Teaching Constraint Programming while using PyCSP^3, ACE and Jupyter Notebook

Christophe Lecoutre
Univ. Artois & CNRS, CRIL, rue Jean Souvraz SP 18, 62307 Lens Cedex, France

Abstract

In this paper, we give a brief overview of how Constraint Programming (CP) is taught at the University of Artois (at the Masters level). This is a 36-hour module introducing the fundamental concepts of CP, based, from a practical point of view, on the PyCSP^3 modeling library and the ACE constraint solver. We indicate how the course has evolved over time and how interactive tools such as Jupyter Notebook are well suited to this type of course.

2012 ACM Subject Classification Social and professional topics → Model curricula; Theory of computation → Constraint and logic programming

Keywords and phrases Constraint Programming, Teaching, Modeling Library, Constraint Solver

Digital Object Identifier 10.4230/LIPIcs.WTCP.2023.2

Funding *Christophe Lecoutre*: This work is supported by the National Research Agency under France 2030 bearing the reference ANR-22-EXES-0009 (MAIA Project)

1 Outline of the Course

The context of this course about Constraint Programming (CP) [4, 1, 11, 6] is a master's degree in computer science (at the University of Artois), as part of an AI program. Most of the students pursue in the business environment rather than at the doctoral level.

The course is composed of 9 time slots of 4 hours each (one per week), with eight main lectures:

1. Introduction to CP: main phases (modeling and solving), formalism (concept of constraint networks), and a few illustrations of (successful) CP applications.
2. Modeling: modeling languages and formats, with a focus on PyCSP^3 [9] and XCSP^3 [2], and description of generic constraints as well as half a dozen global constraints through several illustrative case studies.
3. Filtering (part 1): introduction of the classical properties (arc and bound consistency) that are useful to filter the search space (domains), description of a few (simple) filtering algorithms, and presentation of the principle of constraint propagation.
4. Search (part 1): introduction to backtrack search, look-ahead and look-back schemes, and classical search ordering heuristics; quick manipulation of the constraint solver ACE [8].
5. Optimization: presentation of optimization strategies (notably, the ramp-down technique related to Branch and Bound), comparing complete and incomplete approaches, and introducing Large Neighborhood Search (LNS).
6. Filtering (part 2): succinct description of filtering algorithm for table constraints (Simple Tabular Reduction [12, 7] and Compact-Table [5]) and presentation of several local consistencies (singleton arc consistency, path consistency, soft consistencies).
7. Search (part 2): presentation of restarting mechanisms, nogood recording, and various forms of symmetry-breaking.
8. Various topics: timetable and energetic forms of reasoning for the constraint `Cumulative`, advanced data structures for modeling and/or solving like automata and decision diagrams.

For each of these eight main lectures, a two-hour presentation is actually followed by a two-hour exercise session. There is also a complete session (of 4 hours) dedicated to practical modeling (with PyCSP^3), occurring in the fourth week. Finally, students must submit a mini-project such as the Industrial Modeling Challenge presented at CP'15; see Problem 073 at www.csplib.org.

2 Course Evolution

Over time, the course has evolved into a bit more hands-on, especially modeling, as the audience has changed and the attraction for research has diminished. The most salient points regarding the transformation of the course are:

- paying more attention to modeling, with around one third of the time dedicated to that aspect (when including the first session about introducing CP),
- spending more time to dedicated filtering algorithms (propagators) while discarding general algorithms like AC3, AC2001, etc. (which are barely used in modern constraint solvers),
- using interactive tools as Jupyter Notebook, as explained below.

A first tool to more easily capture the attention of students is the Python library PyCSP^3; see pycsp.org. This is because most students (whatever their background) know this programming language, and the interface of PyCSP^3 has been designed to simplify the handling of the modeling process as much as possible. Besides, as two solvers, ACE [8] and Choco [10], are embedded in PyCSP^3, it is possible to directly execute a model (actually, compile the model and run one solver). Interestingly, it becomes easy to write and use Jupyter Notebook documents. To facilitate CP learning, we have written more than 60 Jupyter Notebooks documents:

- one document for each of the 25 popular following constraints (those of of XCSP^3-core [3]), so as to understand by practice the precise semantics of important constraints:
  - `Intension`, `Extension`, `Regular`, `MDD`
  - `AllDifferent`, `AllDifferentMatrix`, `AllEqual`
  - `Increasing`, `Decreasing`, `LexDecreasing`, `LexIncreasing`, `Precedence`
  - `Sum`, `Count`, `NValues`, `Cardinality`
  - `Element`, `ElementMatrix`, `Channel`, `Minimum`, `Maximum`
  - `BinPacking`, `Cumulative`, `Knapsack`, `NoOverlap`
  - `Circuit`
- one document for each of the 34 classical following problems, so as to learn, step by step, how to model them:
  - easy models: AllInterval, BIBD, BoardColoration, CommunityDetection, CryptoPuzzle, FlowShopScheduling, GolombRuler, LabeledDice, MagicSequence, Queens, RectanglePacking, SubgraphIsomorphism, Sudoku, TrafficLights, Warehouse
  - moderately difficult models: BACP, Blackhole, CCMcp, Layout, Mario, Nonogram, Quasigroup, RCPSP, SocialGolfers, SportScheduling, StableMarriage, SteelMillSlab, Vellino
  - difficult models: Amaze, Diagnosis, OpenStacks, PizzaVoucher, RackConfiguration, TravelingTournament
- other documents concern the library interface, the various ways of specifying data and piloting the embedded solvers.

We would like to mention that we are aware that the course does not cover all the fields of CP. We also want to mention that an abbreviated version of the course (7 hours) has been offered (since two years) to engineers in the car industry (as part of a continuing education diploma) and that the return to the tools used in the course has been very positive.

3 Some Practical Details

In the second year of the computer science master's degree at the university of Artois, there is a unit dedicated to "Inference and Constraint Algorithms". This unit is composed of two parts: a CP course, as described above, and a course dedicated to SAT (Boolean satisfiability) and its extensions. These courses have been carried out for about ten years (once a year), but the revisited version for CP (as described in this paper) only for 2 years. For students enrolled in the AI component of the master, the unit is mandatory. Usually, this concerns around 20 students.

The documentation (notably, slides) is made available chapter after chapter (week after week). In the first part of each main lecture, some relaxed moments are made possible by writing (pieces of) models or executing algorithms/solvers live. Students are encouraged to simultaneously write PyCSP^3 models/code by using Google Colab (which avoids installing any kind of software). In the second part of each main lecture, some supervised exercises are given (while students being solicited to finish some exercices at home). An exam is organized at the end of the module, and its evaluation is combined with that of the mini-projet. The topic of the mini-project may change some years.

In general, the concepts are well understood by the students. But as far as modeling is concerned, intensive practice remains a must. From our experience, this is analogous to mastering polymorphism in OOP: students understand the principle but have difficulty putting it into practice at the start. Finally, the problems chosen to illustrate the course are varied: this includes problems of a playful nature (puzzles) as well as problems of a more industrial nature.

References

1. K.R. Apt. *Principles of Constraint Programming*. Cambridge University Press, 2003.
2. F. Boussemart, C. Lecoutre, G. Audemard, and C. Piette. XCSP3: an integrated format for benchmarking combinatorial constrained problems. *CoRR*, abs/1611.03398, 2016. URL: `http://arxiv.org/abs/1611.03398`.
3. F. Boussemart, C. Lecoutre, G. Audemard, and C. Piette. XCSP3-core: A format for representing constraint satisfaction/optimization problems. *CoRR*, abs/2009.00514, 2020. URL: `https://arxiv.org/abs/2009.00514`, `arXiv:2009.00514`.
4. R. Dechter. *Constraint processing*. Morgan Kaufmann, 2003.
5. J. Demeulenaere, R. Hartert, C. Lecoutre, G. Perez, L. Perron, J.-C. Régin, and P. Schaus. Compact-Table: efficiently filtering table constraints with reversible sparse bit-sets. In *Proceedings of CP'16*, pages 207–223, 2016.
6. C. Lecoutre. *Constraint networks: techniques and algorithms*. ISTE/Wiley, 2009.
7. C. Lecoutre. STR2: Optimized simple tabular reduction for table constraints. *Constraints*, 16(4):341–371, 2011.
8. C. Lecoutre. ACE, a generic constraint solver. *CoRR*, abs/2302.05405, 2023. `arXiv:2302.05405`, `doi:10.48550/arXiv.2302.05405`.
9. C. Lecoutre and N. Szczepanski. PyCSP3: modeling combinatorial constrained problems in Python. *CoRR*, abs/2009.00326, 2020. URL: `https://arxiv.org/abs/2009.00326`, `arXiv:2009.00326`.
10. C. Prud'homme and J.-G. Fages. Choco-solver: A java library for constraint programming. *Journal of Open Source Software*, 7(78):4708, 2022. `doi:10.21105/joss.04708`.
11. F. Rossi, P. van Beek, and T. Walsh, editors. *Handbook of Constraint Programming*. Elsevier, 2006.
12. Julian R. Ullmann. Partition search for non-binary constraint satisfaction. *Information Science*, 177:3639–3678, 2007.