
Antoine Prouvost
Mila, Polytechnique Montréal

Justin Dumouchelle
Polytechnique Montréal

Lara Scavuzzo
Technische Universiteit Delft

Maxime Gasse
Mila, Polytechnique Montréal

Didier Chételat
Polytechnique Montréal

Andrea Lodi
Mila, Polytechnique Montréal

Abstract

We present Ecole, a new library to simplify machine learning research for combinatorial optimization. Ecole exposes several key decision tasks arising in general-purpose combinatorial optimization solvers as control problems over Markov decision processes. Its interface mimics the popular OpenAI Gym library and is both extensible and intuitive to use. We aim at making this library a standardized platform that will lower the bar of entry and accelerate innovation in the field.

Documentation and code can be found at [https://www.ecole.ai](https://www.ecole.ai).

1 Introduction

In many industrial applications, such as day-to-day lot sizing and production planning [33], it is common to repeatedly solve similar NP-hard combinatorial optimization (CO) problems. In practice those are typically fed into an off-the-shelf, general-purpose mathematical solver, which processes each new problem independently and retains no memory of the past. Yet, it is very likely that there exist strong statistical similarities between each of those sequentially solved problems, which could potentially be exploited to solve future problems more efficiently. This observation has motivated two growing lines of machine learning research: 1) pure machine learning (ML) approaches, where CO solvers are entirely replaced by an ML model trained to produce (near)-optimal solutions [6; 11; 29]; and 2) joint approaches, where hand-designed decision criteria within classical CO solvers are replaced by machine learning models trained to optimize a particular metric of the solver [7]. The latter approach is particularly attractive as it allows for exact solving, or at least for mathematical guarantees on the optimality gap (lower and upper bounds), which are often of high value in practice.

Leading general-purpose CO solvers such as Gurobi [20], IBM CPLEX [10], FICO Xpress [3] or SCIP [15] are all based on the branch-and-cut algorithm [1; 18]. This algorithm iteratively divides the feasible space and prunes away sections of that space that cannot contain the optimum using bounds derived from linear programming (LP) relaxations strengthened with cuts [31]. At many points during the algorithm, decisions must be taken that greatly impact the solving performance, traditionally by following a series of hand-crafted rules designed by operations research (OR) experts. Thus, a natural direction to improve the performance of these solvers is to replace the hand-crafted decision rules by machine learning models, trained on representative problems. This promising line of research has already shown improvement on several of these decision tasks, including variable selection [13; 27; 24; 21; 16; 43], node selection [22; 50; 44], cut generation [37], column generation (a.k.a. pricing) [32], primal heuristic selection [28; 23], or formulation selection [8].

All these works have in common that learning can be formulated as a control problem over a Markov decision process (MDP), where a branch-and-cut solver constitutes the environment. Such a formulation opens the door to reinforcement learning (RL) algorithms, which have been successful in
solving extremely complex tasks in other fields. These data-driven policies may hopefully improve upon the expert heuristics currently implemented in commercial solvers, and by doing so highlight new research directions for the combinatorial optimization community.

2 Motivation

Although the idea of using ML for decision-making within CO solvers is receiving increasing attention, research in this area also suffers from several unfortunate technical obstacles, which hinders scientific progress and innovation.

First, reproducibility is currently a major issue. The variety of solvers, problem benchmarks, hand-crafted features, and evaluation metrics used in existing studies impedes reproducibility and comparison to previous works. Those same issues have driven the ML and RL communities to adopt standardized evaluation benchmarks, such as ImageNet or the Arcade Learning Environment. We believe that adopting standard feature sets, problem benchmarks and evaluation metrics for several identified key problems (e.g., branching, node selection, cutting plane generation) will be highly beneficial to this research area as well.

Second, there is a high bar of entry to the field. Modern solvers are complex pieces of software whose implementation always deviates from the vanilla textbook algorithm, and which were not specifically designed for direct customization through machine learning. Implementing a new research idea often requires months of digging in the technical intricacies of low-level C solver code, even for OR experts, and requires ML experts joining the field to be very familiar with the inner working of a CO solver. On the other hand, abstracting away a proper MDP formulation using a solver API is no trivial task either for OR experts, and requires a clear understanding of statistical learning concepts and their significance. We believe that exposing several decision tasks of interest through a unified ML-compatible API will help attract interest from both the traditional ML and OR communities.

Finally, at this time the field hardly benefits from the latest advances in both ML and OR. ML experts typically employ very simplified CO solvers or no solver at all, raising criticism among the OR community, while OR experts typically employ basic ML models and algorithms, thereby missing potential improvements. We believe that a plug-and-play API between a state-of-the-art CO solver and ML algorithms, in the form of a Gym-compatible interface, will allow for closing this gap and let the field benefit from the latest advances from both sides.

3 Proposed solution

To address these practical challenges, we propose a novel open-source library that could serve as a universal platform for research and development in the ML within CO. This new platform, the Extensible Combinatorial Optimization Learning Environments (Ecole) library, is designed as an interface between a CO solver and ML algorithms. It provides a collection of key decision tasks, such as variable selection or cut selection, as partially-observable (PO)-MDP environments in a way that closely mimics OpenAI Gym, a widely popular library among the RL community.

3.1 Design

The design of the library was guided to achieve the following objectives.

Modularity An environment in Ecole is defined by a composition of a task, an observation function and a reward function. For example, in Figure 1, a branching environment is defined with a node bipartite graph observation and the negative number of new nodes created as a reward. Users can define their own observation or reward function to fulfill their specific needs, or even define new environments and simply reuse existing observation and reward functions. These new modules can be defined either directly in C++ for speed, or in Python for flexibility.

Scalability Ecole was designed to add as little overhead as possible on top of the solver. In addition, care was taken to ensure that the library is thread-safe, and in particular Ecole was designed to be free from the Python Global Interpreter Lock (GIL). This allows for straightforward parallelism in Python with multi-threading, which simplifies data collection and policy evaluation in RL algorithms.
import ecole

# set up an MDP environment
env = ecole.environment.Branching()

# use the features from Gasse et al., 2019
observation_function=ecole.observation.NodeBipartite(),
# minimize the B&B tree size
reward_function=-ecole.reward.NNodes()

# set up an instance generator
instances = ecole.instance.CombinatorialAuctionGenerator(n_items=100, n_bids=100)

# generate ten MDP episodes
for _ in range(10):
    # new instances are generated on-the-fly
    instance = next(instances)
    # save instance to disk if desired
    instance.write_problem(f"path/to/problem_{i}.lp")
    # start a new episode
    obs, action_set, reward, done = env.reset(instance)
    # unroll the control loop until the instance is solved
    while not done:
        action = ... # decide on the next action here
        obs, action_set, reward, done, info = env.step(action)

Figure 1: Example code snippet, using Ecole for branching on combinatorial auction problems.

**Speed**  The Ecole core is written in C++, interacts directly with the low-level solver API and provides a thin Python API returning Numpy arrays \cite{38} to interface directly with ML libraries. The initial release of Ecole supports the state-of-the-art open-source solver SCIP \cite{15} as a backend, due to its open code that gives complete access to the solver, and its widespread usage in the literature \cite{22, 21, 16, 14, 42, 19}. We hope in future versions to expand the library to other commercial solvers as well, such as Gurobi \cite{20}, CPLEX \cite{10} or Xpress \cite{3} if the developers of these solvers are interested.

**Flexibility**  Ecole can read instance files in any format understood by SCIP and can therefore be used with existing benchmark collections such as MIPLIB \cite{17}. In addition, the library also provides out-of-the box instance generators for classical CO problems, which can be used to quickly test ideas or to offer standard benchmarks. The generated instances can be saved to disk or passed directly to Ecole environments from memory, as illustrated in Figure 1. Four instance generators are currently implemented (combinatorial auctions, maximum independent set, capacitated facility location and set covering problems) with default parameters chosen to yield solving times on the order of a minute.

**Openness**  To encourage widespread usage of the library, we chose to distribute Ecole under an open-source BSD-3 license \cite{26}. In addition, care was taken that the library could be installed with the popular conda package manager \cite{25}, which is widely used in the ML community.

### 3.2 Supported features

The library currently supports two control tasks. The first is hyperparameter tuning (ecole.environment.Configuring), the task of selecting the best solver hyperparameters before solving. The second is variable selection (ecole.environment.Branching), the task of deciding sequentially on the next variable to branch on during the construction of the branch-and-bound tree. The library also includes an empty “baseline” environment that can be used to benchmark against the solver in its default settings. We are actively working on expanding the library with several other environments that correspond to key research questions in the field, such as node selection and cut selection.

In addition, the library currently supports two observation functions for the state of the solving process. The first is the finite-dimensional variable-aggregated representation from Khalil et al. \cite{27}, and the second is the bipartite graphical representation from Gasse et al. \cite{16}. Finally, the library currently supports two standard reward functions, namely the number of branch-and-bound nodes, and the number of LP iterations added since the last decision. Several standard metrics will be added, such as the primal integral, the dual integral, the primal-dual integral, and the solving time.
Figure 2: Training curve of a branching environment on randomly generated Combinatorial Auction instances. We report the normalized performance on validation instances, the lower the better.

3.3 Example use case

We now demonstrate a typical use case, namely reinforcement learning for variable selection in branch and bound. We used the negative number of nodes created between two decisions as reward function, and the policy has the same architecture as the GNN model of Gasse et al. We pretrained the weights by imitation learning of strong branching, as in the cited paper, and further trained the policy using REINFORCE on samples of transitions encountered during rollouts. Figure 2 shows a 5-10% decrease in the branch-and-bound tree size after 10k episodes. Most interestingly for this article, the environment was defined with only a few lines of code using Ecole, namely a combination of `ecole.environment.Branching`, `ecole.observation.NodeBipartite` and `ecole.reward.NNodes`, and the code was parallelized on 8 threads using the native threading Python library. Equivalent code in PySCIPOpt, the SCIP Python API, would have been substantially more complex to write.

4 Related work

Other open-source libraries have been recently proposed to simplify research at the intersection of machine learning and combinatorial optimization. MIPLearn is a customizable library for machine-learning-based solver configuration currently supporting Gurobi and CPLEX. It offers similar functionalities to the configuration environment in Ecole, which is a (potentially contextual) bandit problem, and can be framed as a borderline case of our MDP framework. In addition, ORGym and OpenGraphGym are Gym-like libraries for learning heuristics for a collection of combinatorial optimization problems that are formulated as sequential decision making problems. Thus, in those libraries there is an explicit MDP formulation like in Ecole, although in those the goal is to replace CO solvers entirely, while Ecole aims at improving existing CO solvers. As such, none of these libraries has the ambitious objective of Ecole, which is to serve as a standardized platform for ML within CO solvers.

5 Conclusions

In this paper, we proposed a new open-source library that offers Gym-like Markov decision process interfaces to key decision tasks in combinatorial optimization solvers. This library was designed to be fast, modular, scalable, and flexible, with easy installation. Such a library is intended to improve reproducibility, lower the bar of entry and simplify integration of recent advances from both fields, in this growing area at the intersection of machine learning and combinatorial optimization.
Acknowledgements

This work was supported by the Canada Excellence Research Chair (CERC) in Data Science for Real-Time Decision Making and IVADO.

References


