Ecole: A Gym-like Library for Machine Learning in Combinatorial Optimization Solvers

Anonymous Author(s)
Affiliation
Address
email

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 this growing field.

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; 28; 2; 4; 21; 16; 43], node selection [22; 36; 34], cut generation [37], column generation (a.k.a. pricing) [32], primal heuristic selection [27; 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 [35; 39]. These data-driven policies may hopefully

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.

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 [12] or the Arcade Learning Environment [5]. 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 [37; 29], raising criticism among the OR community, while OR experts typically employ basic ML models and algorithms [2; 8; 23], 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 [9], 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.

```python
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 [38] to interface directly with ML libraries. The initial release of Ecole supports the state-of-the-art open-source solver SCIP [15] as a backend, due to its open code that gives complete access to the solver, and its widespread usage in the literature [22; 21; 16; 14; 42; 19]. We hope in future versions to expand the library to other commercial solvers as well, such as Gurobi [20], CPLEX [10] or Xpress [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 [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 [26]. In addition, care was taken that the library could be installed with the popular `conda` package manager [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. [28], and the second is the bipartite graphical representation from Gasse et al. [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.

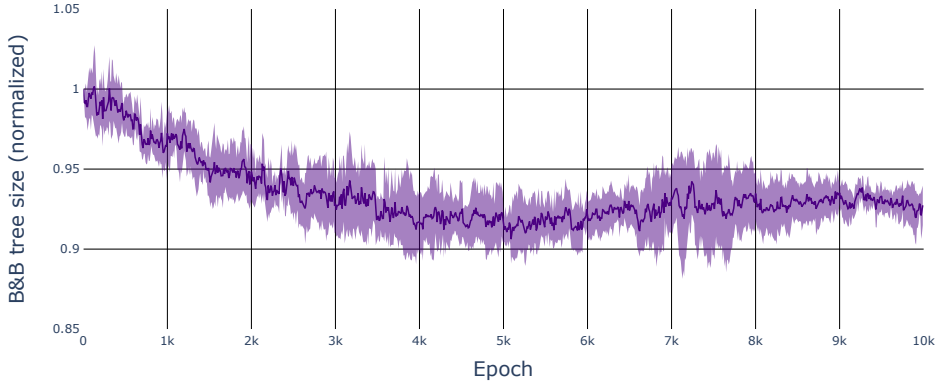

Figure 2: Training curve of a branching environment on randomly generated Combinatorial Auction instances, across 5 seeds. We report the normalized performance on validation instances, the lower the better.

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. [16]. We pretrained the weights by imitation learning of strong branching, as in the cited paper, and further trained the policy using REINFORCE [40] 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 [30], 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 [41] 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 [24] and OpenGraphGym [44] 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.

References

[1] Tobias Achterberg and Roland Wunderling. *Mixed Integer Programming: Analyzing 12 Years of Progress*, pages 449–481. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN 978-3-642-38189-8. doi: 10.1007/978-3-642-38189-8_18.

[2] Alejandro M. Alvarez, Quentin Louveaux, and Louis Wehenkel. A machine learning-based approximation of strong branching. *INFORMS Journal on Computing*, 29:185–195, 2017.

[3] Robert Ashford. Mixed integer programming: A historical perspective with xpress-mp. *Annals of Operations Research*, 149(1):5–17, 2007.

[4] Maria-Florina Balcan, Travis Dick, Tuomas Sandholm, and Ellen Vitercik. Learning to branch. In Jennifer G. Dy and Andreas Krause, editors, *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 353–362. PMLR, 2018.

[5] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The Arcade Learning Environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[6] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. In *Proceedings of the Fifth International Conference on Learning Representations*, 2017.

[7] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 2020.

[8] Pierre Bonami, Andrea Lodi, and Giulia Zarpellon. Learning a classification of mixed-integer quadratic programming problems. In Willem-Jan van Hoeve, editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 595–604, Cham, 2018. Springer International Publishing. ISBN 978-3-319-93031-2.

[9] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[10] IBM CPLEX. CPLEX Optimizer User Manual, 2020. URL `https://www.ibm.com/analytics/cplex-optimizer`.

[11] Hanjun Dai, Elias B. Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[12] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[13] Giovanni Di Liberto, Serdar Kadioglu, Kevin Leo, and Yuri Malitsky. Dash: Dynamic approach for switching heuristics. *European Journal of Operational Research*, 248:943–953, 2016.

[14] Jian-Ya Ding, Chao Zhang, Lei Shen, Shengyin Li, Bing Wang, Yinghui Xu, and Le Song. Optimal solution predictions for mixed integer programs. *arXiv preprint arXiv:1906.09575*, 2019.

[15] Gerald Gamrath, Daniel Anderson, Ksenia Bestuzheva, Wei-Kun Chen, Leon Eifler, Maxime Gasse, Patrick Gemander, Ambros Gleixner, Leona Gottwald, Katrin Halbig, Gregor Hendel, Christopher Hojny, Thorsten Koch, Pierre Le Bodic, Stephen J. Maher, Frederic Matter, Matthias Miltenberger, Erik Mühmer, Benjamin Müller, Marc E. Pfetsch, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Christine Tawfik, Stefan Vigerske, Fabian Wegscheider, Dieter Weninger, and Jakob Witzig. The SCIP Optimization Suite 7.0. ZIB-Report 20-10, Zuse Institute Berlin, March 2020.

[16] Maxime Gasse, Didier Chetelat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 15580–15592. Curran Associates, Inc., 2019.

[17] Ambros Gleixner, Gregor Hendel, Gerald Gamrath, Tobias Achterberg, Michael Bastubbe, Timo Berthold, Philipp Christophel, Kati Jarck, Thorsten Koch, Jeff Linderoth, Marco Lübbecke, Hans D. Mittelmann, Derya Ozyurt, Ted K. Ralphs, Domenico Salvagnin, and Yuji Shinano. MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library. Technical report, Optimization Online, August 2019.

[18] Martin Grötschel. *The sharpest cut: The impact of Manfred Padberg and his work*. SIAM, 2004.

[19] Prateek Gupta, Maxime Gasse, Elias B Khalil, M Pawan Kumar, Andrea Lodi, and Yoshua Bengio. Hybrid models for learning to branch. In *Advances in neural information processing systems*, 2020.

[20] Gurobi Optimization LLC. Gurobi Optimizer Reference Manual, 2020. URL `http://www.gurobi.com`.

[21] Christoph Hansknecht, Imke Joormann, and Sebastian Stiller. Cuts, primal heuristics, and learning to branch for the time-dependent traveling salesman problem. arXiv:1805.01415, 2018.

[22] He He, Hal Daume III, and Jason M Eisner. Learning to search in branch and bound algorithms. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3293–3301. Curran Associates, Inc., 2014.

[23] Gregor Hendel, Matthias Miltenberger, and Jakob Witzig. Adaptive algorithmic behavior for solving mixed integer programs using bandit algorithms. In *Operations Research Proceedings*, pages 513–519. Springer, 2018.

[24] Christian D. Hubbs, Hector D. Perez, Owais Sarwar, Nikolaos V. Sahinidis, Ignacio E. Grossmann, and John M. Wassick. Or-gym: A reinforcement learning library for operations research problems, 2020.

[25] Anaconda Inc. Conda Package Manager, 2020. URL `https://conda.io`.

[26] Open Source Initiative et al. The bsd 3-clause license. *URL: http://opensource. org/licenses/BSD-2-Clause*, 2015.

[27] Elias B. Khalil, Bistra Dilkina, George L. Nemhauser, Shabbir Ahmed, and Yufen Shao. Learning to run heuristics in tree search. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, pages 659–666, 2017.

[28] Elias Boutros Khalil, Pierre Le Bodic, Le Song, George L. Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In Dale Schuurmans and Michael P. Wellman, editors, *AAAI*, pages 724–731. AAAI Press, 2016. ISBN 978-1-57735-760-5.

[29] Wouter Kool, Herke Van Hoof, and Max Welling. Attention, learn to solve routing problems! *International Conference on Learning Representations*, 2019.

[30] Stephen Maher, Matthias Miltenberger, João Pedro Pedroso, Daniel Rehfeldt, Robert Schwarz, and Felipe Serrano. PySCIPOpt: Mathematical programming in python with the SCIP optimization suite. In *Mathematical Software – ICMS 2016*, pages 301–307. Springer International Publishing, 2016. doi: 10.1007/978-3-319-42432-3_37.

[31] John E Mitchell. Branch-and-cut algorithms for combinatorial optimization problems. *Handbook of applied optimization*, 1:65–77, 2002.

[32] Mouad Morabit, Guy Desaulniers, and Andrea Lodi. Machine-learning-based column selection for column generation. Les Cahiers du GERAD G-2020-29, GERAD, HEC Montréal, Canada, 2020.

[33] Yves Pochet and Laurence A Wolsey. *Production planning by mixed integer programming*. Springer Science & Business Media, 2006.

[34] Ashish Sabharwal, Horst Samulowitz, and Chandra Reddy. Guiding combinatorial optimization with uct. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, pages 356–361. Springer, 2012.

[35] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018. ISSN 0036-8075. doi: 10.1126/science.aar6404. URL `https://science.sciencemag.org/content/362/6419/1140`.

[36] Jialin Song, Ravi Lanka, Albert Zhao, Yisong Yue, and Masahiro Ono. Learning to search via retrospective imitation. arXiv:1804.00846, 2018.

[37] Yunhao Tang, Shipra Agrawal, and Yuri Faenza. Reinforcement learning for integer programming: Learning to cut. In *Proceedings of the 37th International Conference on Machine Learning*, pages 1483–1492, 2020.

[38] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22, 2011.

[39] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander Sasha Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Çaglar Gülçehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy P. Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nat.*, 575(7782):350–354, 2019.

[40] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[41] Alinson S Xavier and Feng Qiu. MIPLearn, 2020. URL `https://anl-ceeesa.github.io/MIPLearn`.

[42] Kaan Yilmaz and Neil Yorke-Smith. Learning efficient search approximation in mixed integer branch and bound. *arXiv preprint arXiv:2007.03948*, 2020.

[43] Giulia Zarpellon, Jason Jo, Andrea Lodi, and Yoshua Bengio. Parameterizing branch-and-bound search trees to learn branching policies. *arXiv preprint arXiv:2002.05120*, 2020.

[44] Weijian Zheng, Dali Wang, and Fengguang Song. OpenGraphGym: A parallel reinforcement learning framework for graph optimization problems. In *International Conference on Computational Science*, pages 439–452. Springer, 2020.