A CPMpy-based Python library for Constraint Acquisition - PyCoNA

Dimos Tsouros[1], **Tias Guns**[1]

[1]Department of Computer Science, KU Leuven, Belgium
dimos.tsouros@kuleuven.be, senne.berden@kuleuven.be, tias.guns@kuleuven.be

Abstract

Constraint Programming (CP) has been successfully used to model and solve complex combinatorial problems. However, modeling is often not trivial and requires expertise, which is a bottleneck to wider adoption. As a result, the field of Constraint Acquisition (CA) has evolved with the aim to (semi-) automate the modeling process by combining CP and Machine Learning.*Passive* CA acquires constraints using a set of pre-existing examples of solutions and non-solutions, while in (inter)active CA, the system is interacting with the user, e.g., asking whether a (partial) solution satisfies their (unspecified) constraints or not. Despite recent advancements in CA, a key limitation is the lack of a generic, easily accessible tool for practitioners and researchers to utilize these CA systems. In this work, we present PyCoNA, an open-source Python library for CA, which is based on the powerful CPMpy modeling library. PyCoNA currently includes only interactive CA methods, with the intention to also be extended to include passive CA systems. PyCoNA covers the state-of-the-art in interactive CA, including a variety of different algorithms, methods, and types of queries.

Introduction

Constraint Programming (CP) is considered one of the foremost paradigms for solving combinatorial problems in Artificial Intelligence. In CP, the user declaratively states the constraints over a set of decision variables, defining the feasible solutions to their problem, and then a solver is used to solve it. Although CP has many successful applications on combinatorial problems from various domains, the modeling process is not always trivial. This is limiting non-experts from using CP on complex problems and is considered a major bottleneck for its wider adoption (Freuder and O'Sullivan 2014; Freuder 2018).

Motivated by the need to overcome this obstacle, assisting the user in modeling is considered an important direction (Kolb 2016; De Raedt, Passerini, and Teso 2018; Freuder 2018; Lombardi and Milano 2018). In *Constraint Acquisition (CA)*, which is an area where CP meets Machine Learning (ML), the model of a constraint problem is learned from a set of examples (i.e., assignments to the variables) of solutions and non-solutions, or via interaction with a user.

In *passive* CA, a set of pre-existing examples is given to the system, and using these examples, a set of constraints is returned (Bessiere et al. 2004, 2005; Lallouet et al. 2010; Beldiceanu and Simonis 2012; Bessiere et al. 2017; Kumar, Kolb, and Guns 2022; Berden et al. 2022). Many different approaches have been devised, some of them targeting fixed-arity constraints (Bessiere et al. 2005, 2017; Kumar, Kolb, and Guns 2022; Prestwich et al. 2021; Prestwich 2020), while others learn global constraints from a predefined constraint catalog (Beldiceanu and Simonis 2012).

On the other hand, *active* or *interactive* acquisition systems interact with the user to learn a target set of constraints, which represent the problem the user has in mind (Freuder and Wallace 1998; Bessiere et al. 2007, 2017). The most common way of interaction is through the use of (partial) membership queries (is this a solution or not?) (Bessiere et al. 2007, 2013; Lazaar 2021; Tsouros and Stergiou 2020, 2021; Tsouros, Stergiou, and Bessiere 2019, 2020; Tsouros, Stergiou, and Sarigiannidis 2018). To further reduce the number of queries, more expressive types of queries have also been explored (Bessiere et al. 2014; Daoudi et al. 2015, 2016).

Despite the recent advancements in both passive and active CA, an important limitation remains: there is no generic tool for using CA, including the various different methods in the literature. Although research code implementing CA systems from the literature has been made available, such code is focused on accommodating the reproducibility of paper experiments (Bessiere et al. 2023; Kumar, Kolb, and Guns 2022; Tsouros, Berden, and Guns 2023, 2024).

In this short paper, we present PyCoNA, an open-source Python library for CA to alleviate this limitation. The aim is to make CA systems easily accessible to researchers and practitioners. In PyCoNA, we implemented various state-of-the-art interactive CA algorithms and methods from the literature, including the recent advancements with the use of statistical ML components (Tsouros, Berden, and Guns 2024). Passive CA systems are not included yet, which is left for future work.

PyCoNA focuses on ease of use, both for applying CA systems to new problems and for the development and easy integration of new CA methods, offering interfaces for the various CA elements. PyCoNA also provides an implementation of commonly used CA benchmarks to allow the com-

Algorithm 1: Interactive Constraint Acquisition Template

```
Input: $X$, $D$, $B$, $C_{in}$ ($X$: the set of variables, $D$: the set of domains, $B$: the bias, $C_{in}$: an optional set of known constraints)
Output: $C_L$ : the learned constraint network
1: $C_L \leftarrow C_{in}$
2: while True do
3:     $e \leftarrow$ QGEN($C_L$, $B$)
4:     if $e$ = nil then return $C_L$        ▷ converged
5:     if $ASK(e)$ = True then
6:         $B \leftarrow B \setminus \kappa_B(e)$
7:     else
8:         $(B, S) \leftarrow$ FINDSCOPE($e$, $B$)
9:         $(B, C_L) \leftarrow$ FINDC($S$, $C_L$, $B$)
```

parison of the different methods in the literature.

Background

A *constraint satisfaction problem* (*CSP*) is a triple $P = (X, D, C)$, consisting of:

- a set of n variables $X = \{x_1, x_2, ..., x_n\}$, representing the entities of the problem,
- a set of n domains $D = \{D_1, D_2, ..., D_n\}$, where $D_i \subset \mathbb{Z}$ is the finite set of values for x_i,
- a constraint set (also called constraint network) $C = \{c_1, c_2, ..., c_t\}$.

A *constraint* c is a pair ($rel(c)$, $var(c)$), where $var(c) \subseteq X$ is the *scope* of the constraint, and $rel(c)$ is a relation over the domains of the variables in $var(c)$, that (implicitly) specifies which of their value assignments are allowed. $|var(c)|$ is called the *arity* of the constraint. An *example* e_Y is an assignment on a set of variables $Y \subseteq X$. e_Y is rejected by a constraint c iff $var(c) \subseteq Y$ and the projection $e_{var(c)}$ of e_Y on the variables in the scope $var(c)$ of the constraint is not in $rel(c)$. $\kappa_C(e_Y)$ represents the subset of constraints from a constraint set $C[Y]$ that reject e_Y.

Constraint Acquisition

Algorithm 1 presents the generic process followed in interactive CA through partial queries. The main concept is that the CA system asks queries to an oracle, commonly referred to as the user, and through these queries, constraints can be removed from the candidates, or learned. The most common type of query is the classification question $ASK(e_Y)$, with $Y \subseteq X$, asking the oracle if a (partial) assignment e_Y is a (partial) solution to the problem that the user has in mind. If the example posted to the user is classified as positive (line 5), then the candidate constraints in B that violate it are removed (line 6), while if it is classified as negative (line 7) the system tries to find a violated constraint through additional queries (lines 8-9).

Interactive CA systems consist mainly of three components, where (increasingly simpler) queries are generated and posted to the user: (1) Top-level query generation (line 3), (2) Finding the scope(s) of violated constraints, if a negative example is found (line 8), (3) Finding the relations of constraints in the scopes found (line 9).

PyConA: Constraint Acquisition in Python

PyConA is a Python-based, open-source package for CA. Currently, only interactive CA methods are implemented in PyConA; passive CA methods might be added later. It is developed with the following principles:

- We developed PyConA based on the CPMpy[1] modeling library, a powerful solver-independent Python library for CP modeling, which is used for modeling the variables and constraints of the problem.
- As CPMpy provides access to a variety of solvers that can be used during the acquisition process, PyConA itself is also solver-independent.
- PyConA provides interfaces for various crucial elements of the CA process, focusing on ease of use, both for applying CA systems to new problems and for the development and easy integration of new methods.
- We have re-implemented the major state-of-the-art interactive CA algorithms and methods, allowing the use of the system of choice and the comparison of different methods present in the literature.
- PyConA also provides an implementation of commonly used CA benchmarks to allow the comparison of the different methods in the literature and the evaluation of new methods.
- We have based the predictive component on *scikit-learn*, a well-known and widely used machine learning library, and have implemented the constraint-level feature representation from (Tsouros, Berden, and Guns 2024). We also provide interfaces for using any desired predictor and/or custom feature representations.

We now provide a detailed discussion of the main technical components of PyConA.

Problem Instances

In CA, the system needs as input the vocabulary (X, D) of the problem and a language Γ which describes the relations that can appear in the problem. In PyConA, this information, defining the CA task, is encapsulated in the `ProblemInstance` class.

An object of this class represents an instance of the problem the user wants to acquire the constraints for. It must be initialized with the vocabulary of the problem, i.e. its variables with their domains, and a constraint language. An example of creating a `ProblemInstance` in PyConA can be seen in Listing 1. For the vocabulary, we can use any type of variable supported in CPMpy. Then, to create the language, we use PyConA's abstract variables (created with `absvar (shape)`) which can be used to create the language's abstract relations. Using the vocabulary (X, D) and the relations in the constraint language Γ, the system generates the *constraint bias* B, which is the set of all expressions that are candidate constraints for the problem.

[1]https://github.com/CPMpy/cpmpy

Listing 1: Creating Problem Instances

```
import cpmpy as cp
import pycona as ca

# Vocabulary -------

## Define the variables
## lower bound, upper bound, shape, variable
    names
int_vars = cp.intvar(1, 4, shape=(4,4),
            name="var")

# Language -------

## abstract vars from pycona
AV = ca.absvar(2)

## abstract relations using the abstract
    vars
lang = [AV[0] == AV[1],
        AV[0] != AV[1],
        AV[0] < AV[1],
        AV[0] > AV[1],
        AV[0] >= AV[1],
        AV[0] <= AV[1]]

# ProblemInstance -------
instance = ca.ProblemInstance(variables=
    int_vars, language=lang)
```

Listing 2: Using a CA algorithm

```
# Create an interactive CA system
qa = ca.QuAcq()
learned_instance = qa.learn(instance,
                    verbose=1)
# learned_instance.cl stores C_L
```

Interactive Constraint Acquisition

After creating the `ProblemInstance`, a CA system is then used to acquire the constraints of the problem instance at hand. The goal of CA is to learn a constraint set C_L that is equivalent to the (unknown) target constraint set C_T. PyCoNA implements a range of interactive CA algorithms (module .active_algorithms). The core of interactive CA systems is the `AlgorithmCAInteractive` class. This is subclassed with different algorithms from the literature. An example using QuAcq algorithm is shown in Listing 2.

Oracle In interactive CA, the system interacts with an oracle, which can be a human user or a software system (simulated oracle). The CA system asks queries, and based on the answers of the oracle, it either learns or excludes constraints. In PyCoNA, the default oracle is a human user. A simulated constraint-based oracle is also implemented, in the `ConstraintOracle` class, to allow the use of a constraint set as an oracle for experimental procedures. A simple interface is provided for implementing additional oracles, subclassing the `Oracle` abstract class, allowing the use of other software systems in answering the queries, based on the current application. An example using the constraint-based oracle is presented in Listing 3.

Listing 3: Using a constraint-based oracle

```
# Create an oracle
C = ... # a set of constraints
oracle = ca.ConstraintOracle(C)

# Use the oracle during interactive CA
qa = ca.QuAcq()
learned_instance = qa.learn(instance,
                    oracle=oracle, verbose=1)
```

Listing 4: Creating a custom environment

```
# Creating a custom environment
# Use the basic FindScope instead of the
    default FindScope2.
env = ca.ActiveCAEnv(find_scope=
                ca.FindScope())

# initialize the CA system with env
qa_fs1 = ca.QuAcq(env)
```

CA Environment A CA environment is used in each CA system. CA environments are used to offer the core common functionality and carry the necessary data structures of CA systems. The environment for interactive CA systems is the `ActiveCAEnv`, which is created and used by default. Using the environment, the user can also configure the exact settings of the used interactive CA system, choosing the methods used for its subcomponents:

- Query Generation: The query generation system to be used to generate top-level queries.
- FindScope: The FindScope method to be used for finding the scope of violated constraints.
- FindC: The FindC method to be used for finding the exact violated constraints in the given scopes.

Listing 4 shows how to create a custom environment using a different FindScope method than the default one.

Guiding CA systems In each of the different components of interactive CA, queries are posted to the user. Query generation requires solving a CSP with the goal being to find an assignment to the variables, satisfying the learned constraints and violating at least one candidate constraint. In (Tsouros, Berden, and Guns 2023) a method to guide the top-level query generation was proposed, introducing an objective function that uses probabilities obtained from a predictive model. This approach was extended in (Tsouros, Berden, and Guns 2024) to the rest of the interactive CA components, i.e., in FindScope and FindC.

PyCoNA implements this approach, using the prediction-based environment `ProbaActiveCAEnv`, which has 2 additional configurable options:

- Feature representation (.feature_representation): The feature representation used for the constraints
- Classifier (.classifier): The (probabilistic) classifier used to predict probabilities for the candidate constraints

Listing 5: Using the probabilistic environment

```
# Using the probabilistic environment
env = ca.ProbaActiveCAEnv()
ga_predict = ca.GrowAcq(env)
learned_instance = ga_predict.learn(
                        instance, oracle)
```

Listing 6: Getting the performance statistics

```
# Create an interactive CA system
qa = ca.QuAcq()
learned_instance = qa.learn(instance,
                        verbose=1)
# Access the performance statistics
statistics = qa.env.metrics.statistics
short_statistics = qa.env.metrics.
                        short_statistics
```

When this environment is used, the CA system uses the given feature representation to create a dataset of constraints, and then the given probabilistic classifier will be used to provide probabilities for guiding query generation.

Metrics

PyConA offers evaluation metrics to assess the performance of CA systems, allowing experimental evaluation and comparative analysis of implemented systems. The performance statistics are stored in `CASystem.env.metrics`. Listing 6 presents how to access the statistics stored.

Benchmarks

In PyConA, we also provide an implementation of commonly used CA benchmarks to allow the experimental evaluation and comparison of the different methods in the literature or newly developed ones. An example using the nurse rostering benchmark is shown in Listing 7.

Implemented methods

Various approaches and methods from the literature are implemented in PyConA:

- Algorithms:
  - QuAcq (Bessiere et al. 2013, 2023),
  - G-QuAcq (Daoudi et al. 2015).
  - P-QuAcq (Daoudi et al. 2016),

Listing 7: Using PyConA's benchmarks

```
# importing a benchmark
from pycona.benchmarks import
    construct_nurse_rostering
instance, oracle = construct_nurse_rostering
    (shifts_per_day=3, num_days=5,
    num_nurses=8, nurses_per_shift=2)

qa = ca.QuAcq()
learned_instance = qa.learn(instance,
                        oracle, verbose=1)
```

  - MQuAcq (Tsouros, Stergiou, and Sarigiannidis 2018; Tsouros and Stergiou 2020),
  - MQuAcq-2 (Tsouros, Stergiou, and Bessiere 2019),
  - GrowAcq (Tsouros, Berden, and Guns 2023),
- Query Generation:
  - TQ-Gen (Addi et al. 2018),
  - PQ-Gen (Tsouros, Stergiou, and Bessiere 2019).
  - ML-based query generation objectives (Tsouros, Berden, and Guns 2024)
- FindScope:
  - FindScope (Bessiere et al. 2013),
  - Findscope-2 (Tsouros and Stergiou 2020; Bessiere et al. 2023)
- FindC:
  - FindC (Bessiere et al. 2013),
  - FindC-2 (Bessiere et al. 2023)

Conclusion

We introduced PyConA, an open-source python library for constraint acquisition. We believe that this is a significant step forward in making CA tools more accessible for both researchers and practitioners. Our aim is that this will facilitate further research on CA, along with more real-world applications. Thus, the focus of PyConA is on ease of use, integration, and extension, providing interfaces for various crucial elements of the CA process.

In addition, by basing PyConA on the high-level CPMpy modeling library, we provide a solver-independent environment, allowing the users to build their own CSPs and benchmarks for CA. At the same time, we are bridging the gap with ML, allowing users to exploit the integration of powerful ML packages that are available in Python, benefiting at the same time from advancements in such fields.

PyConA a very wide range state-of-the-art methods in interactive CA, allowing the use of the system of choice for applications in new problems. In addition, PyConA provides evaluation metrics to assess the performance of CA systems, as well ass implementation of commonly used CA benchmarks, to allow the experimental evaluation of new methods, and the comparison with existing ones.

Despite these advancements, there are areas for future improvement. Currently, PyConA focuses solely on interactive CA methods, while passive CA methods are planned for future integration. Adding passive methods to the library will provide users with a more complete and practical toolkit. Additionally, this would allow the use of hybrid (passive and active) CA systems in problems where these can complement each other.

Installation PyConA is available in pip: `pip install pycona`

Code The code is open source and available online: https://github.com/cpmpy/pycona

Acknowledgments

This research received funding from the European Research Council (ERC) under the EU Horizon 2020 research and innovation programme (Grant No. 101002802, CHAT-Opt).

References

Addi, H. A.; Bessiere, C.; Ezzahir, R.; and Lazaar, N. 2018. Time-Bounded Query Generator for Constraint Acquisition. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 1–17. Springer.

Beldiceanu, N.; and Simonis, H. 2012. A model seeker: Extracting global constraint models from positive examples. In *Principles and practice of constraint programming*, 141–157. Springer.

Berden, S.; Kumar, M.; Kolb, S.; and Guns, T. 2022. Learning MAX-SAT Models from Examples using Genetic Algorithms and Knowledge Compilation. In *28th International Conference on Principles and Practice of Constraint Programming (CP 2022)*.

Bessiere, C.; Carbonnel, C.; Dries, A.; Hebrard, E.; Katsirelos, G.; Narodytska, N.; Quimper, C.-G.; Stergiou, K.; Tsouros, D. C.; and Walsh, T. 2023. Learning constraints through partial queries. *Artificial Intelligence*, 319: 103896.

Bessiere, C.; Coletta, R.; Daoudi, A.; Lazaar, N.; Mechqrane, Y.; and Bouyakhf, E.-H. 2014. Boosting Constraint Acquisition via Generalization Queries. In *ECAI*, 99–104.

Bessiere, C.; Coletta, R.; Freuder, E. C.; and O'Sullivan, B. 2004. Leveraging the learning power of examples in automated constraint acquisition. In *International Conference on Principles and Practice of Constraint Programming*, 123–137. Springer.

Bessiere, C.; Coletta, R.; Hebrard, E.; Katsirelos, G.; Lazaar, N.; Narodytska, N.; Quimper, C.-G.; Walsh, T.; et al. 2013. Constraint Acquisition via Partial Queries. In *IJCAI*, volume 13, 475–481.

Bessiere, C.; Coletta, R.; Koriche, F.; and O'Sullivan, B. 2005. A SAT-based version space algorithm for acquiring constraint satisfaction problems. In *European Conference on Machine Learning*, 23–34. Springer.

Bessiere, C.; Coletta, R.; O'Sullivan, B.; Paulin, M.; et al. 2007. Query-Driven Constraint Acquisition. In *IJCAI*, volume 7, 50–55.

Bessiere, C.; Koriche, F.; Lazaar, N.; and O'Sullivan, B. 2017. Constraint acquisition. *Artificial Intelligence*, 244: 315–342.

Daoudi, A.; Lazaar, N.; Mechqrane, Y.; Bessiere, C.; and Bouyakhf, E. H. 2015. Detecting types of variables for generalization in constraint acquisition. In *2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)*, 413–420. IEEE.

Daoudi, A.; Mechqrane, Y.; Bessiere, C.; Lazaar, N.; and Bouyakhf, E. H. 2016. Constraint Acquisition Using Recommendation Queries. In *IJCAI: International Joint Conference on Artificial Intelligence*, 720–726.

De Raedt, L.; Passerini, A.; and Teso, S. 2018. Learning constraints from examples. In *Proceedings in Thirty-Second AAAI Conference on Artificial Intelligence*.

Freuder, E. C. 2018. Progress towards the Holy Grail. *Constraints*, 23(2): 158–171.

Freuder, E. C.; and O'Sullivan, B. 2014. Grand challenges for constraint programming. *Constraints*, 19(2): 150–162.

Freuder, E. C.; and Wallace, R. J. 1998. Suggestion strategies for constraint-based matchmaker agents. In *International Conference on Principles and Practice of Constraint Programming*, 192–204. Springer.

Kolb, S. M. 2016. Learning constraints and optimization criteria. In *Workshops at the Thirtieth AAAI Conference on Artificial Intelligence*.

Kumar, M.; Kolb, S.; and Guns, T. 2022. Learning Constraint Programming Models from Data Using Generate-And-Aggregate. In *28th International Conference on Principles and Practice of Constraint Programming (CP 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

Lallouet, A.; Lopez, M.; Martin, L.; and Vrain, C. 2010. On learning constraint problems. In *Tools with Artificial Intelligence (ICTAI), 2010 22nd IEEE International Conference on*, volume 1, 45–52. IEEE.

Lazaar, N. 2021. Parallel Constraint Acquisition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 3860–3867.

Lombardi, M.; and Milano, M. 2018. Boosting Combinatorial Problem Modeling with Machine Learning. *arXiv preprint arXiv:1807.05517*.

Prestwich, S. D. 2020. Robust constraint acquisition by sequential analysis. *Frontiers in Artificial Intelligence and Applications*, 325: 355–362.

Prestwich, S. D.; Freuder, E. C.; O'Sullivan, B.; and Browne, D. 2021. Classifier-based constraint acquisition. *Annals of Mathematics and Artificial Intelligence*, 1–20.

Tsouros, D.; Berden, S.; and Guns, T. 2023. Guided Bottom-Up Interactive Constraint Acquisition. In *International Conference on Principles and Practice of Constraint Programming*.

Tsouros, D. C.; Berden, S.; and Guns, T. 2024. Learning to Learn in Interactive Constraint Acquisition. In Wooldridge, M. J.; Dy, J. G.; and Natarajan, S., eds., *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, February 20-27, 2024, Vancouver, Canada*, 8154–8162. AAAI Press.

Tsouros, D. C.; and Stergiou, K. 2020. Efficient multiple constraint acquisition. *Constraints*, 25(3): 180–225.

Tsouros, D. C.; and Stergiou, K. 2021. Learning Max-CSPs via Active Constraint Acquisition. In *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

Tsouros, D. C.; Stergiou, K.; and Bessiere, C. 2019. Structure-Driven Multiple Constraint Acquisition. In *International Conference on Principles and Practice of Constraint Programming*, 709–725. Springer.

Tsouros, D. C.; Stergiou, K.; and Bessiere, C. 2020. Omissions in Constraint Acquisition. In *International Conference on Principles and Practice of Constraint Programming*, 935–951. Springer.

Tsouros, D. C.; Stergiou, K.; and Sarigiannidis, P. G. 2018. Efficient Methods for Constraint Acquisition. In *24th International Conference on Principles and Practice of Constraint Programming*.