Antoine Prouvost, Maxime Gasse, Didier Chételat, Justin Dumouchelle, Andrea Lodi

INFORMS 9th of November 2020

DATA SCIENCE FOR REAL-TIME DECISION-MAKING

Mila

IVADO

POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

About Me

- PhD student at Polytechnique Montréal
- Machine learning and discrete optimization
- Open source coder
- Bike enthusiast

AntoineProuvost
AntoinePrv
www.prouvost.dev

Machine Learning for Combinatorial Optimization

MILP solvers

- Rely on handcrafted heuristics
- Not specialized to the problem at hand
- Do not leverage statistical similarities in repeated solving

Handcrafted heuristics

Handcrafted heuristics are a weak form of learning

- Research
- Human trial and errors
- On MIP lib

Learned heuristics

- Scale to more complex tasks (unlike rule based systems)
- Automated
- Large heuristic space

Bengio, Lodi, Prouvost. "*Machine learning for combinatorial optimization: a methodological tour d'horizon*." European Journal of Operational Research (2020).

Why Ecole?

On Year Ago

Learning to select cutting planes in MILP

Antoine Prouvost, Aleksandr M. Kazachkov, Andrea Lodi

INFORMS Seattle
22nd of October 2019

IVADO
CANADA EXCELLENCE RESEARCH CHAIR DATA SCIENCE FOR REAL-TIME DECISION-MAKING
Mila
POLYTECHNIQUE MONTRÉAL WORLD-CLASS ENGINEERING

Research Code

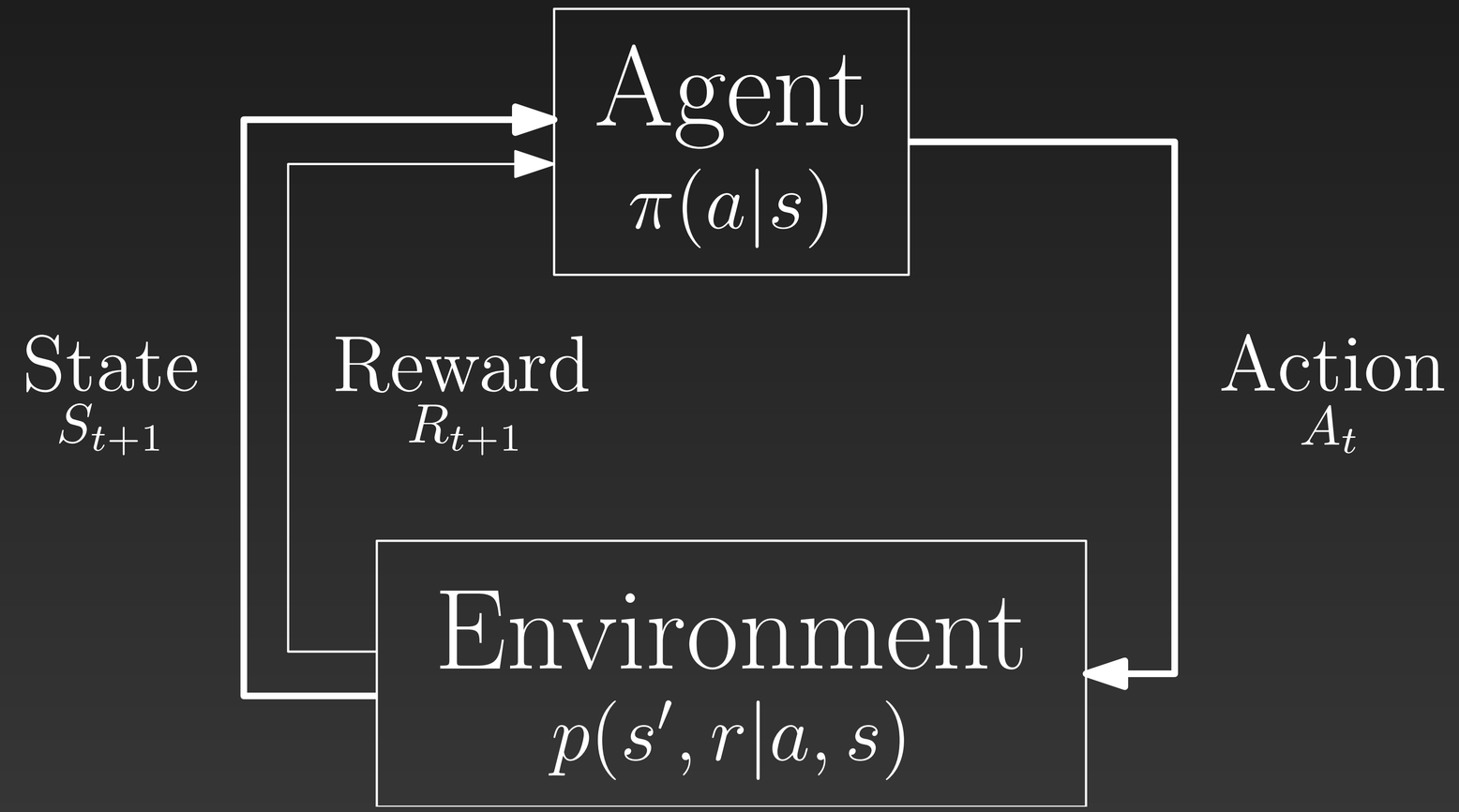
- Not tested
- Not optimized for speed
- Not modular
- Hard to install

Introducing Ecole

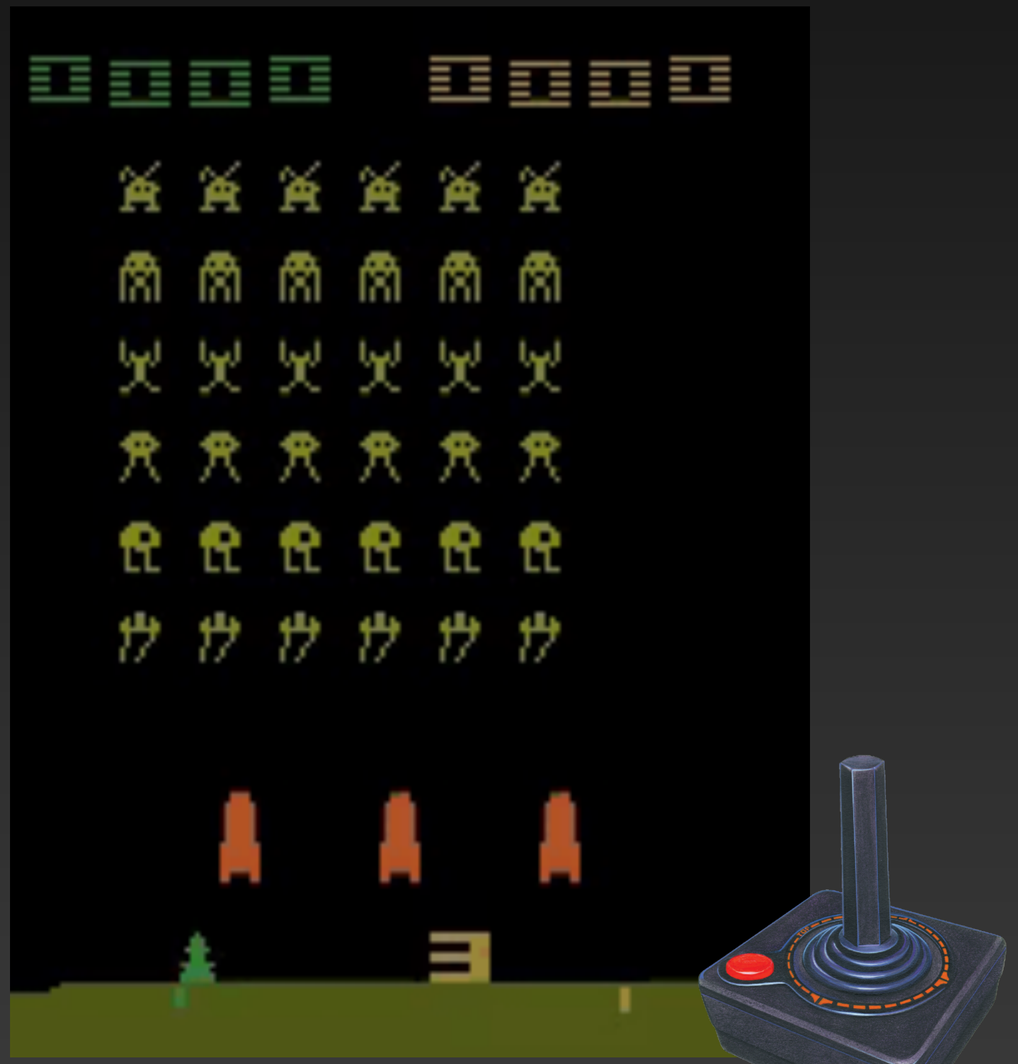
- Fast
- Extensible (through SCIP and PySCIPOpt)
- Well tested
- Good defaults
- Python and C++
- Available through Conda

Learning Environments

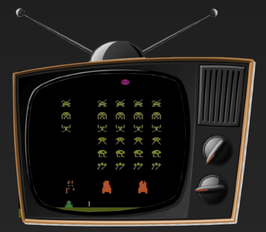
Markov Decision Process

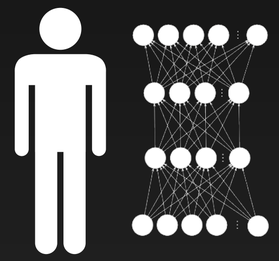


OpenAI Gym



OpenAI Gym





Agent
$\pi(a|s)$

State
S_{t+1}

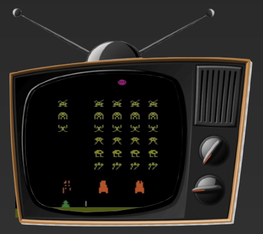
Reward
R_{t+1}

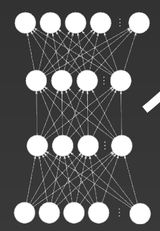
Action
A_t

Environment
$p(s', r|a, s)$

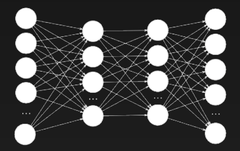
OpenAI Gym

```
import gym


env = gym.make("SpaceInvaders-v0")

for episode in range(1000):
    observation, done = env.reset(), False

    while not done:
        action = policy(observation)
        observation, reward, done, info = env.step(action)
```





Branching with Ecole



Agent
$\pi(a|s)$

Solver statistics

State
S_{t+1}

Reward
R_{t+1}

Action
A_t

Branching variable

Number of nodes

Environment
$p(s', r|a, s)$

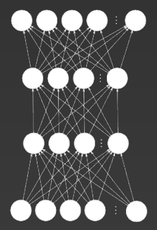
Branching with Ecole

```python
import ecole


env = ecole.environment.Branching()

for episode in range(1000):
    obs, action_set, reward, done = env.reset("path/to/problem")

    while not done:
        action = policy(observation, action_set)
        obs, action_set, reward, done, info = env.step(action)
```

Solved multiple times

Solver statistics

Branching candidates



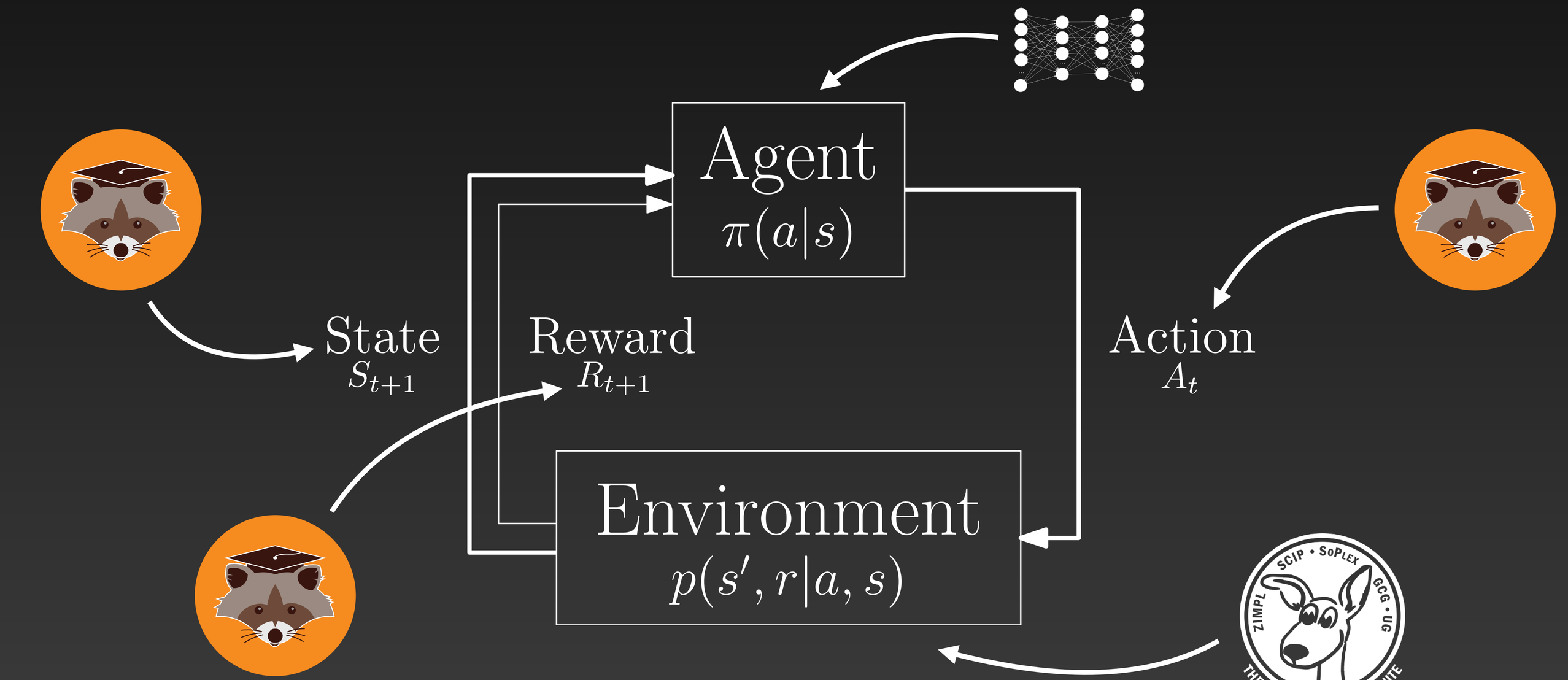
Number of nodes

Instance solved

Branching variable

Extensibility

Highly configurable



Easily swap and define

- Rewards
- States
- Environments

Composition

```
env = ecole.environment.Branching(
    reward_function=(LpIterations()**2 - 3* NNodes()),
    observation_function=NodeBipartite(),
    scip_params={"presolving/maxrestarts": 2},
)
```

New components

```
class NNodeInitLPIterations:

    def before_reset(self, model):
        self.last_iterations = 0.0

    def extract(self, model, done):
        new_iterations = model.as_pyscipopt().getNNodeInitLPIterations()
        diff_iterations = new_iterations - self.last_iterations
        self.last_iterations = new_iterations
        return diff_iterations
```

ds4dm

ds4dm/ecole

www.ecole.ai

doc.ecole.ai