, Emmanuel Hebrard, Brahim Hnich, Zeynep Kiziltan, and Toby Walsh. Filtering algorithms for the nvalue constraint. In *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR?05), volume 3524 of LNCS*, pages 79–93. Springer-Verlag, 2005.

[BHLS04] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In *European Conference on Artificial Intelligence (ECAI'04)*, pages 146–150, 2004.

[BLF08] N. Beldiceanu, X. Lorca, and P. Flener. Combining tree partitioning, precedence, and incomparability constraints. *Constraints*, 13(4), 2008.

[CJL02] Yves Caseau, François-Xavier Josset, and François Laburthe. Claire: combining sets, search and rules to better express algorithms. *Theory Pract. Log. Program.*, 2(6):769–805, 2002.

[DPR06] Sophie Demassey, Gilles Pesant, and Louis-Martin Rousseau. A cost-regular based hybrid column generation approach. *Constraints*, 11(4):315–333, 2006.

[JB00] Narendra Jussien and Vincent Barichard. The PaLM system: explanation-based constraint programming. In *Proceedings of TRICS: Techniques foR Implementing Constraint programming Systems, a post-conference workshop of CP 2000*, pages 118–133, Singapore, September 2000.

[Pes04] Gilles Pesant. A regular language membership constraint for finite sequences of variables. In Mark Wallace, editor, *CP*, volume 3258 of *Lecture Notes in Computer Science*, pages 482–495. Springer, 2004.

[Ref04] P. Refalo. Impact-based search strategies for constraint programming. In *Principles and Practice of Constraint Programming (CP'04*, pages 556–571, 2004.

[Rég94] J.-C. Régin. A filtering algorithm for constraints of difference in CSP. In *AAAI'94*, pages 362–367, 1994.

[Rég96] J.-C. Régin. Generalized arc consistency for global cardinality constraint. In *AAAI'96*, pages 209–215, 1996.