Learning to optimize computational resources: Frugal training with generalization guarantees

Maria-Florina Balcan, Tuomas Sandholm, Ellen Vitercik
Carnegie Mellon University

Algorithm configuration

Algorithms often have **tunable parameters**
- Impact resource consumption such as runtime, memory usage, ...
- Hand-tuning is time-consuming and tedious

**This paper:** theoretical guarantees for algorithm configuration via ML

Learning-based configuration procedure

Input: Set of “typical” problem instances drawn from distribution \( \Gamma \)
- E.g., integer programs (IPs) an airline solves day to day
Output: Parameter setting with low expected resource consumption
- E.g., low expected runtime, memory usage, ...

**Goal:** Procedure itself should have low resource consumption

Notation and example

\( \ell(p, j) \): Resources required to solve instance \( j \) using params \( p \in \mathbb{R}^d \)

**Example:** \( j \) = integer program and \( p \) = CPLEX parameter setting
\( \ell(p, j) \) = size of branch-and-bound tree CPLEX builds

Prior research

Kleinberg et al. ‘17, ‘19 and Weisz et al., ‘18, ‘19:
- Focus on finite parameter spaces
- Can be used on infinite parameter space:
  - Uniformly sample \( \Omega \left( \frac{\ell}{\epsilon} \right) \) configurations; run algorithm over finite set
  - Output configuration is in top \( \gamma \)-quantile

Bad case for randomly sampling parameters:
\( \mathbb{E}_{\ell \sim \Gamma}[\ell(p, j)] \)

These worst-case examples do exist
- E.g., in integer programming [Balcan, Dick, Sandholm, V. ‘18]

Our contributions

Algorithm that finds **finite** set of good params from within **infinite** set
- Set contains **nearly-optimal** parameter with high probability
- Can be input to algorithm for finite parameter spaces
[Kleinberg et al. ‘17, ‘19; Weisz et al., ‘18, ‘19]

Useful (and requisite) structure

We often observe the following structure
- E.g., in integer programming [Balcan, Dick, Sandholm, V. ‘18]

\( \ell(p, j) \) is piecewise-constant

Our algorithm

\[ \text{OPT} = \min_p \mathbb{E}_{\ell \sim \Gamma}[\ell(p, j)] \]

(Actually compete with nuanced notion of OPT, like prior research [Kleinberg et al. ‘17, ‘19; Weisz et al., ‘18, ‘19])

Maintains upper confidence bound (UCB) on OPT, initially set to \( \infty \)

On each round \( t \), draws set \( S_t \) from \( \Gamma \)

Computes partition of parameters into regions where within each:
- For each instance in \( S_t \), the loss \( \ell \), capped at \( 2^t \), is constant
- Implementation guidance in prior research [e.g., Balcan, Dick, Sandholm, V. ‘18]

On each region of partition, if average capped loss sufficiently low:
- Chooses arbitrary parameter from region and deems it “good”

Once cap \( 2^t \) has grown sufficiently large compared to UCB on OPT:
- Algorithm returns set of “good” parameters

Guarantees

**Theorem** (informal):
1. Whp, exists “good” param that’s within \( 1 + \epsilon \) of optimal
2. Algorithm terminates after \( \tilde{O}(\ln(\sqrt{T + \epsilon} \cdot \text{OPT})) \) rounds
3. On final round, let \( P \) be the size of partition algorithm computes.
   - Number of “good” parameters is \( \tilde{O}(\ln(\sqrt{T + \epsilon} \cdot \text{OPT})) \)
4. \( |S_t| \) is polynomial in \( 2^t \) (linear in OPT), \( \ln P \), \( d \), and \( \frac{1}{\epsilon} \)

In bad case for random sampling, algorithm terminates in \( \tilde{O}(1) \) rounds