Generalization in Portfolio-Based Algorithm Selection

Maria-Florina Balcan,[1] **Tuomas Sandholm,**[1,2,3,4] **Ellen Vitercik**[1]
[1]Computer Science Department, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213
[2]Optimized Markets, Inc., Pittsburgh, PA 15213, USA
[3]Strategic Machine, Inc., Pittsburgh, PA 15213, USA
[4]Strategy Robot, Inc., Pittsburgh, PA 15213, USA
{ninamf, sandholm, vitercik}@cs.cmu.edu

Abstract

Portfolio-based algorithm selection has seen tremendous practical success over the past two decades. This algorithm configuration procedure works by first selecting a portfolio of diverse algorithm parameter settings, and then, on a given problem instance, using an *algorithm selector* to choose a parameter setting from the portfolio with strong predicted performance. Oftentimes, both the portfolio and the algorithm selector are chosen using a *training set* of typical problem instances from the application domain at hand. In this paper, we provide the first provable guarantees for portfolio-based algorithm selection. We analyze how large the training set should be to ensure that the resulting algorithm selector's average performance over the training set is close to its future (expected) performance. This involves analyzing three key reasons why these two quantities may diverge: 1) the learning-theoretic complexity of the algorithm selector, 2) the size of the portfolio, and 3) the learning-theoretic complexity of the algorithm's performance as a function of its parameters. We introduce an end-to-end learning-theoretic analysis of the portfolio construction and algorithm selection together. We prove that if the portfolio is large, overfitting is inevitable, even with an extremely simple algorithm selector. With experiments, we illustrate a tradeoff exposed by our theoretical analysis: as we increase the portfolio size, we can hope to include a well-suited parameter setting for every possible problem instance, but it becomes impossible to avoid overfitting.

1 Introduction

Algorithms often come with a variety of tunable parameters. With a deft parameter tuning, these algorithms can often efficiently solve computationally challenging problems. However, the best parameter setting for one problem is rarely optimal for another. *Algorithm portfolios*, which are finite sets of parameter settings, are used in practice to deal with this variability. A portfolio is often used in conjunction with an *algorithm selector*, which is a function that determines which configuration in the portfolio to employ on any input problem instance. Portfolio-based algorithm selection has fueled breakthroughs in combinatorial auction winner determination (Leyton-Brown 2003; Sandholm 2013), SAT (Xu et al. 2008), integer programming (Xu, Hoos, and Leyton-Brown 2010; Kadioglu et al. 2010), planning (Núñez, Borrajo, and López 2015), and many other domains.

Both the portfolio and the algorithm selector are often chosen using a *training set* of problem instances from the application domain at hand. This training set is typically assumed to be drawn from an unknown, application-specific distribution. The portfolio and algorithm selector are chosen to have strong average performance (quantified by low average runtime, for example) over the training set. We investigate whether the learned algorithm selector also has strong expected performance on problems from the same application domain. The difference between average performance and expected performance is known as *generalization error*. If the generalization error is small, every parameter setting's average performance over the training set is close to its expected performance, so the learned algorithm selector will not *overfit*. When overfitting occurs, the learned selector has strong average performance over the training set but poor expected performance on the true distribution.

There are multiple reasons the generalization error might be large in this setting: 1) the learning-theoretic complexity of the algorithm selector, 2) the size of the portfolio, and 3) the learning-theoretic complexity of the algorithm's performance as a function of its parameters. We provide end-to-end bounds in terms of all three elements simultaneously. The multitude of factors impacting generalization differentiates this paper from prior research on generalization in algorithm configuration (e.g., Gupta and Roughgarden 2017; Balcan et al. 2017; Garg and Kalai 2018; Liu et al. 2020). That research focuses on learning a *single* good configuration rather than a portfolio together with an algorithm selector. In the former case, generalization error only grows with (3)—just one of the sources of error we must contend with.

Our bounds apply to the widely-applicable setting where on any fixed input, algorithmic performance is a piecewise-constant function of its parameters with at most t pieces, for some $t \in \mathbb{Z}$. This structure has been observed in algorithm configuration for integer programming, greedy algorithms, clustering, and computational biology (Gupta and Roughgarden 2017; Balcan et al. 2017, 2018, 2021; Balcan 2020). Given a training set of size N, we prove that the generalization error is bounded[1] by $\tilde{O}\left(\sqrt{\left(\bar{d} + \kappa \log t\right)/N}\right)$, where

[1]Here we assume that algorithmic performance is a quantity in

κ is the size of the portfolio and $\bar{d}$ measures the *intrinsic complexity* of the algorithm selector, as we define in Section 3. We also prove that this bound is tight up to logarithmic factors: the generalization error can be as large as $\tilde{\Omega}\left(\sqrt{\left(\bar{d}+\kappa\right)/N}\right)$. This implies that even if the algorithm selector is extremely simple ($\bar{d}$ is small), overfitting cannot be avoided in the worst case when the portfolio size κ is large. Moreover, we instantiate our guarantees for several commonly-used families of algorithm selectors.

Finally, via experiments in the context of integer programming configuration, we illustrate the inherent tradeoff our theory exposes: as we increase the portfolio size, we can hope to include a high-performing parameter setting for any given instance, but it become increasingly difficult to avoid overfitting. We incrementally increase the size of the portfolio and with each addition we train an algorithm selector using regression forest performance models. As the portfolio size increases, the algorithm selector's training performance continues to improve, but there comes a point where the test performance begins to worsen, meaning that the algorithm selector is overfitting to the training set.

Additional related research. Gupta and Roughgarden (2017) also provide generalization guarantees for algorithm configuration. They primarily analyze the problem of learning a single parameter setting with high expected performance. They do provide guarantees for the more general problem of learning a mapping from instances to parameter settings in a few special cases, but do not study the problem of learning a portfolio in conjunction with learning a selector, which we do. They study settings where for each problem instance, a domain expert has defined a number of relevant features, as do we in Section 4. Their first result applies to learning an algorithm selector when the set of features is finite. In contrast, our results apply to infinite feature spaces. Their second set of results is tailored to the problem of learning empirical performance models. An empirical performance model is meant to predict how long a particular algorithm will take to run on a given input. An algorithm selector can use an empirical performance model by selecting the parameter setting with best predicted performance. Gupta and Roughgarden (2017) provide guarantees that bound the difference between the empirical performance model's expected error and average error over the training set. Their guarantees can be applied once the portfolio is already chosen. They do not study the problem of learning the portfolio itself, whereas we study the composite problem of learning the portfolio and the algorithm selector.

2 Problem Formulation and Road Map

Notation. Our theoretical guarantees apply to algorithms parameterized by a real value $\rho \in \mathbb{R}$. We use the notation $\mathcal{Z}$ to denote the set of problem instances the algorithm may take as input. For example, $\mathcal{Z}$ might consist of integer programs (IPs) if we are configuring an IP solver. There is an unknown distribution $\mathcal{D}$ over problem instances in $\mathcal{Z}$.

[0, 1], an assumption we relax in Section 2.

To describe the performance of a parameterized algorithm, we adopt the notation of prior research (Balcan et al. 2021). For every parameter setting $\rho \in \mathbb{R}$, there is a function $u_\rho : \mathcal{Z} \to [0, H]$ that measures, abstractly, the performance of the algorithm parameterized by ρ given an input $z \in \mathcal{Z}$. For example, u_ρ might measure runtime or the quality of the algorithm's output. We use the notation $\mathcal{U} = \{u_\rho : \rho \in \mathbb{R}\}$ to denote the set of all performance functions.

Problem formulation. A portfolio-based algorithm selection procedure relies on two key components: a *portfolio* and an *algorithm selector*. A portfolio is a set $\mathcal{P} = \{\rho_1, \ldots, \rho_\kappa\} \subseteq \mathbb{R}$ of κ parameter settings. An algorithm selector is a mapping $f : \mathcal{Z} \to \mathcal{P}$ from problem instances $z \in \mathcal{Z}$ to parameter settings $f(z) \in \mathcal{P}$. In practice (Xu, Hoos, and Leyton-Brown 2010; Kadioglu et al. 2010; Sandholm 2013), the portfolio and algorithm selector are typically learned using the following high-level procedure:

1. Choose a class $\mathcal{F}$ of algorithm selectors, each of which maps $\mathcal{Z}$ to $\mathbb{R}$. (In Section 4, we provide several examples of classes $\mathcal{F}$ used in practice.)
2. Draw a training set $\mathcal{S} = \{z_1, \ldots, z_N\} \sim \mathcal{D}^N$ of problem instances from the unknown distribution $\mathcal{D}$.
3. Use $\mathcal{S}$ to learn a portfolio $\hat{\mathcal{P}} = \{\rho_1, \ldots, \rho_\kappa\} \subseteq \mathbb{R}$.
4. Use $\mathcal{S}$ to learn an algorithm selector $\hat{f} \in \mathcal{F}$ that maps to parameter settings in the portfolio $\hat{\mathcal{P}}$.

Given an instance $z \in \mathcal{Z}$, the performance of the parameter setting selected by $\hat{f}$ is $u_{\hat{f}(z)}(z)$. We bound the expected quality $\mathbb{E}_{z\sim\mathcal{D}}\left[u_{\hat{f}(z)}(z)\right]$ of the learned algorithm selector.

Road map. We first analyze the extent to which the average performance of the selector $\hat{f}$ over the training set generalizes to its expected performance. We then use this analysis to relate the performance of $\hat{f}$ and the optimal selector under the optimal choice of a portfolio. Specifically, we bound the difference between $\mathbb{E}_{z\sim\mathcal{D}}\left[u_{\hat{f}(z)}(z)\right]$ and $\max_{\mathcal{P}:|\mathcal{P}|\leq\kappa} \mathbb{E}_{z\sim\mathcal{D}}\left[\max_{\rho\in\mathcal{P}} u_\rho(z)\right]$. (If our goal is to minimize $u_\rho(z)$, we may replace each $\max$ with a $\min$.)

3 Sample Complexity Bounds

In this section, we bound the difference between the average performance of any selector $f \in \mathcal{F}$ over the training set $\mathcal{S} \sim \mathcal{D}^N$ and its expected performance. Formally, we bound

$$\left|\frac{1}{N}\sum_{z\in\mathcal{S}} u_{f(z)}(z) - \mathop{\mathbb{E}}_{z\sim\mathcal{D}}\left[u_{f(z)}(z)\right]\right| \tag{1}$$

for any choice of an algorithm selector $f \in \mathcal{F}$. This will serve as a building block for our general analysis of portfolio-based algorithm selection.

Our bounds apply in the widely-applicable setting where on any fixed input, algorithmic performance is a piecewise-constant function of the algorithm's parameters. This structure has been observed in algorithm configuration for integer

programming, greedy algorithms, clustering, and computational biology (Gupta and Roughgarden 2017; Balcan et al. 2017, 2018, 2021; Balcan 2020). To describe this structure more formally, for a fixed input $z \in \mathcal{Z}$, we use the notation $u_z^* : \mathbb{R} \to \mathbb{R}$ to denote algorithmic performance as a function of the parameters (whereas the functions u_ρ defined in Section 2 measure performance as a function of the input z). Naturally, $u_z^*(\rho) = u_\rho(z)$. We refer to u_z^* as a *dual function* (as opposed to u_ρ, which is a *primal function*). We assume algorithmic performance is a piecewise-constant function of the parameters, or more formally, that each function u_z^* is piecewise constant with at most t pieces, for some $t \in \mathbb{Z}$.

Our bounds depend on both the number of pieces t and on the *intrinsic complexity* of the class of algorithm selectors $\mathcal{F}$. We use the following notion of the *multi-class projection* of $\mathcal{F}$ to define the class's intrinsic complexity.

Definition 3.1. Given a selector $f \in \mathcal{F}$, let $\rho_1 < \rho_2 < \cdots < \rho_{\bar{\kappa}}$ be the parameter settings f maps to, with $\bar{\kappa} \leq \kappa$. The function f defines a partition $Z_1, \ldots, Z_{\bar{\kappa}}$ of the problem instances $\mathcal{Z}$ where for any $z \in \mathcal{Z}$, if $f(z) = \rho_i$, then $z \in Z_i$. For each function $f \in \mathcal{F}$ there is therefore a corresponding multi-class function $\bar{f} : \mathcal{Z} \to [\kappa]$ that indicates which set of the partition the instance z belongs to: $\bar{f}(z) = i$ when $z \in Z_i$. We use the notation $\bar{\mathcal{F}} = \left\{\bar{f} : f \in \mathcal{F}\right\}$ to denote the set of all such multi-class functions.

Defining this set of multi-class functions allows us to use classic tools from multi-class learning to reason about the algorithm selectors $\mathcal{F}$. In particular, our bounds depend on the *Natarajan (1989) dimension* of the class $\bar{\mathcal{F}}$, which is a natural extension of the classic VC dimension (Vapnik and Chervonenkis 1971) to multi-class functions.

Definition 3.2 (Natarajan dimension)**.** The set $\bar{\mathcal{F}}$ *multi-class shatters* a set of problem instances $z_1, \ldots, z_N$ if there exist labels $y_1, \ldots, y_N \in [\kappa]$ and $y'_1, \ldots, y'_N \in [\kappa]$ such that:

1. For every $i \in [N]$, $y_i \neq y'_i$, and
2. For any subset $C \subseteq [N]$, there exists a function $\bar{f} \in \bar{\mathcal{F}}$ such that $\bar{f}(z_i) = y_i$ if $i \in C$ and $\bar{f}(z_i) = y'_i$ otherwise.

The *Natarajan dimension* of $\bar{\mathcal{F}}$ is the cardinality of the largest set that can be multi-class shattered by $\bar{\mathcal{F}}$.

In Section 4, we bound the Natarajan dimension of $\bar{\mathcal{F}}$ for several commonly-used classes of algorithm selectors $\mathcal{F}$. We use Natarajan dimension to quantify the intrinsic complexity of the class of selectors, which in turn allows us to bound Equation (1) for every function $f \in \mathcal{F}$. To do so, we relate the Natarajan dimension of $\bar{\mathcal{F}}$ to the *pseudo-dimension* of the function class $\mathcal{U}_\mathcal{F} = \left\{z \mapsto u_{f(z)}(z) : f \in \mathcal{F}\right\}$. Every function in $\mathcal{U}_\mathcal{F}$ is defined by an algorithm selector $f \in \mathcal{F}$. On input $z \in \mathcal{Z}$, $u_{f(z)}(z)$ equals the utility of the algorithm parameterized by $f(z)$ on input z. Pseudo-dimension (Haussler 1992) is a classic learning-theoretic tool for measuring the intrinsic complexity of a class of real-valued functions (whereas Natarajan dimension applies to multi-class functions). Both Natarjan dimension and pseudo-dimension are extensions of the classic VC dimension, so they bear some resemblance. Below, we define the pseudo-dimension of the class $\mathcal{U}_\mathcal{F}$.

Definition 3.3 (Pseudo-dimension)**.** The set $\mathcal{U}_\mathcal{F}$ *shatters* a set of instances $z_1, \ldots, z_N \in \mathcal{Z}$ if there exist *witnesses* $w_1, \ldots, w_N \in \mathbb{R}$ such that for any subset $C \subseteq [N]$, there exists an algorithm selector $f \in \mathcal{F}$ such that $u_{f(z_i)}(z_i) \leq w_i$ if $i \in C$ and $u_{f(z_i)}(z_i) > w_i$ otherwise. The *pseudo-dimension* of $\mathcal{U}_\mathcal{F}$, denoted $\text{Pdim}\left(\mathcal{U}_\mathcal{F}\right)$, is the size of the largest set of instances that can be shattered by $\mathcal{U}_\mathcal{F}$.

Classic learning-theoretic results allow us to provide generalization bounds once we calculate the pseudo-dimension. For example (Haussler 1992), with probability $1-\delta$ over the draw of the set $\{z_1, \ldots, z_N\} \sim \mathcal{D}^N$, for any selector $f \in \mathcal{F}$,

$$\left|\frac{1}{N}\sum_{i=1}^{N} u_{f(z_i)}(z_i) - \mathop{\mathbb{E}}_{z\sim\mathcal{D}}\left[u_{f(z)}(z)\right]\right| = O\left(H\sqrt{\frac{1}{N}\left(\text{Pdim}\left(\mathcal{U}_\mathcal{F}\right) + \log\frac{1}{\delta}\right)}\right). \quad (2)$$

We provide a general bound on $\text{Pdim}\left(\mathcal{U}_\mathcal{F}\right)$, which allows us to bound Equation (1). The proof is in the full version (Balcan, Sandholm, and Vitercik 2020).

Theorem 3.4. *Suppose each dual function u_z^* is piecewise-constant with at most t pieces. Let $\bar{d}$ be the Natarajan dimension of $\bar{\mathcal{F}}$. Then* $\text{Pdim}\left(\mathcal{U}_\mathcal{F}\right) = \tilde{O}\left(\bar{d} + \kappa \log t\right)$.

At a high level, the $\tilde{O}\left(\bar{d}\right)$ term accounts for the intrinsic complexity of the algorithm selectors $\mathcal{F}$. The $O\left(\kappa \log t\right)$ term accounts for the complexity of composing selectors f with the performance functions u_ρ. In Theorem 3.5, we prove this bound is tight up to logarithmic factors.

Proof sketch of Theorem 3.4. Let $z_1, \ldots, z_N \in \mathcal{Z}$ be an arbitrary set of problem instances. Since each dual function $u_{z_i}^*$ is piecewise-constant with at most t pieces, there are $M \leq Nt$ intervals $I_1, \ldots, I_M$ partitioning $\mathbb{R}$ where for any interval I_j and any instance z_i, $u_{z_i}^*(\rho)$ is constant across all $\rho \in I_j$. Given these intervals, we partition the algorithm selectors in $\mathcal{F}$ into at most M^κ sets so that within any one set, all selectors map to the same κ (or fewer) intervals. Focusing on the selectors within one set $\mathcal{F}_0$ of the partition, we prove that the number of ways the utility functions u_f across $f \in \mathcal{F}_0$ can labels the instances $z_1, \ldots, z_N$ is upper bounded by the number of ways the multi-class projection functions $\bar{f}$ across $f \in \mathcal{F}_0$ can label the instances. We can then use the Natarajan dimension of $\bar{\mathcal{F}}$ to bound the number of ways the functions in $\mathcal{U}_\mathcal{F}$ label the instances $z_1, \ldots, z_N$. □

Theorem 3.4 and Equation (2) imply that with probability $1-\delta$ over the draw $\mathcal{S} \sim \mathcal{D}^N$, for any selector $f \in \mathcal{F}$,

$$\left|\frac{1}{N}\sum_{z\in\mathcal{S}} u_{f(z)}(z) - \mathop{\mathbb{E}}_{z\sim\mathcal{D}}\left[u_{f(z)}(z)\right]\right| = \tilde{O}\left(H\sqrt{\frac{1}{N}\left(\bar{d} + \kappa + \log\frac{1}{\delta}\right)}\right). \quad (3)$$

This theorem quantifies a fundamental tradeoff: as the portfolio size increases, we can hope to obtain better and better empirical performance $\sum_{z\in\mathcal{S}} u_{f(z)}(z)$ but the generalization error $\tilde{O}\left(H\sqrt{\left(\bar{d}+\kappa\right)/N}\right)$ will worsen.

We now prove that Theorem 3.4 is tight up to logarithms. The following theorem illustrates that even if the class of algorithm selectors is extremely simple (in that the Natarajan dimension of $\bar{\mathcal{F}}$ is 0), if the portfolio size (that is, the number κ of parameters mapped to) is large, we cannot hope to avoid overfitting. The full proof is in the full version (Balcan, Sandholm, and Vitercik 2020).

Theorem 3.5. *For any $\kappa, \bar{d} \geq 2$, there is a class of functions $\mathcal{U} = \{u_\rho : \rho \in \mathbb{R}\}$ and a class of selectors $\mathcal{F}$ such that:*

1. *Each selector $f \in \mathcal{F}$ maps to $\leq \kappa$ parameter settings.*
2. *Each dual function u_z^* is piecewise-constant with 1 discontinuity,*
3. *The Natarajan dimension of $\bar{\mathcal{F}}$ is at most $\bar{d}$, and*
4. *The pseudo-dimension of $\mathcal{U}_{\mathcal{F}}$ is $\Omega\left(\kappa + \bar{d}\right)$.*

Proof sketch. Let $\mathcal{Z} = (0, 1]$. For each parameter setting $\rho \in \mathbb{R}$, define $u_\rho(z) = \mathbf{1}_{\{z \leq \rho\}}$. Let $\kappa, \bar{d} \geq 2$ be two arbitrary integers. We split this proof into two cases: $\bar{d} \geq \kappa$ and $\kappa > \bar{d}$. In both cases, we construct a class of selectors $\mathcal{F}$ that satisfies the properties in the theorem statement and we prove that $\text{Pdim}\left(\mathcal{U}_{\mathcal{F}}\right) \geq \max\left\{\kappa, \bar{d}\right\} = \Omega\left(\kappa + \bar{d}\right)$. We sketch the proof of the case where $\kappa > \bar{d}$.

We begin by partitioning $\mathcal{Z} = (0, 1]$ into κ intervals $Z_1, \ldots, Z_\kappa$, where $Z_i = \left(\frac{i-1}{\kappa}, \frac{i}{\kappa}\right]$. For each set $C \subseteq [\kappa]$, we define an selector $f_C : \mathcal{Z} \to \mathbb{R}$ as follows. For any $z \in \mathcal{Z}$, let i be the index of the interval z lies in, i.e., $z \in Z_i$. If $i \in C$, we map $f_C(z) = \frac{i}{\kappa}$ and if $i \notin C$, we map $f_C(z) = \frac{i}{\kappa} - \frac{1}{2\kappa}$. Let $\mathcal{F} = \{f_C : C \subseteq [\kappa]\}$. The multi-class projection of $\bar{\mathcal{F}}$ is extremely simple: its Natarjan dimension is 0. Moreover, the set $\mathcal{S} = \left\{\frac{1}{\kappa}, \frac{2}{\kappa}, \ldots, \frac{\kappa-1}{\kappa}, 1\right\}$ is shattered by $\mathcal{U}_{\mathcal{F}}$ because—at a high level—each selector f_C maps each element $z \in \mathcal{S}$ to a parameter just above z or just below z, which allows the function class $\mathcal{U}_{\mathcal{F}}$ to shatter $\mathcal{S}$. □

In the proof of Theorem 3.5, each performance function u_ρ maps to $\{0, 1\}$, so we effectively prove a lower bound on the *VC dimension* of $\mathcal{U}_{\mathcal{F}}$. Classic results from learning theory imply the generalization error of learning a selector $f \in \mathcal{F}$ can therefore be as large as $\tilde{\Omega}\left(H\sqrt{\left(\bar{d} + \kappa\right)/N}\right)$, which matches Equation (3) up to logarithmic factors.

4 Application of Theory to Algorithm Selectors

We now instantiate Theorem 3.4 for several commonly-used classes of algorithm selectors. In each of the case studies, there is a feature mapping $\phi : \mathcal{Z} \to \mathbb{R}^m$ that assigns feature vectors $\phi(z) \in \mathbb{R}^m$ to problem instances $z \in \mathcal{Z}$.

4.1 Linear Performance Models

We begin by providing guarantees for algorithm selectors that use a linear performance model. These have been used extensively in computational research (Xu et al. 2008; Xu, Hoos, and Leyton-Brown 2010). To define this type of selector, let $\boldsymbol{\rho} = (\rho_1, \ldots \rho_\kappa)$ be a set of κ distinct parameter settings. For each $i \in [\kappa]$, define a vector $\boldsymbol{w}_i \in \mathbb{R}^m$ and let

$$W = \begin{pmatrix} | & \cdots & | \\ \boldsymbol{w}_1 & \ddots & \boldsymbol{w}_\kappa \\ | & \cdots & | \end{pmatrix}$$

be a matrix containing all κ weight vectors. The dot product $\boldsymbol{w}_i \cdot \phi(z)$ is meant to estimate the performance of the algorithm parameterized by ρ_i on instance z. We define the algorithm selector $f_{\boldsymbol{\rho},W}(z) = \rho_i$ where $i = \text{argmax}_{j \in [\kappa]}\left\{\boldsymbol{w}_j \cdot \phi(z)\right\}$, which selects the parameter setting with best predicted performance. We define the class of algorithm selectors $\mathcal{F}_L = \{f_{\boldsymbol{\rho},W} : W \in \mathbb{R}^{m \times \kappa}, \boldsymbol{\rho} \in \mathbb{R}^\kappa\}$. To define the class $\bar{\mathcal{F}}_L$, for each matrix $W \in \mathbb{R}^{m \times \kappa}$, let $g_W : \mathcal{Z} \to [\kappa]$ be a function where $g_W(z) = \text{argmax}_{i \in [\kappa]}\left\{\boldsymbol{w}_i \cdot \phi(z)\right\}$. By definition, $\bar{\mathcal{F}}_L = \{g_W : W \in \mathbb{R}^{m \times \kappa}\}$, so $\bar{\mathcal{F}}_L$ is the well-studied *m-dimensional linear class* which has a Natarajan dimension of $O(m\kappa)$ (Shalev-Shwartz and Ben-David 2014). This fact implies the following corollary.

Corollary 4.1. *Suppose the dual functions are piecewise-constant with at most t pieces. The pseudo-dimension of $\mathcal{U}_{\mathcal{F}_L} = \left\{z \mapsto u_{f(z)} : f \in \mathcal{F}_L\right\}$ is $O(\kappa m \log(\kappa m) + \kappa \log t)$.*

4.2 Regression Tree Performance Models

We now analyze algorithm selectors that use a regression tree as the performance model. These have proven powerful in computational research (Hutter et al. 2014). A regression tree T's leaf nodes partition the feature space $\mathbb{R}^m$ into disjoint regions $R_1, \ldots, R_\ell$. In each region R_i, a constant value c_i is used to predict the algorithm's performance on instances in the region. The internal nodes of the tree define this partition: each performs an inequality test on some feature of the input. We use the notation $h_T(z)$ to denote tree T's prediction of the algorithm's performance on instance z. Formally, $h_T(z)$ equals the constant value corresponding to the region of the tree's partition to which $\phi(z)$ belongs.

An algorithm selector can be defined using a regression tree performance model as follows. Let $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_\kappa)$ be a set of κ parameter settings. For each ρ_i, let T_i be a tree that is meant to predict the performance of the algorithm parameterized by ρ_i, and let $\boldsymbol{T} = (T_1, \ldots, T_\kappa)$ be the set of all κ trees. We define the algorithm selector $f_{\boldsymbol{\rho},\boldsymbol{T}}(z) = \rho_i$ where $i = \text{argmax}_{j \in [\kappa]}\left\{h_{T_j}(z)\right\}$. The class of algorithm selectors $\mathcal{F}_R$ consists of all functions $f_{\boldsymbol{\rho},\boldsymbol{T}}$ across all parameter vectors $\boldsymbol{\rho} \in \mathbb{R}^\kappa$ and all κ-tuples of regression trees $\boldsymbol{T} = (T_1, \ldots, T_\kappa)$. The full proof of the following lemma is in the full version (Balcan, Sandholm, and Vitercik 2020).

Lemma 4.2. *Suppose we limit ourselves to building regression trees with at most ℓ leaves. Then the Natarajan dimension of $\bar{\mathcal{F}}_R$ is $O(\ell\kappa \log(\ell\kappa m))$.*

Proof sketch. For each κ-tuple of regression trees $\boldsymbol{T} = (T_1, \ldots, T_\kappa)$, let $g_{\boldsymbol{T}} : \mathcal{Z} \to [\kappa]$ be a function where $g_{\boldsymbol{T}}(z) = \text{argmax}_{i \in [\kappa]}\left\{h_{T_i}(z)\right\}$. By definition, the set $\bar{\mathcal{F}}_R$ consists of the functions $g_{\boldsymbol{T}}$ across all κ-tuples of regression trees $\boldsymbol{T}$ with at most ℓ leaves. Let $z_1, \ldots, z_N \in \mathcal{Z}$ be a set of problem instances. Our goal is to bound the number of ways the

functions $g_{\boldsymbol{T}}$ can label these instances. A single regression tree induces a partition of these N problem instances defined by which leaf each instance is mapped to as we apply the tree's inequality tests. The key step in this proof is bounding the total number of partitions we can induce by varying the tree's inequality tests. We then generalize this intuition to bound the number of partitions κ regression trees can induce as we vary all their parameters. Once the partition of each regression tree is fixed, the tree with the largest prediction for each problem instance depends on the relative ordering of the constants at the trees' leaves. There is a bounded number of possible relative orderings, and we aggregate all of these bounds to prove the lemma statement. □

Corollary 4.3. *Suppose the dual functions are piecewise-constant with at most t pieces and we limit ourselves to building regression trees with at most ℓ leaves. Then* $\text{Pdim}\left(\mathcal{U}_{\mathcal{F}_R}\right) = O(\ell\kappa\log(\ell\kappa m) + \kappa\log t)$.

This pseudo-dimension bound reflects the end-to-end nature of our analysis, since the guarantee bounds the generalization error of both selecting the portfolio and training the regression tree performance model. This is why the bound grows with both the size of the portfolio (κ) and the complexity of the regression trees (ℓ and m).

4.3 Clustering-Based Algorithm Selectors

We now provide guarantees for clustering-based algorithm selectors (Kadioglu et al. 2010). This type of selector clusters the feature vectors $\phi(z_1), \ldots, \phi(z_N) \in \mathbb{R}^m$ and chooses a good parameter setting for each cluster. On a new instance z, the selector determines which cluster center is closest to $\phi(z)$ and runs the algorithm using the parameter setting assigned to that cluster. More formally, let $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_\kappa)$ be a set of parameter settings and let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_\kappa \in \mathbb{R}^m$ be a set of vectors. We define the matrix

$$X = \begin{pmatrix} | & \cdots & | \\ \boldsymbol{x}_1 & \ddots & \boldsymbol{x}_\kappa \\ | & \cdots & | \end{pmatrix},$$

where each $\boldsymbol{x}_i$ represents a cluster center. We define the selector $f_{\boldsymbol{\rho},X}(z) = \rho_i$ with $i = \text{argmin}_{j\in[\kappa]}\left\{\|\boldsymbol{x}_j - \phi(z)\|_p\right\}$ for some ℓ_p-norm with $p \geq 1$, and the class $\mathcal{F}_C = \{f_{\boldsymbol{\rho},X} : \boldsymbol{\rho} \in \mathbb{R}^\kappa, X \in \mathbb{R}^{m\times\kappa}\}$. This lemma's proof is in the full version (Balcan, Sandholm, and Vitercik 2020).

Lemma 4.4. *For any $p \in [1,\infty)$, the Natarajan dimension of $\bar{\mathcal{F}}_C$ is $O(m\kappa\log(m\kappa p))$.*

Proof sketch. For each matrix X, let $g_X : \mathcal{Z} \to [\kappa]$ be defined such that $g_X(z) = \text{argmin}_{i\in[\kappa]}\left\{\|\boldsymbol{x}_i - \phi(z)\|_p^p\right\}$. By definition, $\bar{\mathcal{F}}_C = \{g_X : X \in \mathbb{R}^{m\times\kappa}\}$. Let $z_1, \ldots, z_N \in \mathcal{Z}$ be a set of problem instances. Our goal is to bound the number of ways the functions g_X can label these instances as we vary $X \in \mathbb{R}^{m\times\kappa}$. We do so by analyzing, for each instance z_i, the boundaries in $\mathbb{R}^{m\times\kappa}$ where if we shift X from one side of the boundary to the other, the column in X closest to $\phi(z_i)$ changes. We show that these boundaries are defined by multi-dimensional polynomials. We bound the total number of regions these boundaries induce in $\mathbb{R}^{m\times\kappa}$, which implies a bound on the Natarajan dimension of $\bar{\mathcal{F}}_C$. □

Lemma 4.4 and Theorem 3.4 imply the following bound.

Corollary 4.5. *If the dual functions are piecewise-constant with at most t pieces, then* $\text{Pdim}\left(\mathcal{U}_{\mathcal{F}_C}\right) = \tilde{O}\left(m\kappa + \kappa\log t\right)$.

5 Learning Procedure with Guarantees

We use the results from the previous section to provide guarantees for the high-level learning procedure from Section 2:

1. Draw a training set of problem instances $\mathcal{S} \sim \mathcal{D}^N$.
2. Use $\mathcal{S}$ to select a set of at most κ configurations $\hat{\mathcal{P}} \subseteq \mathbb{R}$.
3. Use $\mathcal{S}$ to learn an algorithm selector $\hat{f} \in \mathcal{F}$ that maps problem instances $z \in \mathcal{Z}$ to parameter settings $\hat{f}(z) \in \hat{\mathcal{P}}$.

Our guarantees depend on the quality of the portfolio $\hat{\mathcal{P}}$ and selector $\hat{f}$, as formalized by the following definition.

Definition 5.1. Given a training set $\mathcal{S} \subseteq \mathcal{Z}^N$ and parameters $\alpha \in (0,1]$, $\beta \in [0,1]$, and $\epsilon \in [0,1]$, we say the portfolio $\hat{\mathcal{P}}$ and the algorithm selector $\hat{f}$ are (α,β,ϵ)-optimal if:

1. The portfolio $\hat{\mathcal{P}}$ is nearly optimal over the training set:

$$\frac{1}{N}\sum_{z\in\mathcal{S}}\max_{\rho\in\hat{\mathcal{P}}} u_\rho(z) \geq \alpha \max_{\mathcal{P}\subset\mathbb{R}:|\mathcal{P}|\leq\kappa}\frac{1}{N}\sum_{z\in\mathcal{S}}\max_{\rho\in\mathcal{P}} u_\rho(z) - \beta.$$

   (The maximization means that performance is measured with respect to an oracle that selects an optimal algorithm parameter ρ from the portfolio for each instance.)

2. The algorithm selector $\hat{f}$ returns high-performing parameter settings from the set $\hat{\mathcal{P}}$ in the sense that

$$\frac{1}{N}\sum_{z\in\mathcal{S}} u_{\hat{f}(z)}(z) \geq \frac{1}{N}\sum_{z\in\mathcal{S}}\max_{\rho\in\hat{\mathcal{P}}} u_\rho(z) - \epsilon. \qquad (4)$$

For example, when algorithmic performance as a function of the parameters is piecewise constant, there are only a finite number of meaningfully different parameter values to choose among—one per piece. Then, since $\sum_{z\in\mathcal{S}}\max_{\rho\in\hat{\mathcal{P}}} u_\rho(z)$ is a submodular function of the portfolio $\hat{\mathcal{P}}$, we can use a greedy algorithm to select $\hat{\mathcal{P}}$, and we obtain $\alpha = 1 - \frac{1}{e}$ and $\beta = 0$, as we prove in the full version (Balcan, Sandholm, and Vitercik 2020). Alternatively, integer programming could be used to select the optimal portfolio from the finite set of candidate parameter values, in which case we would obtain $\alpha = 1$ and $\beta = 0$. Moreover, the value ϵ can be calculated directly from the training set.

The following theorem bounds the difference between the expected performance of the selector $\hat{f}$ and an oracle that selects an optimal selector and an optimal portfolio. The proof is in the full version (Balcan, Sandholm, and Vitercik 2020).

Theorem 5.2. *Suppose that each dual function u_z^* is piecewise constant with at most t pieces. Given a training set $\mathcal{S} \subseteq \mathcal{Z}$ of size N, suppose we learn an (α,β,ϵ)-optimal*

portfolio $\hat{\mathcal{P}} \subset \mathbb{R}$ and algorithm selector $\hat{f} : \mathcal{Z} \to \hat{\mathcal{P}}$ in $\mathcal{F}$. With probability $1-\delta$ over the draw $\mathcal{S} \sim \mathcal{D}^N$,

$$\mathop{\mathbb{E}}_{z\sim\mathcal{D}}\left[u_{\hat{f}(z)}(z)\right] \geq \alpha \max_{\mathcal{P}:|\mathcal{P}|\leq\kappa} \mathbb{E}\left[\max_{\rho\in\mathcal{P}} u_\rho(z)\right] - \epsilon - \beta - \tilde{O}\left(H\sqrt{\frac{\bar{d}+\kappa}{N}}\right),$$

where $\bar{d}$ is the Natarajan dimension of $\bar{\mathcal{F}}$.

Proof sketch. First, let $\mathcal{P}^*$ be the optimal portfolio in the sense that $\mathcal{P}^* = \text{argmax}_{\mathcal{P}\subset\mathbb{R}:|\mathcal{P}|\leq\kappa} \mathbb{E}_{z\sim\mathcal{D}}\left[\max_{\rho\in\mathcal{P}} u_\rho(z)\right]$. We use a Hoeffding bound to relate the expected performance of $\mathcal{P}^*$ under the oracle algorithm selector and its average performance over the training set. We then use Definition 5.1 to relate the latter quantity to the average performance of the learned selector $\hat{f}$ over the training set. Finally, we use Theorem 3.4 to relate the average performance of $\hat{f}$ to its expected performance. Putting all of these bounds together, we prove the theorem statement. □

We can obtain symmetric guarantees when our goal is to minimize rather than maximize a performance measure.

6 Experiments

We provide experiments that illustrate the tradeoff we investigated from a theoretical perspective in the previous sections: as we increase the portfolio size, we can hope to include a well-suited parameter setting for any problem instance, but it becomes increasingly difficult to avoid overfitting. We illustrate this in the context of integer programming algorithm configuration. We configure CPLEX, one of the most widely used commercial solvers. CPLEX uses the *branch-and-cut (B&C)* algorithm (branch-and-bound with cutting planes, primal heuristics, preprocessing, etc.) to solve integer programs (IPs). We tune a parameter $\rho \in [0,1]$ of CPLEX that controls its *variable selection policy*[2] and has been studied extensively in prior research (Gauthier and Ribière 1977; Bénichou et al. 1971; Beale 1979; Linderoth and Savelsbergh 1999; Achterberg 2009; Balcan et al. 2018). We leave CPLEX's other techniques on and unchanged. We provide a more detailed overview of CPLEX and the parameter we tune in the full version (Balcan, Sandholm, and Vitercik 2020). B&C partitions the IP's feasible region, finding locally optimal solutions within the sets of the partition, and eventually verifies that the best solution found so far is globally optimal. It organizes this partition as a tree. As in prior research (Balcan et al. 2018; Gupta et al. 2020; Zarpellon et al. 2020), our goal is to find parameter settings leading to small trees—and thereby fast run time—so we define $u_\rho(z)$ to be the size of the tree B&C builds. We aim to learn a portfolio $\hat{\mathcal{P}}$ and selector $\hat{f}$ resulting in small expected tree size $\mathbb{E}\left[u_{\hat{f}(z)}(z)\right]$.

[2]We override the default variable selection of CPLEX 12.8.0.0 using the C API. All experiments were run on a 64-core machine with 512 GB of RAM, a m4.16xlarge Amazon AWS instance, and a cluster of m4.xlarge Amazon AWS instances.

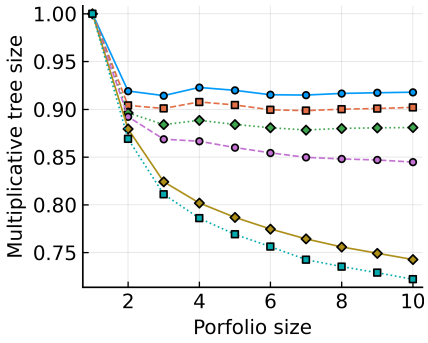
Distribution over IPs. We analyze a distribution over IPs formulating the combinatorial auction winner determination problem, which we generate using the Combinatorial Auction Test Suite (Leyton-Brown, Pearson, and Shoham 2000). We use the "arbitrary" generator with 200 bids and 100 goods, resulting in IPs with around 200 variables, and the "regions" generator with 400 bids and 200 goods, resulting in IPs with around 400 variables. We define a heterogeneous distribution $\mathcal{D}$ as follows: with equal probability, we draw an instance from the "arbitrary" or "regions" distributions. To assign features to these IPs, we use all the features developed in prior research by Leyton-Brown, Pearson, and Shoham (2000) and Hutter et al. (2014), resulting in 140 features.

Experimental procedure. We first learn a portfolio of size 10 in the following way. We draw a training set of $M = 1000$ IPs $z_1, \ldots, z_M \sim \mathcal{D}$ and solve for the dual functions $u^*_{z_1}, \ldots, u^*_{z_M}$—which measure tree size as a function of the parameter ρ—using the algorithm described in Appendix D.1 of the paper by Balcan et al. (2018). These functions are piecewise-constant with at most t pieces, for some $t \in \mathbb{N}$. Therefore, there are at most Mt parameter settings leading to different algorithmic performance over the training set. Let $\bar{\mathcal{P}}$ be this set of parameter settings. We use a greedy algorithm to select 10 parameter settings from $\bar{\mathcal{P}}$. First, we find a parameter setting ρ_1 which minimizes average tree size over the training set: $\rho_1 \in \text{argmin} \sum_{i=1}^M u_{z_i^*}(\rho)$. Then, we find a parameter setting ρ_2 that minimizes average tree size when the better of ρ_1 or ρ_2 is used: $\rho_2 \in \text{argmin} \sum_{i=1}^M \min\left\{u_{z_i^*}(\rho), u_{z_i^*}(\rho_1)\right\}$. We continue greedily until we have a portfolio $\hat{\mathcal{P}} = \{\rho_1, \ldots, \rho_{10}\}$.

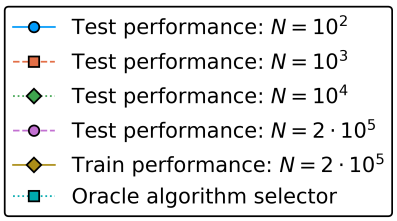
We then use a regression forest to select among parameter settings in the portfolio $\hat{\mathcal{P}}$. Prior research (Hutter et al. 2014) has illustrated that regression forests can be strong predictors of B&C runtime. Here, we use them to predict B&C tree size. A regression forest is a set $F = \{T_1, \ldots, T_M\}$ of regression trees (reviewed in Section 4.2). On an input IP z, the regression forest's prediction, denoted $h_F(z)$, is the average of the trees' predictions: $h_F(z) = \frac{1}{M}\sum_{i=1}^M h_{T_i}(z)$. We learn regression forests $F_1, \ldots, F_{10}$ for each of the 10 parameter settings in the portfolio $\hat{\mathcal{P}}$. We then define the algorithm selector $\hat{f}(z) = \rho_i$ where $i = \text{argmin}_{j\in[10]}\left\{h_{F_j}(z)\right\}$.

To learn the regression forest, we draw a training set $z_1, \ldots, z_N \sim \mathcal{D}$ of IPs (with N specified below). For each parameter setting $\rho_i \in \hat{\mathcal{P}}$ and IP z_j, we compute $u_{\rho_i}(z_j)$, the size of the tree B&C builds using the parameter setting ρ_i. We then train the forest F_i corresponding to the parameter setting ρ_i using the labeled training set $\{(z_1, u_{z_1}(\rho_i)), \ldots, (z_N, u_{z_N}(\rho_i))\}$. We use Python's default scikit-learn regression forest (Pedregosa et al. 2011).

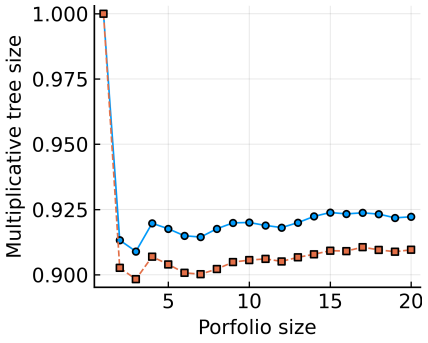
In Figure 1a, we plot the performance of the regression forests as both the training set and portfolio grow. For a fixed training set size N, we train the regression forests $F_1, \ldots, F_{10}$ using the method described above. We then evaluate performance as a function of the portfolio size κ. Specifically, for each $\kappa \in [10]$, we define a selector $\hat{f}_\kappa(z) = \rho_i$ where $i = \text{argmin}_{j\in[\kappa]}\left\{h_{F_j}(z)\right\}$. We draw $N_t = 10^4$ test



(a) Plot with portfolio sizes 1 through 10.



(b) Legend for Figures 1a and 1c.



(c) Plot with portfolio sizes 1 through 20.

Figure 1: In Figures 1a and 1c, we plot the multiplicative tree size improvement we obtain as we increase both the portfolio size along the horizontal axis and the size of the training set, denoted N. Fixing a training set size and letting $\hat{v}_\kappa$ be the average tree size we obtain over the test set using a portfolio of size κ (see Equation (5)), we plot $\hat{v}_\kappa/\hat{v}_1$. In Figure 1a, the portfolio size ranges from 1 to 10 and the training set size N ranges from 100 to 200,000. In Figure 1c, the portfolio size ranges from 1 to 20 and the training set size ranges from 100 to 1000. In Figure 1a, we also plot a similar curve for the test performance of the oracle algorithm selector, as well as the training performance of the learned algorithm selector when $N = 2 \cdot 10^5$.

instances $\mathcal{S}_t \sim \mathcal{D}^{N_t}$. The test performance of $\hat{f}_\kappa$ is

$$\hat{v}_\kappa = \frac{1}{N_t} \sum_{z \in \mathcal{S}_t} u_{\hat{f}_\kappa(z)}(z). \tag{5}$$

In Figure 1a, we plot the multiplicative performance improvement we obtain as we increase κ. Specifically, we plot $\hat{v}_\kappa/\hat{v}_1$. These are the blue solid ($N = 10^2$), orange dashed ($N = 10^3$), green dotted ($N = 10^4$), and purple dashed ($N = 2 \cdot 10^5$) lines. By the iterative fashion we constructed the portfolio, $\hat{v}_1$ is the performance of the best single parameter setting, so $\hat{v}_1$ is already highly optimized.

We plot a similar curve for the test performance of the oracle algorithm selector which always selects the optimal parameter setting from the portfolio. Specifically, for each portfolio size $\kappa \in [10]$, let f^*_κ be the oracle algorithm selector $f^*_\kappa(z) = \text{argmin}_{\rho_1,\ldots,\rho_\kappa} u_{\rho_i}(z)$. Given a test set $\mathcal{S}_t \sim \mathcal{D}^{N_t}$, we define the average test performance of f^*_κ as

$$v^*_\kappa = \frac{1}{N_t} \sum_{z \in \mathcal{S}_t} u_{f^*_\kappa(z)}(z).$$

The blue dotted line equals v^*_κ/v^*_1 as a function of κ.

Finally, when the training set is of size $N = 2 \cdot 10^5$, we provide a similar curve for the training performance of the learned algorithm selectors $\hat{f}_\kappa$. Letting $z_1, \ldots, z_N$ be the training set, we denote the average training performance as

$$\tilde{v}_\kappa = \frac{1}{N} \sum_{i=1}^{N} u_{f^*_\kappa(z_i)}(z_i).$$

The yellow solid line is $\tilde{v}_\kappa/\tilde{v}_1$ as a function of κ.

In Figure 1c, we plot $\hat{v}_\kappa/\hat{v}_1$ as a function of the portfolio size κ for larger portfolio sizes ranging from 1 to 20. We greedily extend the portfolio $\hat{\mathcal{P}}$ to include an additional 20 parameter settings. We then train 20 regression forests using freshly drawn training sets of size 100 and 1000. This plot illustrates the fact that as we increase the portfolio size, overfitting causes test performance to worsen.

Discussion. Focusing first on test performance using the largest training set size $N = 2 \cdot 10^5$, we see that test performance continues to improve as we increase the portfolio size, though training and test performance steadily diverge. This illustrates the tradeoff we investigated from a theoretical perspective in this paper: as we increase the portfolio size, we can hope to include a well-suited parameter setting for every instance, but the generalization error will worsen. Figure 1c shows that for a given training set size, there is a portfolio size after which test performance actually starts to get strictly worse, as our theory predicts. In other words, we observe overfitting: the selector has strong average performance over the training set but poor test performance.

7 Conclusions

We provided guarantees for learning a portfolio of parameter settings in conjunction with an algorithm selector for that portfolio. We provided a tight (up to log factors) bound on the number of samples sufficient and necessary to ensure that the selector's average performance on the training set generalizes to its expected performance on the real unknown problem instance distribution. Our guarantees apply in the widely-applicable setting where the algorithm's performance on any input problem instance is a piecewise-constant function of its parameters. Our theoretical bounds indicate that even with an extremely simple algorithm selector, we cannot hope to avoid overfitting in the worst-case if the portfolio is large. Thus, there is a tradeoff when increasing the portfolio size, since a large portfolio allows for the possibility of including a strong parameter setting for every instance, but this potential for performance improvement is overshadowed by a worsening propensity towards overfitting. We concluded with experiments illustrating this tradeoff in the context of integer programming. A direction for future research is to understand how the diversity of a portfolio impacts its generalization error, since algorithm portfolios are often expressly designed to be diverse.

Acknowledgments

This material is based on work supported by the National Science Foundation under grants CCF-1535967, CCF-1733556, CCF-1910321, IIS-1617590, IIS-1618714, IIS-1718457, IIS-1901403, and SES-1919453; the ARO under awards W911NF1710082 and W911NF2010081; the Defense Advanced Research Projects Agency under cooperative agreement HR00112020003; an AWS Machine Learning Research Award; an Amazon Research Award; a Bloomberg Research Grant; a Microsoft Research Faculty Fellowship; an IBM PhD fellowship; and a fellowship from Carnegie Mellon University's Center for Machine Learning and Health.

References

Achterberg, T. 2009. SCIP: solving constraint integer programs. *Mathematical Programming Computation* 1(1): 1–41.

Balcan, M.-F. 2020. Data-Driven Algorithm Design. In Roughgarden, T., ed., *Beyond Worst Case Analysis of Algorithms*. Cambridge University Press.

Balcan, M.-F.; DeBlasio, D.; Dick, T.; Kingsford, C.; Sandholm, T.; and Vitercik, E. 2021. How much data is sufficient to learn high-performing algorithms? In *Proceedings of the Annual Symposium on Theory of Computing (STOC)*.

Balcan, M.-F.; Dick, T.; Sandholm, T.; and Vitercik, E. 2018. Learning to Branch. In *International Conference on Machine Learning (ICML)*.

Balcan, M.-F.; Nagarajan, V.; Vitercik, E.; and White, C. 2017. Learning-Theoretic Foundations of Algorithm Configuration for Combinatorial Partitioning Problems. *Conference on Learning Theory (COLT)* .

Balcan, M.-F.; Sandholm, T.; and Vitercik, E. 2020. Generalization in portfolio-based algorithm selection. *arXiv preprint arXiv:2012.13315* .

Beale, E. 1979. Branch and bound methods for mathematical programming systems. *Annals of Discrete Mathematics* 5: 201–219.

Bénichou, M.; Gauthier, J.-M.; Girodet, P.; Hentges, G.; Ribière, G.; and Vincent, O. 1971. Experiments in mixed-integer linear programming. *Mathematical Programming* 1(1): 76–94.

Garg, V.; and Kalai, A. 2018. Supervising Unsupervised Learning. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*.

Gauthier, J.-M.; and Ribière, G. 1977. Experiments in mixed-integer linear programming using pseudo-costs. *Mathematical Programming* 12(1): 26–47.

Gupta, P.; Gasse, M.; Khalil, E.; Mudigonda, P.; Lodi, A.; and Bengio, Y. 2020. Hybrid models for learning to branch. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*.

Gupta, R.; and Roughgarden, T. 2017. A PAC approach to application-specific algorithm selection. *SIAM Journal on Computing* 46(3): 992–1017.

Haussler, D. 1992. Decision theoretic generalizations of the PAC model for neural net and other learning applications. *Information and computation* 100(1): 78–150.

Hutter, F.; Xu, L.; Hoos, H. H.; and Leyton-Brown, K. 2014. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence* 206: 79–111.

Kadioglu, S.; Malitsky, Y.; Sellmann, M.; and Tierney, K. 2010. ISAC—Instance-Specific Algorithm Configuration. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*.

Leyton-Brown, K. 2003. *Resource allocation in competitive multiagent systems*. Ph.D. thesis, Stanford University.

Leyton-Brown, K.; Pearson, M.; and Shoham, Y. 2000. Towards a Universal Test Suite for Combinatorial Auction Algorithms. In *Proceedings of the ACM Conference on Electronic Commerce (ACM-EC)*, 66–76. Minneapolis, MN.

Linderoth, J.; and Savelsbergh, M. 1999. A computational study of search strategies for mixed integer programming. *INFORMS Journal of Computing* 11(2): 173–187.

Liu, S.; Tang, K.; Lei, Y.; and Yao, X. 2020. On Performance Estimation in Automatic Algorithm Configuration. In *AAAI Conference on Artificial Intelligence (AAAI)*.

Natarajan, B. K. 1989. On learning sets and functions. *Machine Learning* 4(1): 67–97.

Núñez, S.; Borrajo, D.; and López, C. L. 2015. Automatic construction of optimal static sequential portfolios for AI planning and beyond. *Artificial Intelligence* 226: 75–101.

Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; and Duchesnay, E. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12: 2825–2830.

Sandholm, T. 2013. Very-Large-Scale Generalized Combinatorial Multi-Attribute Auctions: Lessons from Conducting $60 Billion of Sourcing. In Neeman, Z.; Roth, A.; and Vulkan, N., eds., *Handbook of Market Design*. Oxford University Press.

Shalev-Shwartz, S.; and Ben-David, S. 2014. *Understanding machine learning: From theory to algorithms*. Cambridge University Press.

Vapnik, V.; and Chervonenkis, A. 1971. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications* 16(2): 264–280.

Xu, L.; Hoos, H.; and Leyton-Brown, K. 2010. Hydra: Automatically Configuring Algorithms for Portfolio-Based Selection. In *AAAI Conference on Artificial Intelligence (AAAI)*.

Xu, L.; Hutter, F.; Hoos, H.; and Leyton-Brown, K. 2008. SATzilla: portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research* 32(1): 565–606.

Zarpellon, G.; Jo, J.; Lodi, A.; and Bengio, Y. 2020. Parameterizing Branch-and-Bound Search Trees to Learn Branching Policies. *arXiv preprint arXiv:2002.05120* .