# Learnable and Instance-Robust Predictions for Online Matching, Flows and Load Balancing

**Thomas Lavastida**
Carnegie Mellon University, Pittsburgh, PA, USA

**Benjamin Moseley**
Carnegie Mellon University, Pittsburgh, PA, USA

**R. Ravi**
Carnegie Mellon University, Pittsburgh, PA, USA

**Chenyang Xu**[1]
Zhejiang University, China

## Abstract

We propose a new model for augmenting algorithms with predictions by requiring that they are formally learnable and instance robust. Learnability ensures that predictions can be efficiently constructed from a reasonable amount of past data. Instance robustness ensures that the prediction is robust to modest changes in the problem input, where the measure of the change may be problem specific. Instance robustness insists on a smooth degradation in performance as a function of the change. Ideally, the performance is never worse than worst-case bounds. This also allows predictions to be objectively compared.

We design online algorithms with predictions for a network flow allocation problem and restricted assignment makespan minimization. For both problems, two key properties are established: high quality predictions can be learned from a small sample of prior instances and these predictions are robust to errors that smoothly degrade as the underlying problem instance changes.

**2012 ACM Subject Classification** Theory of computation → Online algorithms

**Keywords and phrases** Learning-augmented algorithms, Online algorithms, Flow allocation

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2021.59

**Related Version** *Full Version*: `https://arxiv.org/abs/2011.11743`

**Funding** *Chenyang Xu:* Supported in part by Science and Technology Innovation 2030 – "The Next Generation of Artificial Intelligence" Major Project No.2018AAA0100902 and China Scholarship Council No.201906320329.
*Thomas Lavastida, and Benjamin Moseley:* Supported in part by a Google Research Award, an Infor Research Award, a Carnegie Bosch Junior Faculty Chair and NSF grants CCF-1824303, CCF-1845146, CCF-1733873 and CMMI-1938909.
*R. Ravi*: Supported in part by the U. S. Office of Naval Research under award number N00014-21-1-2243 and the Air Force Office of Scientific Research under award number FA9550-20-1-0080.

## 1 Introduction

Inspired by advances in machine learning, there is an interest in augmenting algorithms with predictions, especially in online algorithm design [21, 10, 12, 15, 26, 28, 34, 27]. Algorithms augmented with predictions have had empirical success in domains such as look-up tables [26], caching [28], and bloom-filters [31]. These successes and the availability of data to make predictions using machine learning have motivated the development of new analysis models

[1] The corresponding author

for going beyond worst-case bounds where an algorithm is supplied with accurate predictions. In these models, an algorithm is given access to a prediction about the problem instance. The algorithm's performance is bounded in terms of the quality of this prediction. Typically the algorithm learns such predictions from a limited amount of past data leading to *error-prone* predictions. The algorithm with accurate predictions should result in better performance than the best worst-case bound. Ideally, the algorithm never performs asymptotically worse than the best worst-case algorithm even if the prediction error is large. In-between, there is a graceful degradation in performance as the prediction error increases. For example, competitive ratio or running time can be parameterized by prediction error. See [32] for a survey.

**Learnable and Instance Robust Predictions.** The model proposed in this paper has two pillars for augmenting algorithms with predictions. (**Learnability**:) Predictions should be learnable from representative data. (**Instance Robustness**[2]:) Predictions should be robust to minor changes in the problem instance. As in prior models, determining what to predict remains a key algorithmic challenge.

Suppose there is an unknown distribution $\mathcal{D}$ over instances $\mathcal{I}$. Building on data driven algorithm design [21] and PAC-learning models, we require that predicted parameters are provably learnable using a small number of sample instances from $\mathcal{D}$. The sample complexity of this task can be used to compare how difficult different predictions are to construct. Practically, the motivation is that parameters are learned from prior data (e.g. instances of the problem).

In practice, future problem instances may not come from the same distribution used to learn the parameters. Therefore, we also desire predictions that are robust to modest changes in the input. In particular, if the predictions perform well on some instance $\mathcal{I}$, then the performance on a nearby instance $\mathcal{I}'$ should be bounded as a function of the distance between these instances. This measure of the distance between instances is necessarily problem specific.

We note that learnability is rarely addressed in prior work and our robustness model differs from many prior works by bounding the error by differences in problem instances (instance robustness), rather than by the differences in the predictions themselves (parameter robustness). We present learnable and instance-robust predictions for two concrete online problems.

**Online Flow Allocation Problem.** We consider a general flow and matching problem. The input is a Directed-Acyclic-Graph (DAG) $G$. Each node $v$ has an associated capacity $C_v$. There is a sink node $t$, such that all nodes in the DAG can reach the sink. Online source nodes arrive that have no incoming edges (and never have any incoming edges in the future) and the other nodes are offline and fixed. We will refer to online nodes $I$ as impressions. When impression $i$ arrives, it is connected to a subset of nodes $N_i$. At arrival, the algorithm must decide a (fractional) flow from $i$ to $t$ of value at most 1 obeying the node capacities. This flow is fixed the moment the node arrives. The goal is to maximize the total flow that reaches $t$ without exceeding node capacities. Instances are defined by the number of each *type* of impression. The type of an impression is given by the subset of the nodes of $G$ to which it has outgoing arcs. We may consider specific worst-case instances or a distribution over types

[2] Note that the robustness here is different from the definition of robustness mentioned in previous work, which we refer to as parameter robustness. See Section 2 for a discussion.

in our analysis. This problem captures fractional versions of combinatorial problems such as online matching, unweighted Adwords, and finding a maximum independent set in a laminar matroid or gammoid. We call the problem the Online Flow Allocation Problem.

**Restricted Assignment Makespan Minimization.** In this problem, there are $m$ machines and $n$ jobs arrive in an online order. When job $j$ arrives it must be immediately and irrevocably assigned to a machine. The job has size $p_j$ and can only be assigned to a subset of machines specific to that job. After $j$ is assigned the next job arrives. A machine's load is the total size of all jobs assigned to it. The goal is to minimize the makespan, or maximum load, of the assignment.

## 1.1 Overview of Results for Flow Allocation and Restricted Assignment

We first focus on the flow allocation problem and then we give similar results for the makespan minimization problem.

**Node Parameters.** Our results on learnability and robustness are enabled by showing the existence of node weights which capture interesting combinatorial properties of flows. Inspired by the weights proven in [2] for bipartite matching, we establish that there is a single weight for each node in the DAG that completely describe near optimal flows on a single problem instance. The weights determine an allocation of flow for each impression which is independent of the other impressions. Each node in the DAG routes the flow leaving it proportionally to the weights of its outgoing neighbors. Moreover, the flow is near optimal, giving a $(1-\epsilon)$-approximate solution for any constant $\epsilon > 0$ (but requiring time polynomial in $1/\epsilon$ to compute). Given these weights, the flow can be computed in one forward pass for each impression in isolation. Thus they can be used online if given as a prediction. These weights are also efficiently computable offline given the entire problem instance (see Theorem 3).

**Instance Robustness.** We measure the distance of the two instances as the difference of the number of impressions of each type (see Theorem 5). We show that if the weights are near optimal for one instance, the performance degrades gracefully according to the distance between the two instances. This distance is defined for any two instances irrespective of whether they are generated by specific distributions [3].

**Learnability.** For learnability it is assumed that impressions are drawn from an unknown distribution over types. We show that learning near-optimal weights for this distribution has low sample complexity under two assumptions. First, we assume the unknown distribution is a product distribution. Second, we assume that the optimal solution of the "expected instance" (to be defined later) has at least a constant amount of flow routed through each node. In the 2-layer case, this assumption can be simplified to requiring each node's capacity to be at least a constant (depending on $\frac{1}{\epsilon}$).[4] The number of samples is polynomial in the size of the DAG without the impressions. Note that in problems such as Adwords, the impressions are usually much larger than the fixed portion of the graph.

We now present our main theorem on the flow allocation problem.

[3] We also show that our predictions for the online flow allocation problem have "parameter robustness", the kind of robustness that has been considered in prior work (Theorem 6).

[4] This is similar to the lower bound requirement on budgets in the online analysis of the AdWords problem [16].

▶ **Theorem 1** (Flow Allocation - Informal)**.** *There exist algorithmic parameters for the Online Flow Allocation problem with the following properties:*

**(i)** *(Learnability) Learning near-optimal parameters has sample complexity polynomial in $\frac{1}{\epsilon}$ and the size of the graph excluding the impressions. These parameters result in an online algorithm that is a $(1-\epsilon)$-approximate solution in expectation as compared to the expected optimal value on the distribution for any constant $\epsilon > 0$. (Theorem 4)*

**(ii)** *(Instance Robustness) Using the optimal parameters for an instance on another instance gives a competitive ratio that improves as their distance decreases, where the distance is proportional to the difference of impressions (Theorem 5).*

**(iii)** *(Worst-Case Robustness) The competitive ratio of the online algorithm using the parameters is never worse than $\frac{1}{d+1}$, regardless of the distance between the two instances, where $d$ is the diameter of $G$. (Theorem 5)*

The theorem states that weights are learnable and only a small number of samples are required to construct weights that are near optimal. These predictions break the worst-case $1 - \frac{1}{e}$ bound on the competitive ratio for any randomized algorithm for online fractional matching, a special case. Moreover, the difference in the types of impressions between two instances gives a metric under which we can demonstrate instance robustness. Further the algorithm has worst-case guarantees, i.e. the ratio is never worse than $\frac{1}{d+1}$, which is tight for deterministic integral online algorithms and $d$-layer graphs (see Theorem 61 in the full version) even though we output fractional allocations.

We now discuss our results for makespan minimization.

▶ **Theorem 2** (Restricted Assignment - Informal)**.** *There exist algorithmic parameters for the Restricted Assignment Makespan Minimization problem with the following properties:*

**(i)** *(Learnability) Learning the near optimal parameters has sample complexity polynomial in $m$, the number of machines, and $\frac{1}{\epsilon}$. These parameters result in an online algorithm that is a $(1+\epsilon)$ approximate solution in expectation as compared to the expected optimal value on the distribution for any constant $\epsilon > 0$. (Theorem 8)*

**(ii)** *(Instance Robustness) Using the optimal parameters for any instance on a nearby instance gives a competitive ratio for fractional assignment that is proportional to their distance, where the distance is proportional to the relative difference of job sizes of the same type. (Theorem 7)*

**(iii)** *(Worst-Case Robustness) The competitive ratio of the algorithm using the parameters is never worse than $O(\log m)$, matching the known $\Omega(\log m)$ lower-bound on any integral online algorithm. (Theorem 7)*

This theorem shows that the predictions of [27] have much stronger properties than what is known and are learnable and instance robust. That paper left open the question if their predictions can be formally learned in any model. Moreover, it was not known if they are instance robust. We remark that this theorem assumes fractional assignments, whereas the original problem (and the lower bound [8]) requires integer assignments. Lattanzi et al. [27] shows that any fractional assignment can be rounded online while losing a $O((\log \log m)^3)$ factor in the makespan.

## 1.2 Related Work

**Algorithms with Predictions.** In this paper, we consider augmenting the standard model of online algorithms with erroneous predictions. Several online problems have been studied in this context, including caching [28, 35, 23, 37], page migration [22], metrical task systems [6], ski rental [34, 19, 3], scheduling [34], load balancing [27], online linear optimization [13], speed scaling [38], set cover [39], and bipartite matching and secretary problems [7].

Antoniadis et al. [7] studies online weighted bipartite matching problems with predictions. The main aspect of this work which distinguishes it from ours is that it considers the random order arrival model, rather than adversarial orders.

Mahdian et al. [29] focuses on the design of robust algorithms. Rather than considering online algorithms which use a prediction, they consider two black box online algorithms, one optimistic and the other pessimistic. The goal is to give an online algorithm which never performs much worse than the better of these two algorithms for any given instance. This is shown for problems such as load balancing, facility location, and ad allocation.

The predictions utilized in our algorithm come in the form of vertex weights that guide a proportional allocation scheme. Agrawal et al. [2] first studied proportional allocations for maximum cardinality fractional matching as well as weighted fractional matchings with high entropy. Lattanzi et al. [27] utilize similar predictions based on proportional weights to give algorithms with predictions for online load balancing.

**Data-Driven Algorithm Design.** This paper considers the learnability of the predictions through the model of data-driven algorithms. In classical algorithm design, the main desire is finding an algorithm that performs well in the worst case against all inputs for some measure of performance, e.g. running time or space usage. Data-driven algorithm design [21, 9, 11, 10, 12, 15], in contrast, wants to find an algorithm that performs well on the instances that the user is typically going to see in practice. This is usually formalized by fixing a class of algorithms and an unknown distribution over instances, capturing the idea that some (possibly worst case) instances are unlikely to be seen in practice. The typical question asked is: how many sample instances are needed to guarantee you have found the best algorithm for your application domain?

**Other Related Work.** Online matching and related allocation problems have been extensively studied in both the adversarial arrival setting [25, 24, 17, 30] and with stochastic arrivals [16, 18, 20, 33, 1]. A related but different setting to ours is the online algorithms with advice setting [14]. Here the algorithm has access to an oracle which knows the offline input. The oracle is allowed to communicate information to the algorithm about the full input, and the goal is to understand how many bits of information are necessary to achieve a certain competitive ratio. This has also been extended to the case where the advice can be arbitrarily wrong [4]. This can be seen as similar to our model, however the emphasis isn't on tying the competitive ratio to the amount of error in the advice.

## 2 Algorithms with Learnable and Instance-Robust Predictions

**Learnability via Sample Complexity.** We consider the following setup inspired by PAC learning and recently considered in data-driven algorithms. Assume a maximization problem and let $\mathcal{D}$ be an unknown distribution over problem instances. Let $\mathrm{ALG}(\mathcal{I}, y)$ be the performance[5] of an algorithm using parameters $y$ on instance $\mathcal{I}$. The ideal prediction for this distribution is then $y^* := \arg\max_y \mathbb{E}_{\mathcal{I}\sim\mathcal{D}}[\mathrm{ALG}(\mathcal{I}, y)]$. Since we assume that $\mathcal{D}$ is unknown, we wish to learn from samples. In particular, we wish to use some number $s$ of independent samples to compute a parameter $\hat{y}$ such that $\mathbb{E}_{\mathcal{I}\sim\mathcal{D}}[\mathrm{ALG}(\mathcal{I}, \hat{y})] \geq (1-\epsilon)\mathbb{E}_{\mathcal{I}\sim\mathcal{D}}[\mathrm{ALG}(\mathcal{I}, y^*)]$

[5] In general this can be any performance metric, such as running time or solution value. Here we focus on the value of some objective function such as the size of a fractional flow.

with probability $1-\delta$, for any $\epsilon, \delta \in (0,1)$. The sample complexity $s$ depends on the problem size as well as $1/\epsilon$ and $1/\delta$. As is standard in learning theory, we require the sample complexity to be polynomial in these parameters[6].

Inspired by competitive analysis, we also compare to the following stronger benchmark in this paper. For any instance $\mathcal{I}$, let $\mathrm{OPT}(\mathcal{I})$ be the value of an optimal solution on $\mathcal{I}$. We give learning algorithms producing predicted parameters $\hat{y}$ such that $\mathbb{E}_{\mathcal{I}\sim\mathcal{D}}[\mathrm{ALG}(\mathcal{I},\hat{y})] \geq (1-\epsilon)\mathbb{E}_{\mathcal{I}\sim\mathcal{D}}[\mathrm{OPT}(\mathcal{I})]$ and polynomial sample complexity under the assumptions on $\mathcal{D}$ described earlier. Note that this guarantee implies the first one.

**Instance Robustness.** Let $\mathcal{I}$ and $\mathcal{I}'$ be two problem instances, and consider running the algorithm on instance $\mathcal{I}'$ with the prediction $y^*(\mathcal{I})$. We bound the performance of the algorithm as a function of the difference between these two instances. In contrast, prior work focuses on differences in the predicted parameters $y^*$ and $y'$ for the same instance $\mathcal{I}$. Moreover, it is desirable that the algorithm never performs worse than the best worst-case algorithm.

For example, in online flow allocation, we can consider an instance as a vector of types, i.e. $\mathcal{I}_i$ is the number of impressions of type $i$. Then we can take the difference between the instances as $\gamma = \|\mathcal{I}-\mathcal{I}'\|_1$. Say $y^*(\mathcal{I})$ can be used to give a $c$-competitive algorithm on instance $\mathcal{I}$. Let $\alpha$ be the best competitive ratio achievable in the worst-case model. We desire an algorithm that is $\max\{f(c,\gamma),\alpha\}$-competitive where $f$ is a monotonic function depending on $c$ and $\gamma$.We remark that the online model requires $\mathcal{I}'$ to arrive in a worst-case order.

## 2.1 Putting the Model in Context

**Relationship to Prior Predictions Model.** The first main difference in this model as compared to prior work is *learnability.* With the notable exception of [3], prior work has introduced predictions without establishing they are learnable. Without this requirement there is no objective metric to detail if a prediction is reasonable or not. To see this shortcoming, imagine simply predicting the optimal solution for the problem instance. This is often not reasonable because the optimal solution is too complex to learn and use as a prediction. We introduce bounded sample complexity for learning predictions in our model to ensure predictions can be provably learned.

Next difference is in how to measure error. The performance of the algorithm is bounded in terms of the error in the prediction in the prior model. For example, say the algorithm is given a predicted vector $\hat{y}(\mathcal{I})$ for problem instance $\mathcal{I}$ and the true vector that should have been predicted is $y^*(\mathcal{I})$. One can define $\eta_{\hat{y}(\mathcal{I})} = \|\hat{y}(\mathcal{I}) - y^*(\mathcal{I})\|_p$ to be the *error* in the parameters for some norm $p \geq 1$. The goal is to give an algorithm that is $f(\eta_{\hat{y}(\mathcal{I})})$-competitive for an online algorithm where $f$ is some non-decreasing function of $\eta_{\hat{y}(\mathcal{I})}$: the better the function $f$, the better the algorithm performance. One could also consider run time or approximation ratio similarly. Notice the bound is worst-case for a given error in the prediction. This we call *parameter robustness.*

It is perhaps more natural to define a difference between two problem instances as in our model rather than the difference between two predicted parameters. Indeed, consider predicting optimal dual linear program values. These values can be different for problem instances that are nearly identical. Therefore, accurate parameters will not be sufficient to handle inconsequential changes in the input. Instance robustness allows for more accurate

[6] For more difficult problems, we can relax the $1-\epsilon$ requirement to be a weaker factor.

**Table 1** Relationship to Prior Work.

| Problem | Parameter Robustness | Learnability | Instance Robustness |
|---|---|---|---|
| Caching | [28, 35, 23, 37] | - | - |
| Completion Time Scheduling | [34] | - | [34] |
| Ski Rental | [34, 3, 19] | [3] | [34, 3, 19] |
| Restricted Assignment | [27] | This Paper | This Paper |
| $b$-Matching | This Paper | This Paper | This Paper |
| Flow Allocation | This Paper | This Paper | This Paper |

comparison of two predictions on similar problem instances. More practically, instance closeness is easier to monitor than closeness of the proposed predictions to an unknown optimal prediction for the whole instance.

**Learning Algorithm Parameters.** Learning algorithmic parameters has distinct advantages over learning an input distribution. In many cases it can be easier to learn a decision rule than it is to learn a distribution. For example, consider the unweighted $b$-matching problem in bipartite graphs for large $b$ in the online setting. In this problem there is a bipartite graph $G = (I \cup A, E)$ with capacities $b \in \mathbb{Z}_+^A$. The objective is to find a collection of edges such that each node in $I$ is matched at most once and each node $a \in A$ is matched at most $b_a$ times. Nodes on one side of the graph arrive online and must be matched on arrival. Say the nodes are i.i.d. sampled from an unknown discrete distribution over types. A type is defined by the neighbors of the node. Let $s$ be the number of types. Then the sample complexity of learning the distribution is $\Omega(s)$. Notice that $s$ could be superlinear in the number of nodes. In our results, the sample complexity is independent of the number of types in the support of the distribution. The phenomenon that it is sometimes easier to learn good algorithmic parameters rather than the underlying input distribution has been observed in several prior works. See [21, 9, 11, 10, 12, 15] for examples.

Table 1 illustrates how our paper relates to prior work, focusing on the two pillars for augmenting algorithms emphasized in our model.

**Paper Organization.** In this extended abstract, we give a technical overview of our results. For online flow allocation, both learnability and instance-robustness rely on showing the existence of node predictions which is described in Section 3, followed by learnability and robustness in Sections 4 and 5 respectively. Section 6 considers online load balancing and gives a brief technical overview. All proofs of our results are available in the full version of this paper.

## 3 Matchings and Flows: Existence of Weights

Consider a directed acyclic graph $G = (\{s, t\} \cup V, E)$, where each vertex $v \in V$ has capacity $C_v$ and is on some $s - t$ path. Our goal is to maximize the flow sent from $s$ to $t$ without violating vertex capacities. Before considering the general version, we examine the 3-layered version. Say a graph is $d$-layered if the vertices excluding $s, t$ can be partitioned into $d$ ordered sets where all arcs go from one set to the next. Then the 3-layered case is defined as follows. The vertices in $V$ are partitioned into 3 sets $I, A$, and $B$. $s$ is connected to all of $I$ and $t$ is connected from all of $B$, while the remaining edges are only allowed to cross from $I$ to $A$ and from $A$ to $B$. Let $N_u := \{v \in V \mid (u, v) \in E\}$ be $u$'s out-neighbors. We have the following result generalizing the prior work of Agrawal et al [2] on 2-layered graphs.

▶ **Theorem 3.** *Let $G = (\{s,t\} \cup V, E)$ be a 3-layered DAG. For each edge $(u,v) \in E$, let $x_{uv}$ be the proportion of flow through $u$ which is routed along $(u,v)$. For any $\epsilon \in (0,1)$, there exist weights $\{\alpha_v\}_{v \in V}$ such that setting $x_{uv} = \frac{\alpha_v}{\sum_{v' \in N_u} \alpha_{v'}}$ yields a $(1-\epsilon)$-approximate flow. Moreover, these weights can be obtained in time $O(n^4 \log(n/\epsilon)/\epsilon^2)$.*

We can generalize this theorem to $d$-layered graphs. In particular, our algorithm for the $d$-layered case produces weights with additional properties, which we leverage to handle general DAGs. Notice that the number of weights is proportional to the number of nodes in the graph and not the number of edges. We believe it is an interesting combinatorial property that such succinct weights on the nodes can encode a good flow on the asymptotically larger number of edges and is of independent interest.

## Technical Overview

Here we give a technical overview. The full proof is omitted in this version.

**A Simple Algorithm for Layered Graphs.** Prior work [2] showed that there exists a set of weights giving nearly the same guarantees we show, but only for bipartite graphs. The existence of such weights can be generalized to $d$-layer graphs easily as follows. First find an (optimal) maximum flow $f$. For each vertex $v$, let $f(v)$ be the flow going through $v$. Reset the vertex capacity of $v$ to be $f(v)$. For each pair of adjacent layers find the weights between the two layers independently using the algorithm of [2], treating nodes $v$ on the left hand side as $f(v)$ individual impressions. By the previous result, each layer only loses a negligible portion of the total flow which can be compounded to yield a low loss for these set of weights.

The above reduction does not generalize to general DAGs. One can arrange a DAG into layers, but there are fundamental algorithmic challenges with constructing weights that arise when edges cross layers. One of this paper's algorithmic contributions is showing how to construct such weights for general DAGs. Moreover, as an intermediate result, we show how to compute the weights directly extending the approach of [2] for multi-layer graphs without first solving a flow problem optimally as in the above reduction.

**Finding Weights for Bipartite Graphs.** We begin by first simplifying the algorithm of [2] for bipartite graphs. Let $G = (\{s,t\} \cup I \cup A, E)$ be such a graph. In this case, the fraction of flow $u \in I$ sends to $v \in A$ simplifies to $x_{uv} = \frac{\alpha_v}{\sum_{v' \in N_u} \alpha_{v'}}$ where $\{\alpha\}_{v \in I \cup A}$ are the set of weights. Initially all of the weights are 1 for a vertex $a \in A$. Some of the nodes receive more flow than their capacity in this initial proportional allocation according to the weights. We say a node for which the current proportional allocation of flow exceeds its capacity by a $1+\epsilon$ factor is *overallocated.* In an iteration, the algorithm decreases the weights of these nodes by a $1+\epsilon$ factor.[7] After this process continues for a poly-logarithmic number of iterations, we will be able to show the resulting weights result in a near optimal flow.

To prove that the final weights are near optimal, we show that the weights can be directly used to identify a vertex cut whose value matches the proportional flow given by the weights. In particular, we will partition the nodes in $A$ based on their weight values. For a parameter $\beta$, we say a node is "above the gap" if its weight is larger than $\beta n/\epsilon$. A node of weight less

[7] Prior work [2] performed this operation as well as *increasing* the weights of nodes whose allocation was significantly below the capacity. Our simplification to allow only decreases helps with the generalization to more complex graphs and correcting for error in the weights.

than $\beta$ is below the gap. All others are in the gap. The parameter $\beta$ is chosen such that the nodes in the gap contribute little to the overall flow and they can essentially be discarded via an averaging argument[8]. Assume this set is empty for simplicity. Let $\mathcal{G}(A)^+$ and $\mathcal{G}(A)^-$ be the sets of vertices in $A$ above and below the gap, respectively.

We now describe a cut. Let $I_0 \subseteq I$ be the impression nodes adjacent to at least one node in $\mathcal{G}(A)^+$. Then the vertex cut is $I_0 \cup \mathcal{G}(A)^-$. Since all paths must either cross $I_0$ or $\mathcal{G}(A)^-$, this is a valid vertex cut. We now show the cut value is close to the flow achieved by the weights, completing the analysis using the weaker direction of the max-flow min-cut theorem.

First, nodes in $I_0$ are cut. Due to the way flow is sent based on the weight proportions, for any vertex $I_0$, at least a $(1-\epsilon)$ proportion of its flow will be sent to $\mathcal{G}(A)^+$. Since the nodes in $\mathcal{G}(A)^+$ did not decrease the weights at least once, at some point they were not over-allocated. We claim that because of this, they will never be over-allocated hence and, therefore, nodes in $I_0$ send nearly all of their flow to the sink successfully. Next nodes in $\mathcal{G}(A)^-$ are cut. These nodes decreased their weights (almost) every iteration because they are at or above their allocation. Thus, for all these nodes we get flow equal to their total capacity. The fraction of this flow through $\mathcal{G}(A)^-$ coming from paths using $I_0$ is negligible because of the weight proportions so this flow is almost disjoint from that of $I_0$. Thus, we have found a proportional flow nearly matching the value of the cut identified.

**General Graphs.** Now we consider the more general algorithm. To convey intuition, we will only consider directly computing weights for a 3-layered graph $G = (\{s\} \cup I \cup A \cup B \cup \{t\}, E)$ where edges are between adjacent layers. This will highlight several of the new ideas. As before, weights of all nodes are initially one. And as before, a node decreases its weight if it is over-allocated, which we will refer now to as a *self-decrease*. Now though, whenever a node in $A$ decreases its weight it does so by a $(1+\epsilon')$ factor and those in $B$ decrease at a $(1+\epsilon)$ factor where $\epsilon' \leq \epsilon$.

A new challenge is that a node $b$ in layer $B$ may be over allocated and it may not be enough for $B$ to reduce its weight. Indeed, $B$ may need some neighbors in $A$ to reduce their allocation. For instance, if $b \in B$ has neighbors in $A$ for which it is the only neighbor, then reducing $b$'s weight does not change its allocation and the flow needs to be redistributed in the first layer. In this case, the nodes in $B$ will specify that some nodes in $A$ need to decrease their allocation. We call this a *forced decrease*. This set has to be carefully chosen and intuitively only the nodes in $A$ that are the largest weight as compared to $b \in B$ are decreased. We run this procedure for a polylogarithmic number of steps and again we seek to find a cut matching the achieved flow.

We discuss the need for different $\epsilon$ and $\epsilon'$. In the bipartite case when a node decreased its weight, that node is guaranteed to receive no more allocation in the next round (it could remain the same though). Intuitively, this is important because in the above proof for bipartite graphs we want that if a node is in $\mathcal{G}(A)^+$, above the gap, if it was ever under-allocated then it never becomes over-allocated in later iterations. Our update ensures this will be the case since self-decreases will continue henceforth to keep the load of such a node below its capacity. Consider setting $\epsilon = \epsilon'$ for illustration. Because of the interaction between layers, a node $i$ in $B$ could receive more allocation even if it decreases its weight in an iteration. This is because the nodes in $A$ and $B$ could change their weights. Nodes in $A$ changing their weight can give up to an extra $(1+\epsilon)$ allocation (via predecessors of $i$ that are

[8] This averaging is what necessitates the poly-logarithmic number of weight update iterations in the algorithm

not decreased), and the same for $B$ for a total of $(1+\epsilon)^2$ extra allocation arriving at $i$. The node decreasing its weight reduces its allocation by a $(1+\epsilon)$ factor for a total change of a $(1+\epsilon)^2 \cdot \frac{1}{(1+\epsilon)} > 1$ factor. By choosing $\epsilon$ and $\epsilon'$ to be different, as well as the characterization of which nodes decrease during a forced decrease, we can show any node will not receive less allocation if its weight does not decrease and will not receive more allocation if it performs a decrease. We call these properties "Increasing monotonicity" and "Decreasing monotonicity" in the full version of this paper.

As in the bipartite case, we can find a gap in layers $A$ and $B$, which gives sets above the gap $\mathcal{G}(A)^+$ and $\mathcal{G}(B)^+$ in $A$ and $B$, respectively. Let the sets $\mathcal{G}(A)^-$ and $\mathcal{G}(B)^-$ be the nodes below the gap. As before nodes in $\mathcal{G}(A)^-$ (resp., $\mathcal{G}(B)^-$) decreased their weight many more iterations than $\mathcal{G}(A)^+$ (resp. , $\mathcal{G}(B)^+$,). For simplicity, assume this partitions the nodes of the entire graph, so no nodes are inside either gap. Let $I_0 \subseteq I$ be the nodes adjacent to at least one node in $\mathcal{G}(A)^+$. Let $A_0$ be nodes in $\mathcal{G}(A)^-$, below the gap, that have an edge to $\mathcal{G}(B)^+$ (the analogue of $I_0$ in the $A$-layer) . Any flow path that crosses the $A$ layer at $\mathcal{G}(A)^+, A_0$ and $\mathcal{G}(A)^- \setminus A_0$ are blocked by the sets $I_0, A_0$ and $\mathcal{G}(B)^-$ respectively showing that this set forms a vertex cut.

We show the flow obtained is nearly the value of the vertex cut $I_0 \cup A_0 \cup \mathcal{G}(B)^-$. As before, nodes in $I_0$ (resp. $A_0$) send almost all their flow to nodes above the gap in the next layer $\mathcal{G}(A)^+$ (resp. $\mathcal{G}(B)^+$). Like before $\mathcal{G}(A)^+$ and $\mathcal{G}(B)^+$ do not decrease every round, so we can show they are not allocated more than their capacity (see Lemma 11 and Lemma 13 in this paper's full version). The algorithmic key is that, by choosing the forced decrease carefully, we can show each node in $\mathcal{G}(A)^+$ has a neighbor in $\mathcal{G}(B)^+$. This ensures almost all of $\mathcal{G}(A)^+$'s flow reaches the sink because all of these nodes will send essentially all their the flow to $\mathcal{G}(B)^+$ and these nodes are not at capacity. Thus, $I_0$ can send all its flow to the sink successfully. Similarly, $A_0$ sends its flow to $\mathcal{G}(B)^+$ and then to the sink. Finally, as before, $\mathcal{G}(B)^-$ (as well as $\mathcal{G}(A)^-$) are sets of nodes near their allocation because they decreased essentially every iteration (see Lemma 12 and Lemma 14 in the full version). Thus, $\mathcal{G}(B)^-$ sends its flow directly to the sink. Moreover, we ensure that only a negligible fraction of this flow is double counted by the definition of the large weight reduction across the gaps (see Lemma 15 in the full version); therefore we have found a flow allocation obtained by the weights whose value nearly matches the value of an identified cut.

This analysis generalizes to layered DAGs where edges do not cross layers. To extend the existence of these weights to general DAGs we reduce the problem to finding weights with additional structural properties on a layered DAG. From the input DAG we make additional copies of nodes that have ancestors in earlier layers and link these copies via a path to the original neighbor. We convert any edge that crosses many layers to one in the layered DAG from its original head node to the copy of its tail node in the next layer. Then we argue that a key functional relation exists between the weights of any original node and its copies in earlier layers. This allows us to transfer the weights computed in the auxiliary layered DAG to the original DAG.

## 4 Matchings and Flows: Learnability of Predictions

We show that the weights are efficiently learnable. Assuming that each arriving impression is i.i.d. sampled from an unknown distribution, we want to learn a set of weights from a collection of past instances and examine their expected performance on a new instance from the same distribution[9]. A direct approach might be to learn the unknown distribution from samples and utilize known ideas from stochastic optimization (where knowledge of the distribution is key). A major issue though is that there can be a large number of possible types (potentially exponential in the number of nodes of the DAG). A distribution that is sparse over types is not easy to learn with a small number of samples.

We claim that the weights are efficiently learnable, even if the distribution of types of impressions is not. We show that this task has low sample complexity and admits an efficient learning algorithm. Consequently, if there is an unknown arbitrary distribution that the impressions are drawn from, then only a small number of instances is required to compute the weights. The number of samples is proportional to size of the DAG without the impressions. In most problems such as Adwords, the number of arriving impressions is much larger than the fixed (offline) portion of the graph.

Before stating our results, we introduce two necessary assumptions. The first assumption is that each impression is i.i.d. sampled from an unknown distribution $\mathcal{D}$. Where no ambiguity will result, we also say an instance is sampled from $\mathcal{D}$ if each impression is an i.i.d sample from $\mathcal{D}$. The second assumption is related to the expected instance of the distribution $\mathcal{D}$. The **expected instance** of a distribution is the instance where the number of each type of impressions is exactly the expected value.[10] We assume that in the optimal solution of the expected instance, the load of each node is larger than a constant. Namely, it cannot happen that in the optimal flow, there exist many vertices which obtain very small amount of flow.

▶ **Theorem 4.** *Under the two assumptions above, for any $\epsilon, \delta \in (0, 1)$, there exists a learning algorithm such that, after observing $O(\frac{n^2}{\epsilon^2}\ln(\frac{n\log n}{\delta}))$ instances, returns weights $\{\hat{\alpha}\}$, satisfying that with probability at least $1-\delta$, $\mathbb{E}_{I\sim\mathcal{D}}[R(\hat{\alpha}, I)] \geq (1-\epsilon)\mathbb{E}_{I\sim\mathcal{D}}[R(\alpha^*, I)]$ where $R(\alpha, I)$ is the value of the fractional flow obtained by applying $\alpha$ to instance $I$ and $\alpha^* = \arg\max_{\alpha} \mathbb{E}_{I\sim\mathcal{D}}[R(\alpha, I)]$.*

**Technical Overview.** Here we overview the analysis. The full proof is omitted in this version. To show that the weights are learnable we utilize a model similar to that of data-driven algorithm design. To illustrate our techniques we focus on the case when the instance is a bipartite graph $G = (I \cup A, E)$ with capacities $C_a$ for each $a \in A$ (also recall that $|A| = n$).

In this setting there is an unknown distribution $\mathcal{D}$ over instances of $I$ of length $m$. The $t$'th entry of $I$ represents the $t$'th impression arriving online. Our goal is to find a set of weights that performs well for the distribution $\mathcal{D}$. In particular let $\alpha^* := \arg\max_{\alpha\in\mathcal{S}} \mathbb{E}_{I\sim\mathcal{D}}[R(\alpha, I)]$ be the *best* set of weights for the distribution. Define $R(\alpha, I)$ to be the value of the matching using weights $\alpha$ on instance $I$. Here $\mathcal{S}$ is a set of "admissible" weights, and in particular we only consider weights output by a proportional algorithm similar to the algorithm of Agrawal et al. [2]. We are allowed to sample $s$ independent samples $I_1, I_2, \ldots, I_s$ from $\mathcal{D}$ and use these samples to compute a set of weights $\hat{\alpha}$. We say that a learning algorithm $(\epsilon, \delta)$-learns the weights if with probability at least $1-\delta$ over the samples from $\mathcal{D}$, we compute a set of weights $\hat{\alpha}$ satisfying $\mathbb{E}_{I\sim\mathcal{D}}[R(\hat{\alpha}, I)] \geq (1-\epsilon)\mathbb{E}_{I\sim\mathcal{D}}[R(\alpha^*, I)]$.

This definition is similar to PAC learning [36]. Also note we are aiming for a relative error guarantee rather than an absolute error. The main quantity of interest is then the *sample complexity*, i.e. how large does $s$ need to be as a function of $n$, $m$, $\epsilon$, and $\delta$ in order to $(\epsilon, \delta)$-learn a set of weights? Ideally, $s$ only depends polynomially on $n$, $m$, $1/\epsilon$, and $1/\delta$, and smaller is always better.

[9] We can also analyze the performance on similar distributions by applying techniques from our instance robustness result

[10] Note this could be a fractional value.

The standard way to understand the sample complexity for this type of problem is via the *pseudo-dimension*. Intuitively, pseudo-dimension is the natural extension of VC-dimension to a class of real valued functions. In our case the class of functions is $\{R(\alpha, \cdot) \mid \alpha \in \mathcal{S}\}$, i.e. we are interested in the class of function mapping each instance $I$ to the value of the fractional matching given by each fixed set of weights $\alpha$. If the pseudo-dimension of this class of functions is $d$, then $s \approx \frac{d}{\epsilon^2}\log(1/\delta)$ samples are needed to $(\epsilon, \delta)$ learn the weights, given that we are able to approximately optimize the empirical *average* performance [5].

The good news for our setting is that the pseudo-dimension of our class of functions is bounded. Each node in the set $A$ can only have one of $T$ different weight values for some parameter $T$. Then since the number of nodes in $A$ is $n$, there can only be at most $T^n$ different "admissible" weights. It is well known that the pseudo-dimension of a finite class of $k$ different functions is $\log_2(k)$. Thus the pseudo-dimension of our class of functions is $d = n\log_2(T)$. As long as $T$ isn't growing too fast as a function of $n$ and $m$, we get polynomial sample complexity.

Unfortunately, finding weights to optimize the average performance across the $s$ sampled instances is complicated. Note that for a fixed instance $I$, the value of the matching as a function of the weights, $R(\cdot, I)$, is non-linear in the weights since we are using proportional allocation. Moreover, it is neither convex nor concave in the parameters $\alpha$ so applying a gradient descent approach will not work. Due to this, it is difficult to analyze the learnability via known results on pseudo-dimension.

The main tool we have at our disposal is that for a fixed instance $I$, we can compute weights $\alpha$ such that $R(\alpha, I) \geq (1-\epsilon)\mathrm{OPT}(I)$. This motivates the following natural direct approach. Take the $s$ sampled instances $I_1, I_2, \ldots, I_s$ and take their union to form a larger "stacked" instance $\hat{I}$. We then run the aforementioned algorithm on $\hat{I}$ to get weights $\hat{\alpha}$. Intuitively, if $s$ is large enough, then by standard concentration inequalities $\hat{I} \approx s\mathbb{E}[I]$, i.e. the stacked instance approaches $s$ copies of the "expected" instance. Then to complete the analysis, we need to show that $\mathbb{E}[R(\hat{\alpha}, I)] \approx R(\hat{\alpha}, \mathbb{E}[I])$. In general, it is not true that $\mathbb{E}[R(\hat{\alpha}, I)] \approx R(\hat{\alpha}, \mathbb{E}[I])$. Using more careful analysis, we show that when the distribution $\mathcal{D}$ is a product distribution and our two assumptions hold, this is in fact the case.

## 5 Matching and Flows: Robustness

**Instance Robustness.** To show the instance robustness, we assume that we can describe the instance directly. Say we have a description of the entire instance denoted by a vector $\hat{I} = (\hat{m}_1, \ldots, \hat{m}_i, \ldots)$, where $\hat{m}_i$ is the number of impressions of type $i$. We show that if a set of weights performs well in instance $\hat{I}$, it can be transferred to a nearby instance $I$ robustly.

▶ **Theorem 5.** *For any $\epsilon > 0$, if a set of weights $\hat{\alpha}$ returns a $(1-\epsilon)$-approximated solution in instance $\hat{I}$, it yields an online flow allocation on instance $I$ whose value is at least* $\max\{(1-\epsilon)OPT - 2\gamma, OPT/(d+1)\}$. *Here $OPT$ is the maximum flow value on instance $I$, $d$ is the diameter of this graph excluding vertex $t$, and $\gamma$ is the difference between two instances, defined by $||\hat{I} - I||_1$.*

This theorem can be interpreted as follows. If we sample the instance and compute the weights in it, these weights will work well and break through worst-case bounds when the type proportions are sampled well. Indeed, the weights will perform well in nearby instances with similar type proportions. Moreover, the algorithm never performs worse than a $\frac{1}{d+1}$ factor of optimal. We remark that this is the best competitive ratio a deterministic integral algorithm can achieve because we show a lower bound on such algorithms (see Appendix G.2

in the full version of this paper). This example builds a recursive version of the simple lower bound of $\frac{1}{2}$ on the competitive ratio of deterministic online algorithms for the matching problem.

Recall that the key to showing existence of the weights was to construct a cut whose capacity is nearly the same as the value of the flow given by the weights. To show robustness against nearby instances, we observe how this proof can be extended to nearby cuts. This allows us to argue about the optimal value of the new instance. Standard calculations then let us connect the value of the predicted weights to this optimal value while losing only $O(\gamma)$ in the value of the flow. To ensure the algorithm is never worse than a $\frac{1}{d+1}$ factor of the optimal, we guarantee that the algorithm always returns a maximal allocation.

**Parameter Robustness.** For the parameter robustness, we show that the performance degrades linearly in the relative error of the weight parameter. Thus, our algorithm has the same robustness guarantees shown in other works as well.

▶ **Theorem 6.** *Consider a prediction of $\hat{\alpha}_v$ for each vertex $v \in V$. Due to scale invariance, we can assume that the minimum predicted vertex weight $\hat{\alpha}_{min} = 1$. Define the prediction error $\eta := \max_{v\in V}(\frac{\hat{\alpha}_v}{\alpha^*_v}, \frac{\alpha^*_v}{\hat{\alpha}_v})$, where $\{\alpha^*_v\}_{v\in V}$ are vertex weights that can achieve an $(1-\epsilon)$-approximate solution and $\alpha^*_{min} = 1$ for any fixed $\epsilon > 0$. Employing predictions $\hat{\alpha}$, we can obtain a solution with competitive ratio $\max(\frac{1}{d+1}, \frac{1-\epsilon}{\eta^{2d}})$, where $d$ is the diameter of this graph excluding vertex $t$.*

When the prediction error $\eta$ approaches one, the performance smoothly approaches optimal. While the above theorem involves comparing the propagation of the prediction errors in the performance analysis, we also investigate how inaccurate predictions can be adaptively corrected and improved in the 2-layered Adwords case. For that case, we give an improved algorithm that can correct error in the weights, so that the loss is only $O(\log \eta)$. Moreover, we show that the way we adapt is *tight* and the best possible up to constant factors for any algorithm given predicted weights with error $\eta$.

The parameter robustness follows almost directly from the definition of the proportional assignment given by the weights. In each layer, the over allocation (potentially above the capacity) can be easily bounded by a $\eta^2$ factor, resulting in a loss of at most $\eta^{2d}$ on a $d$ layer graph. We remark that directly using the weights ensures the algorithm is never worse than a $\frac{1}{d+1}$ factor of the optimal solution. The technical proof, along with the weight-adapting technique for the 2-layered case, is present in Appendix G of this paper's full version.

## 6 Results on Load Balancing

Next we show results for restricted assignment load balancing problem in our model. In particular, we study the instance robustness and learnability of proportional weights for this problem. The existence of useful weights and parameter robustness for predicting these weights were shown in prior work [2, 27]. As discussed before, we focus on analyzing fractional assignments.

To describe our results we introduce the following notation. Let $[m]$ denote the set of machines and $S$ denote a set of jobs. Each job $j \in S$ has a size $p_j$ and a neighborhood $N(j)$ of feasible machines. Given a set of positive weights $\{w_i\}_{i\in[m]}$ on the machines, we define a fractional assignment for each job $j$ by setting $x_{ij}(w) = \frac{w_i}{\sum_{i'\in N(j)} w_{i'}}$ for each $i \in N(j)$. Let $\mathrm{ALG}(S, w)$ be the fractional makespan on the jobs in $S$ with weights $w$ and similarly let $\mathrm{OPT}(S)$ be the optimal makespan on the jobs in $S$. Prior work [2, 27] shows that for any set of jobs $S$ and $\epsilon > 0$ there exists weights $\alpha$ such that $\mathrm{ALG}(S, w) \leq (1 + \epsilon)\mathrm{OPT}(S)$.

To describe our instance robustness result we consider an instance of the problem as follows. Consider instances with $n$ jobs. The type of a job is the subset of machines to which it can be assigned. Let $S_j$ be the total size of jobs of type $j$ in instance $S$. We consider relative changes in the instance and define the difference between instances $S$ and $S'$ as $\eta(S, S') := \max_j \max\{\frac{S_j}{S'_j}, \frac{S'_j}{S_j}\}$. Our instance robustness result is given in the following theorem.

▶ **Theorem 7.** *For any instance $S$ and $\epsilon > 0$, let $w$ be weights such that $\text{ALG}(S, w) \leq (1+\epsilon)OPT(S)$. Then for any instance $S'$ we have $\text{ALG}(S', w) \leq (1+\epsilon)^2\eta(S, S')^2 OPT(S')$.*

Let $w$ be as in the statement of the theorem and $w'$ be weights such that $\text{ALG}(S', w') \leq (1+\epsilon)\text{OPT}(S')$. Intuitively, we lose the first factor of $\eta(S, S')$ by bounding the performance of $w$ on $S'$ and the second factor by bounding the performance of $w'$ on $S$.

Next we study learnability. We give the first result showing these weights are learnable in any model. In order to understand the sample complexity of learning the weights we need to consider an appropriate discretization of the space of possible weights. For integer $R > 0$ and $\epsilon > 0$, let $\mathcal{W}(R) = \{\alpha \in \mathbb{R}^m \mid \alpha_i = (1+\epsilon)^k, i \in [m], k \in \{0, 1, \ldots, R\}\}$. Additionally, let $p_{\max}$ be an upper bound on all jobs sizes. The following theorem characterizes the learnability of the weights for restricted assignment load balancing.

▶ **Theorem 8.** *Let $\epsilon, \delta \in (0,1)$ be given and set $R = O(\frac{m^2}{\epsilon^2}\log(\frac{m}{\epsilon}))$ and let $\mathcal{D} = \prod_{j=1}^n \mathcal{D}_j$ be a product distribution over $n$-job restricted assignment instances such that $\mathbb{E}_{S\sim\mathcal{D}}[OPT(S)] \geq \Omega(\frac{1}{\epsilon^2}\log(\frac{m}{\epsilon}))$. There exists an algorithm which finds weights $w \in \mathcal{W}(R)$ such that $\mathbb{E}_{S\sim\mathcal{D}}[\text{ALG}(S, w)] \leq (1+\epsilon)\mathbb{E}_{S\sim\mathcal{D}}[OPT(S)]$with probability at least $1-\delta$ when given access to $s = \tilde{O}(\frac{m^3}{\epsilon^2}\log(\frac{1}{\delta}))$ independent samples $S_1, S_2, \ldots, S_s \sim \mathcal{D}$.*

Our techniques here are similar to that of the online flow allocation problem in that we use the samples to construct a "stacked" instance then compute a set of near optimal weights on this instance. We then have to show that these weights work well in expectation with high probability. This step necessitates the two assumptions in the theorem statement. First, we need that the expected optimal makespan is reasonably large so that the expected makespan is close to the maximum of the expected loads of the machines. Second, we need that the instance is drawn from a product distribution so that the stacked instance converges in some sense to $s$ copies of the "expected" instance. See the full version of the paper for complete arguments.

## References

1 Shipra Agrawal, Zizhuo Wang, and Yinyu Ye. A dynamic near-optimal algorithm for online linear programming. *Oper. Res.*, 62(4):876–890, 2014. `doi:10.1287/opre.2014.1289`.

2 Shipra Agrawal, Morteza Zadimoghaddam, and Vahab Mirrokni. Proportional allocation: Simple, distributed, and diverse matching with high entropy. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 99–108, Stockholmsmässan, Stockholm Sweden, July 2018. PMLR. URL: `http://proceedings.mlr.press/v80/agrawal18b.html`.

3 Keerti Anand, Rong Ge, and Debmalya Panigrahi. Customizing ml predictions for online algorithms. *ICML 2020*, 2020.

4 Spyros Angelopoulos, Christoph Dürr, Shendan Jin, Shahin Kamali, and Marc P. Renault. Online computation with untrusted advice. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle,*

*Washington, USA*, volume 151 of *LIPIcs*, pages 52:1–52:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ITCS.2020.52.

**5** Martin Anthony and Peter L. Bartlett. *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, USA, 1st edition, 2009.

**6** Antonios Antoniadis, Christian Coester, Marek Eliás, Adam Polak, and Bertrand Simon. Online metric algorithms with untrusted predictions. *CoRR*, abs/2003.02144, 2020. arXiv:2003.02144.

**7** Antonios Antoniadis, Themis Gouleakis, Pieter Kleer, and Pavel Kolev. Secretary and online matching problems with machine learned advice. *CoRR*, abs/2006.01026, 2020. arXiv:2006.01026.

**8** Yossi Azar, Joseph Seffi Naor, and Raphael Rom. The competitiveness of on-line assignments. *J. Algorithms*, 18(2):221–237, 1995. doi:10.1006/jagm.1995.1008.

**9** Maria-Florina Balcan, Dan F. DeBlasio, Travis Dick, Carl Kingsford, Tuomas Sandholm, and Ellen Vitercik. How much data is sufficient to learn high-performing algorithms? *CoRR*, abs/1908.02894, 2019. arXiv:1908.02894.

**10** Maria-Florina Balcan, Travis Dick, Tuomas Sandholm, and Ellen Vitercik. Learning to branch. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 353–362. PMLR, 2018. URL: http://proceedings.mlr.press/v80/balcan18a.html.

**11** Maria-Florina Balcan, Travis Dick, and Ellen Vitercik. Dispersion for data-driven algorithm design, online learning, and private optimization. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 603–614. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00064.

**12** Maria-Florina Balcan, Travis Dick, and Colin White. Data-driven clustering via parameterized lloyd's families. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pages 10664–10674, 2018. URL: http://papers.nips.cc/paper/8263-data-driven-clustering-via-parameterized-lloyds-families.

**13** Aditya Bhaskara, Ashok Cutkosky, Ravi Kumar, and Manish Purohit. Online learning with imperfect hints. *CoRR*, abs/2002.04726, 2020. arXiv:2002.04726.

**14** Joan Boyar, Lene M. Favrholdt, Christian Kudahl, Kim S. Larsen, and Jesper W. Mikkelsen. Online algorithms with advice: A survey. *SIGACT News*, 47(3):93–129, 2016. doi:10.1145/2993749.2993766.

**15** Shuchi Chawla, Evangelia Gergatsouli, Yifeng Teng, Christos Tzamos, and Ruimin Zhang. Pandora's box with correlations: Learning and approximation, 2020. arXiv:1911.01632.

**16** Nikhil R. Devanur and Thomas P. Hayes. The adwords problem: online keyword matching with budgeted bidders under random permutations. In *Proceedings 10th ACM Conference on Electronic Commerce (EC-2009), Stanford, California, USA, July 6–10, 2009*, pages 71–78, 2009.

**17** Nikhil R. Devanur, Kamal Jain, and Robert D. Kleinberg. Randomized primal-dual analysis of RANKING for online bipartite matching. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 101–107. SIAM, 2013. doi:10.1137/1.9781611973105.7.

**18** Jon Feldman, Monika Henzinger, Nitish Korula, Vahab S. Mirrokni, and Clifford Stein. Online stochastic packing applied to display ad allocation. In Mark de Berg and Ulrich Meyer, editors, *Algorithms - ESA 2010, 18th Annual European Symposium, Liverpool, UK, September 6-8, 2010. Proceedings, Part I*, volume 6346 of *Lecture Notes in Computer Science*, pages 182–194. Springer, 2010. doi:10.1007/978-3-642-15775-2_16.

19 Sreenivas Gollapudi and Debmalya Panigrahi. Online algorithms for rent-or-buy with expert advice. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 2319–2327. PMLR, 2019. URL: `http://proceedings.mlr.press/v97/gollapudi19a.html`.

20 Anupam Gupta and Marco Molinaro. How the experts algorithm can help solve lps online. *Math. Oper. Res.*, 41(4):1404–1431, 2016. `doi:10.1287/moor.2016.0782`.

21 Rishi Gupta and Tim Roughgarden. A PAC approach to application-specific algorithm selection. *SIAM J. Comput.*, 46(3):992–1017, 2017. `doi:10.1137/15M1050276`.

22 Piotr Indyk, Frederik Mallmann-Trenn, Slobodan Mitrovic, and Ronitt Rubinfeld. Online page migration with ML advice. *CoRR*, abs/2006.05028, 2020. `arXiv:2006.05028`.

23 Zhihao Jiang, Debmalya Panigrahi, and Kevin Sun. Online algorithms for weighted paging with predictions. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 69:1–69:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. `doi:10.4230/LIPIcs.ICALP.2020.69`.

24 Bala Kalyanasundaram and Kirk Pruhs. An optimal deterministic algorithm for online b-matching. *Theor. Comput. Sci.*, 233(1-2):319–325, 2000. `doi:10.1016/S0304-3975(99)00140-1`.

25 Richard M. Karp, Umesh V. Vazirani, and Vijay V. Vazirani. An optimal algorithm for on-line bipartite matching. In Harriet Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 352–358. ACM, 1990. `doi:10.1145/100216.100262`.

26 Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, pages 489–504, New York, NY, USA, 2018. ACM.

27 Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1859–1877. SIAM, 2020. `doi:10.1137/1.9781611975994.114`.

28 Thodoris Lykouris and Sergei Vassilvtiskii. Competitive caching with machine learned advice. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3302–3311, Stockholmsmässan, Stockholm Sweden, July 2018. PMLR. URL: `http://proceedings.mlr.press/v80/lykouris18a.html`.

29 Mohammad Mahdian, Hamid Nazerzadeh, and Amin Saberi. Online optimization with uncertain information. *ACM Trans. Algorithms*, 8(1):2:1–2:29, 2012. `doi:10.1145/2071379.2071381`.

30 Aranyak Mehta, Amin Saberi, Umesh V. Vazirani, and Vijay V. Vazirani. Adwords and generalized online matching. *J. ACM*, 54(5):22, 2007. `doi:10.1145/1284320.1284321`.

31 Michael Mitzenmacher. A model for learned bloom filters and optimizing by sandwiching. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, pages 462–471, 2018.

32 Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions, 2020. `arXiv:2006.09123`.

33 Marco Molinaro and R. Ravi. Geometry of online packing linear programs. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part I*, volume 7391 of *Lecture Notes in Computer Science*, pages 701–713. Springer, 2012. `doi:10.1007/978-3-642-31594-7_59`.

34 Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ML predictions. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, pages 9684–9693, 2018. URL: `http://papers.nips.cc/paper/8174-improving-online-algorithms-via-ml-predictions`.

35 Dhruv Rohatgi. Near-optimal bounds for online caching with machine learned advice. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1834–1845. SIAM, 2020. `doi:10.1137/1.9781611975994.112`.

36 Leslie G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984. `doi:10.1145/1968.1972`.

37 Alexander Wei. Better and simpler learning-augmented online caching. *CoRR*, abs/2005.13716, 2020. `arXiv:2005.13716`.

38 Étienne Bamas, Andreas Maggiori, Lars Rohwedder, and Ola Svensson. Learning augmented energy minimization via speed scaling, 2020. `arXiv:2010.11629`.

39 Étienne Bamas, Andreas Maggiori, and Ola Svensson. The primal-dual method for learning augmented algorithms, 2020. `arXiv:2010.11632`.