Things we underestimated while developing the CPMpy constraint modelling library

Tias Guns and team ✉
DTAI, KU Leuven, Belgium

Abstract

We propose to share our experience of developing CPMpy at ModRef, through an informal presentation that highlights topics we underestimated when developing the library. This will be an informal presentation, aimed at sharing experiences and getting feedback. There is no formal abstract or specific document intended to accompany the talk. We will start working on a journal paper describing the system and design decisions, which can include aspects of the topics that will be discussed. The rest of this 'extended abstract' sketches the topics we would present in the proposed talk.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming

Keywords and phrases Modelling, Reformulation, Constraint Programming, CPMpy

Funding This research received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (Grant No. 101002802, CHAT-Opt).

1 CPMpy

CPMpy is a Constraint Programming and Modeling library in Python, based on numpy, with direct solver access.

It is a top-down effort to make CP technology more accessible to AI researchers and users in general.

Its key features are:

- Easy to integrate with Python machine learning and visualization libraries, because decision variables are NumPy arrays.
- Solver-independent: transparently translating to CP, SMT, ILP, SAT, and BDD solvers
- Incremental solving and direct access to the underlying solvers
- UNSAT Core extraction, hyper-parameter tuning, and visualizations out-of-the-box

2 Architecture

Like all constraint modeling systems (such as Minizinc [2] or Essence [1]), its purpose is to translate a high-level model, e.g. an expression tree over Boolean and Integer decision variables, into a constraint specification that a specific solver accepts as input language.

In CPMpy, we do this through well-defined transformations that rewrite parts of the expression trees. By applying different transformations, we can do only those transformations that are needed to obtain valid input to different solving paradigms.

Figure 1 shows the current state of our transformation setup.

3 Things we underestimated

CPMpy started as a prototype with a position paper at ModRef 19 "Increasing modeling language convenience with a universal n-dimensional array, CPpy as python-embedded example". It is only in 2021, after Prof. Guns obtained his ERC grant on "Conversational

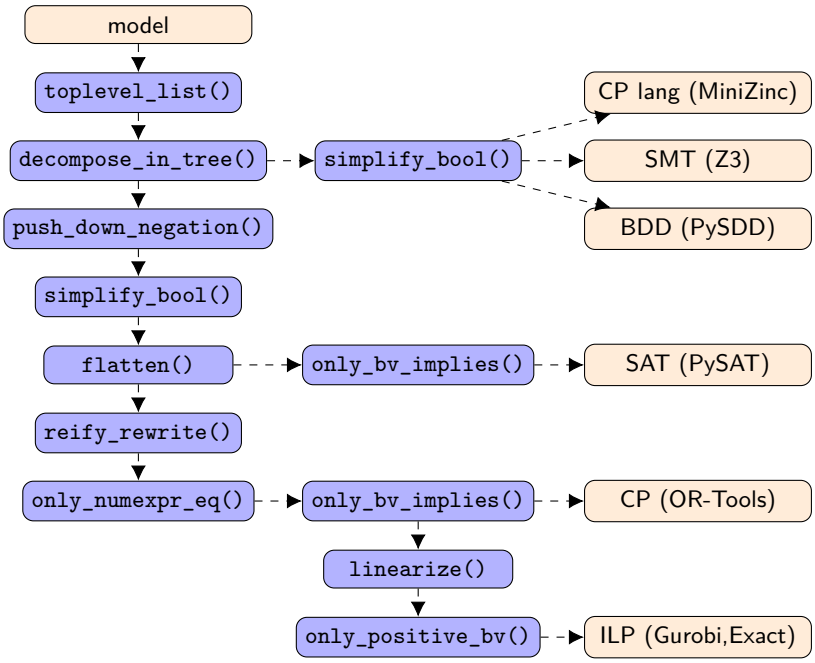

Figure 1 Tranformations workflow of CPMpy

Human-Aware Technology for Optimisation", that we decided to develop it properly and start using it in our research. This was kicked off with a 1-week hackathon with virtual presentations called "Constraint modeling in python"[1]

Hakan Kjellerstrand's early adopter use of our library was a major drive in the development of the usage primitives, the documentation, and the identification of missing features. We now have 5+ researchers working on it, with varying intensity and different expertise and interests.

The things that we underestimated and found/find hardest to get right, which we wish to discuss at the workshop talk, are:

- **Global constraints** are what sets CP solvers apart, and they can be decomposed for other solvers. You think... But in a modeling language, the main building blocks are (possibly nested) expressions. A 'constraint' is simply a Boolean expression at top-level that must be true. All expressions, Boolean or numeric, can be used as arguments of another expression; and we want to support their decomposition without having to flatten the entire model."
- **Negation and reification**, especially of global constraints like 'Circuit', or constraints on partial functions, like 'Element'.
- **Testing** for the correctness and regressions
- **Linearisation** ILP modeling is so similar and yet so different...

Much of this stems from the difference between "getting a typical CP model right" and "getting all expressible CP models right". For all the different solver paradigms...

[1] talks and videos: `https://people.cs.kuleuven.be/~tias.guns/constraints_in_python.html`

References

1 Alan M Frisch, Warwick Harvey, Chris Jefferson, Bernadette Martínez-Hernández, and Ian Miguel. E ssence: A constraint language for specifying combinatorial problems. *Constraints*, 13:268–306, 2008.

2 Nicholas Nethercote, Peter J Stuckey, Ralph Becket, Sebastian Brand, Gregory J Duck, and Guido Tack. Minizinc: Towards a standard cp modelling language. In *International Conference on Principles and Practice of Constraint Programming*, pages 529–543. Springer, 2007.