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

Christophe Lecoutre

CRIL, Univ. Artois & CNRS, France

WTCP'23
Toronto – August 27, 2023

Context

The context of this course about Constraint Programming (CP):

- University of Artois, France
- Second year of master's degree in computer science
- Unit dedicated to "Inference and Constraint Algorithms", with two parts:
  - a course dedicated to SAT (Boolean satisfiability) and its extension
  - a CP course (this talk)
- Carried out for about ten years (once a year)
- Revisited (mainly from the use of advanced tools) these 2 last years

In our opinion, most of the material of this general CP course is adapted to third-year undergraduate student.

Program of the CP Course (1/2)

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 and XCSP^3, 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.

Program of the CP Course (2/2)

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 and Compact-Table) 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.

And also a complete session (of 4 hours) dedicated to practical modeling (with PyCSP^3), occurring in the fourth week.

Lectures

For each of the 8 main teaching slots, a 2-hour presentation (lecture) is actually followed by a 2-hour exercise session (after a pause of 20').

Concerning lectures:

- the documentation (notably, slides) is made available chapter after chapter (week after week),
- 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):

1. create (or upload) a new notebook on Colab
2. as a first code cell: `pip install pycsp3`
3. as a second code cell: `from pycsp3 import *`
4. you are ready to work!

Directed Works

Importantly, a substantial number of exercises do not solicit the use of computers. On paper sheets and/or black boards:

- write a model for the n-queens problem
- write a filtering algorithm for a specific constraint
- compute the result of running constraint propagation on a small constraint network
- build a search tree
- compute symmetries
- ...

Of course, don't overlook the interest of such exercices.

Practical Works: Developing Pieces of Code

Over years, less practical exercises consisting in writing pieces of code in solvers.

For example, writing a heuristic in ACE looks like:

```
class Ddeg extends HeuristicVariablesDynamic implements TagMaximize {

  public Ddeg(Solver solver, boolean anti) {
    super(solver, anti);
  }

  @Override
  public double scoreOf(Variable x) {
    return x.ddeg();
  }
}
```

Practical Works: Testing Solvers

However, observing the solver behavior remains interesting/informative:

- What about solving a RLFAP instance with default setting?

```
java ace Rlfap-graph-07-opt_c22.xml.lzma
```

- What about not using Solution-Saving?

```
java ace Rlfap-graph-07-opt_c22.xml.lzma -sos=0
```

- What about using the new heuristic *pick/dom*?

```
java ace Rlfap-graph-07-opt_c22.xml.lzma -varh=PickOnDom
```

What can be tested:

- different value ordering heuristics
- different arrays of variables seens as decision variables
- different variable ordering heuristics
- different parameters for restarts
- ...

Important: will be more documented in next version of PyCSP^3/ACE

Practical Works: Mini-Project

At the end of the course, students must submit a mini-project.

As an example, this may be the Industrial Modeling Challenge presented at CP'15; see Problem 073 at www.csplib.org.

What is expected is some work/development about:

- modeling
- experimenting
- developing specific code

Useful solvers:

- solvers in first line: ACE and Choco
- other solvers (recognizing XCSP^3): Picat, Mistral, CoSoCo, , ...
- and possibly more with new forthcoming XCSP^3 parsers written in Rust and Python

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.

PyCSP^3 and Jupyter Notebook

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,
- the interface of PyCSP^3 has been designed to simplify the handling of the modeling process as much as possible,
- the library has become mature ("any" problem can be tackled).

Besides, as two solvers, ACE and Choco, are embedded in PyCSP^3, it is possible to directly execute a model.

Interestingly, it becomes easy to write and use Jupyter Notebook documents.

Learning Constraints

To facilitate CP learning, we have written one Jupyter Notebook document for each of 25 popular constraints, 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`

As an example, let us look at Cardinality.ipynb

Understanding Models Step by Step

To facilitate understanding models, we have written one Jupyter notebook document for each of 34 classical problems:

- 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

As an example, let us look at Warehouse.ipynb

Extending the Scope of the Course?

In our opinion:

1. PyCSP^3 + Jupyter Notebook are well-suited tools for gently introducing CP
2. One can easily extend the existing material with some fancy interfaces written in Python
3. One can write (more interactive) exercice documents with missing parts to be completed by students
4. One can go deeper on certain topics, with complementary libraries:
   - Choco (for example, for dealing with set or graph variables)
   - MiniCP, for exploring the mysteries of code
   - Minizinc or CPMpy, for testing other solving systems/technologies

To Conclude

- We are aware that the course does not cover all the fields of CP
- Since two years, an abbreviated version of the course (7 hours) has been offered 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

Useful links:

- https://www.cril.univ-artois.fr/~lecoutre/#/teaching; on the left, click on Master, and then Constraints
- pycsp.org