Towards White-Box Benchmarks for Algorithm Control

André Biedenkapp, H. Furkan Bozkurt, Frank Hutter, Marius Lindauer,
University of Freiburg

In a Nutshell

- **Algorithm configuration**, often necessary to achieve peak performance over a set of instances (e.g., AI Planning and SAT)
- It has been shown that different parameter settings are optimal at different stages of an algorithm
- Algorithm control adjusts parameters depending on an observed state
- **Goal**: Learn this control policy from data

Related Work

- **RL for Algorithm Control**:
  - Battiti and Campigotto (2012) applied Least Squares Policy Iteration to learn a policy of one SAT parameter
  - Daniel et al. (2016) used Relative Entropy Policy Search to learn a controller for the step size of NN optimizers.
  - Jaderberg et al. (2017) use Population Based Training to adjust the hyperparameters of RL agents over time

Definition: Algorithm Control

Given:
- a parameterized algorithm $A$
- a configuration space $\Lambda$
- a state description $s(t)$ of algorithm $A$ at each time point $t$
- a space of control policies $\Pi: \mathcal{S} \rightarrow \Lambda$ mapping from states to configurations, and
- a cost metric $c$ assessing the cost of a control policy $\pi$ by running $A$ with $\pi$ (e.g., runtime or accuracy), and
- a set of problems $\mathcal{I}$

**Goal**: obtain a control policy $\pi^* \in \Pi$ with minimal cost $c$. 

RL for Algorithm Control

Control of hyperparameter $\lambda$ at time step $t$ on problem $i$. State $s$ is given by some internal statistics of the Algorithm.

Agents

- $\epsilon$-greedy: Tabular Q-learning with exploration factor 0.1
- URS: Tabular Q-learning with exploration factor 1.0
- SMACv3: Black-box Bayesian optimization (BO)
- DQN: Q-learning with NN function approximator

Insights Gained on White-Box Benchmarks

**Agents**

- $\epsilon$-greedy: Tabular Q-learning with exploration factor 0.1
- URS: Tabular Q-learning with exploration factor 1.0
- SMACv3: Black-box Bayesian optimization (BO)
- DQN: Q-learning with NN function approximator

**Counting**

- BO already performs well for short policies and small action spaces

**Luby**

- RL quickly learns to handle larger action spaces and long policies

**State Features**

- RL not only depends on a good reward function but also good state features
- Plethora of instance-feature we can use as part of the state-space
- What is a good feature describing algorithms?
- What temporal information can we encode?