Online Metric Algorithms with Untrusted Predictions

Antonios Antoniadis [*1] **Christian Coester** [*2]
Marek Eliáš [*3] **Adam Polak** [*4] **Bertrand Simon** [*5]

Abstract

Machine-learned predictors, although achieving very good results for inputs resembling training data, cannot possibly provide perfect predictions in all situations. Still, decision-making systems that are based on such predictors need not only to benefit from good predictions but also to achieve a decent performance when the predictions are inadequate. In this paper, we propose a prediction setup for arbitrary *metrical task systems (MTS)* (e.g., *caching*, *k-server* and *convex body chasing*) and *online matching on the line*. We utilize results from the theory of online algorithms to show how to make the setup robust. Specifically for caching, we present an algorithm whose performance, as a function of the prediction error, is exponentially better than what is achievable for general MTS. Finally, we present an empirical evaluation of our methods on real world datasets, which suggests practicality.

1. Introduction

Metrical task systems (MTS), introduced by Borodin et al. (1992), are a rich class containing several fundamental problems in online optimization as special cases, including *caching*, *k-server*, *convex body chasing*, and *convex function chasing*. MTS are capable of modeling many problems arising in computing and production systems (Sleator & Tarjan, 1985; Manasse et al., 1990), movements of service vehicles (Dehghani et al., 2017; Coester & Koutsoupias, 2019), power management of embedded systems as well as data centers (Irani et al., 2003; Lin et al., 2013), and are also related to the *experts* problem in online learning (see Daniely & Mansour, 2019; Blum & Burch, 2000).

Initially, we are given a metric space M of *states*, which can be interpreted for example as actions, investment strategies, or configurations of some production machine. We start at a predefined initial state x_0. At each time $t = 1, 2, \ldots$, we are presented with a *cost function* $\ell_t \colon M \to \mathbb{R}^+ \cup \{0, +\infty\}$ and our task is to decide either to stay at x_{t-1} and pay the cost $\ell_t(x_{t-1})$, or to move to some other (possibly cheaper) state x_t and pay $dist(x_{t-1}, x_t) + \ell_t(x_t)$, where $dist(x_{t-1}, x_t)$ is the cost of the transition between states x_{t-1} and x_t. The objective is to minimize the overall cost incurred over time.

Given that MTS is an online problem, one needs to make each decision without any information about the future cost functions. This makes the problem substantially difficult, as supported by strong lower bounds for general MTS (Borodin et al., 1992) as well as for many special MTS problems (see e.g. Karloff et al., 1994; Fiat et al., 1998). For the recent work on MTS, see Bubeck et al. (2019); Coester & Lee (2019); Bubeck & Rabani (2020).

In this paper, we study how to utilize predictors (possibly based on machine learning) in order to decrease the uncertainty about the future and achieve a better performance for MTS. We propose a natural prediction setup for MTS and show how to develop algorithms in this setup with the following properties of *consistency* (i) and *robustness* (ii).

(i) Their performance improves with accuracy of the predictor and is close-to-optimal with perfect predictions.

(ii) When given poor predictions, their performance is comparable to that of the best online algorithm which does not use predictions.

The only MTS that has been studied before in this context of utilizing predictors is the caching problem. Algorithms by Lykouris & Vassilvitskii (2018) and Rohatgi (2020) provide similar guarantees by using predictions about the time of the next occurrence of the current page in the input sequence. However, as we show in this paper, such predictions are not useful for more general MTS, even for weighted caching.

Using the prediction setup proposed in this paper, we can design robust and consistent algorithms for any MTS. For

[*]Equal contribution [1]Saarland University and Max-Planck Institute for Informatics, Saarbrücken, Germany [2]CWI, Amsterdam, Netherlands [3]EPFL, Lausanne, Switzerland [4]Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland [5]University of Bremen, Bremen, Germany. Correspondence to: Antonios Antoniadis <aantonia@mpi-inf.mpg.de>, Christian Coester <christian.coester@cwi.nl>, Marek Eliáš <marek.elias@epfl.ch>, Adam Polak <polak@tcs.uj.edu.pl>, Bertrand Simon <bsimon@uni-bremen.de>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020.

the (unweighted) caching problem, we develop an algorithm that obtains a better dependency on the prediction error than our general result, and whose performance in empirical tests is either better or comparable to the algorithms by Lykouris & Vassilvitskii (2018) and Rohatgi (2020). This demonstrates the flexibility of our setup. We would like to stress that specifically for the caching problem, the predictions in our setup can be obtained by simply converting the predictions used by Lykouris & Vassilvitskii (2018) and Rohatgi (2020), a feature that we use in order to compare our results to those previous algorithms. Nevertheless our prediction setup is applicable to the much broader context of MTS. We demonstrate this and suggest practicability of our algorithms also for MTS other than caching by providing experimental results for the *ice cream* problem (Chrobak & Larmore, 1998), a simple example of an MTS. Finally, we extend our theoretical result beyond MTS to *online matching on the line*.

Prediction Setup for MTS. At each time t, the predictor produces a prediction p_t of the state where the algorithm should be at time t. We define the *prediction error* with respect to some offline algorithm OFF as

$$\eta = \sum_{t=1}^{T} \eta_t; \quad \eta_t = dist(p_t, o_t), \tag{1}$$

where o_t denotes the state of OFF at time t and T denotes the length of the input sequence.

The predictions could be, for instance, the output of a machine-learned model or a heuristic which tends to produce a good solution in practice, but possibly without a theoretical guarantee. The offline algorithm OFF can be an optimal one, but also other options are plausible. For example, if the typical instances are composed of subpatterns known from the past and for which good solutions are known, then we can think of OFF as a near-optimal algorithm which composes its output from the partial solutions to the subpatterns. The task of the predictor in this case is to anticipate which subpattern is going to follow and provide the precomputed solution to that subpattern. In the case of the caching problem, as mentioned above and explained in Section 1.3, we can actually convert the predictions used by Lykouris & Vassilvitskii (2018) and Rohatgi (2020) into predictions for our setup.

Note that, even if the prediction error with respect to OFF is low, the cost of the solution composed from the predictions $p_1, \ldots, p_T$ can be much higher than the cost incurred by OFF, since $\ell_t(p_t)$ can be much larger than $\ell_t(o_t)$ even if $dist(p_t, o_t)$ is small. However, we can design algorithms which use such predictions and achieve a good performance whenever the predictions have small error with respect to any low-cost offline algorithm. We aim at expressing the performance of the prediction-based algorithms as a function of η/OFF, where (abusing notation) OFF denotes the cost of the offline algorithm. This is to avoid scaling issues: if the offline algorithm incurs movement cost 1000, predictions with total error $\eta = 1$ give us a rather precise estimate of its state, unlike when $\text{OFF} = 0.1$.

1.1. Our Results

We prove two general theorems providing robustness and consistency guarantees for any MTS.

Theorem 1. *Let A be a deterministic α-competitive online algorithm for a problem P belonging to MTS. There is a prediction-based deterministic algorithm for P achieving competitive ratio*

$$9 \cdot \min\{\alpha,\ 1 + 4\eta/\text{OFF}\}$$

against any offline algorithm OFF*, where η is the prediction error with respect to* OFF.

Roughly speaking, the competitive ratio (formally defined in Section 2) is the worst case ratio between the cost of two algorithms. If OFF is an optimal algorithm, then the expression in the theorem is the overall competitive ratio of the prediction-based algorithm.

Theorem 2. *Let A be a randomized α-competitive online algorithm for an MTS P with metric space diameter D. For any $\epsilon \leq 1/4$, there is a prediction-based randomized algorithm for P achieving cost below*

$$(1+\epsilon) \cdot \min\{\alpha, 1 + 4\eta/\text{OFF}\} \cdot \text{OFF} + O(D/\epsilon),$$

where η is the prediction error with respect to an offline algorithm OFF*. Thus, if* OFF *is (near-)optimal and $\eta \ll$* OFF*, the competitive ratio is close to $1 + \epsilon$.*

We note that the proofs of these theorems are based on the powerful results by Fiat et al. (1994) and Blum & Burch (2000). In Theorem 9, we show that the dependence on η/OFF in the preceding theorems is tight up to constant factors for some MTS instance.

For some other specific MTS, however, the dependence on η/OFF can be improved. In particular, we present in Section 4 a new algorithm for caching, a special case of MTS, whose competitive ratio has a logarithmic dependence on η/OFF. One of the main characteristics of our algorithm, which we call TRUST&DOUBT, compared to previous approaches, is that it is able to *gradually* adapt the level of trust in the predictor throughout the instance.

Theorem 3. *There is a prediction-based randomized algorithm for (unweighted) caching with a competitive ratio $O(\min\{1 + \log(1 + \frac{\eta}{\text{OFF}}), \log k\})$ against any algorithm* OFF*, where η is the prediction error and k is the cache size.*

Although we designed our prediction setup with MTS in mind, it can also be applied to problems beyond MTS. We demonstrate this by employing our techniques to provide an algorithm of similar flavor for *online matching on the line*, a problem not known to be an MTS.

Theorem 4. *There is a prediction-based deterministic algorithm for online matching on the line which attains a competitive ratio $O(\min\{\log n, 1+\eta/\text{OFF}\})$, where η is the prediction error with respect to some offline algorithm* OFF.

We also show that Theorem 4 can be generalized to give a $O(\min\{2n-1, \eta/\text{OFF}\})$-competitive algorithm for *online metric bipartite matching*.

We show that the predictions used by Lykouris & Vassilvitskii (2018) and Rohatgi (2020) for caching do not help for more general problems like weighted caching:

Theorem 5. *The competitive ratio of any algorithm for weighted caching even if provided with precise predictions of the time of the next request to each page is $\Omega(\log k)$.*

Note that there are $O(\log k)$-competitive online algorithms for weighted caching which do not use any predictions (see Bansal et al., 2012). This motivates the need for a different prediction setup as introduced in this paper.

We round up by presenting an extensive experimental evaluation of our results that suggests practicality. We test the performance of our algorithms on public data with previously used models. With respect to caching, our algorithms outperform all previous approaches in most settings (and are always comparable). A very interesting use of our setup is that it allows us to employ any other online algorithm as a predictor for our algorithm. For instance, when using the Least Recently Used (LRU) algorithm – which is considered the gold standard in practice – as a predictor for our algorithm, our experiments suggest that we achieve the same practical performance as LRU, but with an exponential improvement in the theoretical worst-case guarantee ($O(\log k)$ instead of k). Finally we applied our general algorithms to a simple MTS called the ice cream problem and were able to obtain results that also suggest practicality of our setup beyond caching.

Supplementary Material. Due to space constraints, this paper contains only the intuition behind our results for general MTS (Theorems 1 and 2), the algorithm for caching (Theorem 3), and the results of our empirical experiments. The formal proofs and full details of all our results are included in the supplementary material which contains a copy of the full version of our paper (Antoniadis et al., 2020).

1.2. Related Work

Our work is part of a larger and recent movement to prove rigorous performance guarantees for algorithms based on machine learning. There are already exciting results on this topic in both classical (see Kraska et al., 2018; Khalil et al., 2017) and online problems: Rohatgi (2020) on caching, Lattanzi et al. (2020) on restricted assignment scheduling, Lykouris & Vassilvitskii (2018) on caching, Purohit et al. (2018) on ski rental and non-clairvoyant scheduling, Gollapudi & Panigrahi (2019) on ski rental with multiple predictors, Mitzenmacher (2020) on scheduling/queuing, and Medina & Vassilvitskii (2017) on revenue optimization.

Most of the online results are analyzed by means of *consistency* (competitive-ratio in the case of perfect predictions) and *robustness* (worst-case competitive-ratio regardless of prediction quality), which was first defined in this context by Purohit et al. (2018), while Mitzenmacher (2020) uses a different measure called *price of misprediction*. It should be noted that the exact definitions of consistency and robustness are slightly inconsistent between different works in the literature, making it often difficult to directly compare results.

Results on Caching. The probably closest results to our work are the ones by Lykouris & Vassilvitskii (2018) and Rohatgi (2020), who study the caching problem (a special case of MTS) with machine learned predictions. Lykouris & Vassilvitskii (2018) introduced the following prediction setup for caching: whenever a page is requested, the algorithm receives a prediction of the time when the same page will be requested again. The prediction error is defined as the ℓ_1-distance between the predictions and the truth, i.e., the sum – over all requests – of the absolute difference between the predicted and the real time of the next occurrence of the same request. For this prediction setup, they adapted the classic Marker algorithm in order to achieve, up to constant factors, the best robustness and consistency possible. In particular, they achieved a competitive ratio of $O\big(1+\min\{\sqrt{\eta/\text{OPT}}, \log k\}\big)$ and their algorithm was shown to perform well in experiments. Later, Rohatgi (2020) achieved a better dependency on the prediction error: $O(1+\min\{\frac{\log k}{k}\frac{\eta}{\text{OPT}}, \log k\})$. He also provides a close lower bound.

Following the original announcement of our work, we learned about further developments by Wei (2020) and Jiang et al. (2020). Wei (2020) improves upon the result of Rohatgi (2020), proving a guarantee of $O(1+\min\{\frac{1}{k}\frac{\eta}{\text{OPT}}, \log k\})$ for a robust version of a natural algorithm called Blind Oracle. The paper by Jiang et al. (2020) proposes an algorithm for weighted caching in a very strong prediction setup, where the predictor reports at each time step the time t of the next occurrence of the currently requested page along with all page requests until t. Jiang et al.

(2020) provide a collection of lower bounds for weaker predictors (including an independent proof of Theorem 5), justifying the need for such a strong predictor.

We stress that the aforementioned results use different prediction setups and they do not directly imply any bounds for our setup. This is due to a different way of measuring prediction error, see Section 1.3 for details.

Combining Worst-Case and Optimistic Algorithms. An approach in some ways similar to ours was developed by Mahdian et al. (2012) who assume the existence of an optimistic algorithm and developed a meta-algorithm that combines this algorithm with a classical one and obtains a competitive ratio that is an interpolation between the ratios of the two algorithms. They designed such algorithms for several problems including facility location and load balancing. The competitive ratios obtained depend on the performance of the optimistic algorithm and the choice of the interpolation parameter. Furthermore the meta-algorithm is designed on a problem-by-problem basis. In contrast, (i) our performance guarantees are a function of the prediction error, (ii) generally we are able to approach the performance of the best algorithm, and (iii) our way of simulating multiple algorithms can be seen as a black box and is problem independent.

Online Algorithms with Advice. Another model for augmenting online algorithms, but not directly related to the prediction setting studied in this paper, is that of *advice complexity*, where information about the future is obtained in the form of some always correct bits of advice (see Boyar et al., 2017, for a survey). Emek et al. (2011) considered MTS under advice complexity, and Angelopoulos et al. (2020) consider advice complexity with possibly adversarial advice and focus on Pareto-optimal algorithms for consistency and robustness in several similar online problems.

1.3. Comparison to the Setup of Lykouris&Vassilvitskii

Although the work of Lykouris & Vassilvitskii (2018) for caching served as an inspiration, our prediction setup cannot be understood as an extension or generalization of their setup. Here we list the most important connections and differences.

Conversion of Predictions for Caching. One can convert the predictions of Lykouris & Vassilvitskii (2018) for caching into predictions for our setup using a natural algorithm[1]: At each page fault, evict the page whose next request is predicted furthest in the future. Note that, if given perfect predictions, this algorithm produces an optimal solution (Belady, 1966). The states of this algorithm at each time are then interpreted as predictions in our setup. We use this conversion to compare the performance of our algorithms to those of Lykouris & Vassilvitskii (2018) and Rohatgi (2020) in empirical experiments in Section 5.

Prediction Error. The prediction error as defined by Lykouris & Vassilvitskii (2018) is not directly comparable to ours. Here are two examples.

(1) If we modify the perfect predictions in the setup of Lykouris & Vassilvitskii (2018) by adding 1 to each predicted time of the next occurrence, we get predictions with error $\Omega(T)$, where T is the length of the input sequence (potentially infinite). However, the shift by 1 does not change the order of the next occurrences and the conversion algorithm above will still produce an optimal solution, i.e., the converted predictions will have error 0 with respect to the offline optimum.

(2) One can create a request sequence consisting of $k+1$ distinct pages where swapping two predicted times of next arrivals causes a different prediction to be generated by the conversion algorithm. The modified prediction in the setup of Lykouris & Vassilvitskii (2018) may only have error 2 while the error in our setup with respect to the offline optimum can be arbitrarily high (depending on how far in the future these arrivals happen). However, our results provide meaningful bounds also in this situation. Such predictions still have error 0 in our setup with respect to a near-optimal algorithm which incurs only one additional page fault compared to the offline optimum. Theorems 1–3 then provide constant-competitive algorithms with respect to this near-optimal algorithm.

The first example shows that the results of Lykouris & Vassilvitskii (2018); Rohatgi (2020); Wei (2020) do not imply any bounds in our setup. On the other hand, the recent result of Wei (2020) shows that our algorithms from Theorems 1–3, combined with the prediction-converting algorithm above, are $O(1+\min\{\frac{1}{k}\frac{\eta}{\mathrm{OPT}}, \log k\})$-competitive for caching in the setup of Lykouris & Vassilvitskii (2018), thus also matching the best known competitive ratio in that setup: The output of the conversion algorithm has error 0 with respect to itself and our algorithms are constant-competitive with respect to it. Since the competitive ratio of the conversion algorithm is $O(1+\frac{1}{k}\frac{\eta}{\mathrm{OPT}})$ by Wei (2020), our algorithms are $O(\min\{1+\frac{1}{k}\frac{\eta}{\mathrm{OPT}}, \log k\})$-competitive, where η denotes the prediction error in the setup of Lykouris & Vassilvitskii (2018).

Succinctness. In the case of caching, we can restrict ourselves to *lazy* predictors, where each predicted cache content differs from the previous predicted cache content by at most one page, and only if the previous predicted cache content did not contain the requested page. This is motivated by the fact that any algorithm can be transformed into a lazy

[1] Wei (2020) calls this algorithm Blind Oracle and proves that it is $O(1+\frac{1}{k}\frac{\eta}{\mathrm{OPT}})$-competitive in the setup of Lykouris & Vassilvitskii (2018).

version of itself without increasing its cost. Therefore, it is enough to receive predictions of size $O(\log k)$ per time step saying which page should be evicted, compared to $\Theta(\log T)$ bits needed to encode the next occurrence in the setup of Lykouris & Vassilvitskii (2018). In fact, we need to receive a prediction only for time steps where the current request is not part of the previous cache content of the predictor. In cases when running an ML predictor at each of these time steps is too costly, our setup allows predictions being generated by some fast heuristic whose parameters can be recalculated by the ML algorithm only when needed.

2. Preliminaries

In MTS, we are given a metric space M of states and an initial state $x_0 \in M$. At each time $t = 1, 2, \ldots$, we receive a task $\ell_t \colon M \to \mathbb{R}^+ \cup \{0, +\infty\}$ and we have to choose a new state x_t without knowledge of the future tasks, incurring cost $dist(x_{t-1}, x_t) + \ell_t(x_t)$. Note that $dist(x_{t-1}, x_t) = 0$ if $x_{t-1} = x_t$ by the identity property of metrics.

Although MTS share several similarities with the *experts* problem from the theory of online learning (Freund & Schapire, 1997; Chung, 1994), there are three important differences. First, there is a *switching cost*: we need to pay cost for switching between states equal to their distance in the underlying metric space. Second, an algorithm for MTS has *one-step lookahead*, i.e., it can see the task (or loss function) before choosing the new state and incurring the cost of this task. Third, there can be *unbounded costs* in MTS, which can be handled thanks to the lookahead. See Blum & Burch (2000) for more details on the relation between experts and MTS.

In the caching problem we have a two-level computer memory, out of which the fast one (cache) can only store k pages. We need to answer a sequence of requests to pages. Such a request requires no action and incurs no cost if the page is already in the cache, but otherwise a *page fault* occurs and we have to add the page and evict some other page at a cost of 1. Caching can be seen as an MTS with states being the cache configurations.

To assess the performance of algorithms, we use the *competitive ratio* – the classical measure used in online algorithms.

Definition 1 (Competitive ratio)**.** *Let $\mathcal{A}$ be an online algorithm for some cost-minimization problem P. We say that $\mathcal{A}$ is* r-competitive *and call r the* competitive ratio *of $\mathcal{A}$, if for any input sequence $I \in P$, we have*

$$\mathbb{E}[cost(\mathcal{A}(I))] \leq r \cdot \textsc{Opt}_I + \alpha,$$

where α is a constant independent of the input sequence, $\mathcal{A}(I)$ is the solution produced by the online algorithm and $\textsc{Opt}_I$ is the cost of an optimal solution computed offline with the prior knowledge of the whole input sequence. The expectation is over the randomness in the online algorithm. If $\textsc{Opt}_I$ is replaced by the cost of some specific algorithm Off*, we say that $\mathcal{A}$ is r-competitive against* Off.

2.1. Combining Online Algorithms

Consider m algorithms $A_0, \ldots, A_{m-1}$ for some problem P belonging to MTS. We describe two methods to combine them into one algorithm which achieves a performance guarantee close to the best of them. Note that these methods are also applicable to problems which do not belong to MTS as long as one can simulate all the algorithms at once and bound the cost for switching between them.

Deterministic Combination. The following method was proposed by Fiat et al. (1994) for the k-server problem, but can be generalized to MTS. We note that a similar combination is also mentioned in Lykouris & Vassilvitskii (2018). We simulate the execution of $A_0, \ldots, A_{m-1}$ simultaneously. At each time, we stay in the configuration of one of them, and we switch between the algorithms in the manner of a solution for the m-lane *cow path* problem, see Algorithm 1 for details.

Algorithm 1: MIN^{det} (Fiat et al., 1994)

choose $1 < \gamma \leq 2$; set $\ell := 0$
repeat
  $i := \ell \bmod m$
  while $cost(A_i) \leq \gamma^\ell$, follow A_i
  $\ell := \ell + 1$
until *the end of the input*

Theorem 6 (generalization of Theorem 1 in Fiat et al. (1994))**.** *Given m online algorithms $A_0, \ldots A_{m-1}$ for a problem P in MTS, the algorithm MIN^{det} achieves cost at most $(\frac{2\gamma^m}{\gamma-1} + 1) \cdot \min_i\{cost_{A_i}(I)\}$, for any input sequence I.*

A proof of this theorem can be found in supplementary material. The optimal choice of γ is $\frac{m}{m-1}$. Then $\frac{2\gamma^m}{\gamma-1} + 1$ becomes 9 for $m = 2$, and can be bounded by $2em$ for larger m.

Randomized Combination. Blum & Burch (2000) proposed the following way to combine online algorithms based on the WMR (Littlestone & Warmuth, 1994) (Weighted Majority Randomized) algorithm for the experts problem. At each time t, it maintains a probability distribution p^t over the m algorithms updated using WMR. Let $dist(p^t, p^{t+1}) = \sum_i \max\{0, p_i^t - p_i^{t+1}\}$ be the earth-mover distance between p^t and p^{t+1} and let $\tau_{ij} \geq 0$ be the transfer of the probability mass from p_i^t to p_j^{t+1} certifying this distance, so that $p_i^t = \sum_{j=0}^{m-1} \tau_{ij}$ and $dist(p^t, p^{t+1}) = \sum_{i \neq j} \tau_{ij}$. If we are now following algorithm A_i, we switch to A_j with proba-

bility τ_{ij}/p_i^t. See Algorithm 2 for details. The parameter D is an upper bound on the switching cost states of two algorithms.

Algorithm 2: MIN^{rand} (Blum & Burch, 2000)

$\beta := 1 - \frac{\epsilon}{2}$; `// for parameter` $\epsilon < 1/2$
$w_i^0 := 1$ for each $i = 0, \ldots, m-1$;
foreach *time* t **do**
  $c_i^t :=$ cost incurred by A_i at time t;
  $w_i^{t+1} := w_i^t \cdot \beta^{c_i^t/D}$ and $p_i^{t+1} := \frac{w_i^{t+1}}{\sum w_i^{t+1}}$;
  $\tau_{i,j} :=$ mass transferred from p_i^t to p_j^{t+1};
  switch from A_i to A_j w.p. τ_{ij}/p_i^t;

Theorem 7 (Blum & Burch (2000))**.** *Given m on-line algorithms $A_0, \ldots A_{m-1}$ for an MTS with diameter D and $\epsilon < 1/2$, there is a randomized algorithm MIN^{rand} such that, for any instance I, its expected cost is at most*

$$(1+\epsilon) \cdot \min_i \{cost(A_i(I))\} + O(D/\epsilon) \ln m.$$

3. Robust Algorithms for MTS

The goal of this section is to prove Theorem 1 and Theorem 2. We use algorithms MIN^{det} and MIN^{rand} respectively to combine the online algorithm A with a deterministic algorithm *Follow the Prediction* (FTP) proposed in the following lemma. The proofs then follow by using Theorem 6 and Theorem 7.

Lemma 8. *There is a prediction-based deterministic algorithm* FTP *for any MTS which achieves competitive ratio $1 + \frac{4\eta}{\text{OFF}}$ against any offline algorithm* OFF*, where η is the prediction error with respect to* OFF.

Here, we only describe the FTP algorithm. The proof of the lemma can be found in supplementary material.

Algorithm Follow the Prediction (FTP). Intuitively, our algorithm follows the predictions but still somewhat cautiously: if there exists a state "close" to the predicted one that has a much cheaper service cost, then it is to be preferred. Let us consider a metrical task system with a set of states X. We define the algorithm FTP (Follow the Prediction) as follows: at time t, after receiving task ℓ_t and prediction p_t, it moves to the state

$$x_t \leftarrow \arg\min_{x \in X} \{\ell_t(x) + 2dist(x, p_t)\}. \tag{2}$$

In other words, FTP follows the predictions except when it is beneficial to move from the predicted state to some other state, pay the service and move back to the predicted state.

Lower Bound. Theorem 9 shows that our guarantees in Theorems 1 and 2 are tight up to a constant factor (the proof is included in supplementary material).

Theorem 9. *There exists a Metrical Task System instance in which no randomized (or deterministic) online algorithm using our prediction setup can be better than $\min\{L, 1 + \frac{\eta}{2\,\text{OPT}}\}$-competitive, where L is the optimal competitive ratio of randomized (or deterministic) online algorithms without predictions.*

4. Logarithmic Error Dependence for Caching

We describe in this section[2] a new algorithm for the caching problem, which we call TRUST&DOUBT. It achieves a competitive ratio logarithmic in the error (thus overcoming the lower bound of Theorem 9), while also attaining the optimal worst-case guarantee of $O(\log k)$.

Let r_t be the page that is requested at time t and let P_t be the configuration (i.e., set of pages in the cache) of the predictor at time t. We assume that the predictor is lazy in the sense that P_t differs from P_{t-1} only if $r_t \notin P_{t-1}$ and, in this case, $P_t = P_{t-1} \cup \{r_t\} \setminus \{q\}$ for some page $q \in P_{t-1}$.

The request sequence can be decomposed into maximal time periods (*phases*) where k distinct pages were requested. The first phase begins with the first request. A phase ends (and a new phase begins) after k distinct pages have been requested in the current phase and right before the next arrival of a page that is different from all these k pages. For a given point in time, we say that a page is *marked* if it has been requested at least once in the current phase. For each page p requested in a phase, we call the first request to p in that phase the *arrival* of p. This is the time when p gets marked. Many algorithms, including that of Lykouris & Vassilvitskii (2018), belong to the class of so-called *marking algorithms*, which evict a page only if it is unmarked. In general, no marking algorithm can be better than 2-competitive even when provided with perfect predictions. As will become clear from the definition of TRUST&DOUBT below, it may in some cases evict a page even when it is marked, meaning that it is not a marking algorithm. As can be seen in our experiments in Section 5, this allows TRUST&DOUBT to outperform these other algorithms when predictions are good.

TRUST&DOUBT maintains several sets of pages during its execution: A page is called *ancient* if it is in TRUST&DOUBT's cache even though it has been requested in neither the previous nor the current phase (so far). The set of ancient pages is denoted by A. Whenever there is a page fault and $A \neq \emptyset$, TRUST&DOUBT evicts a page from A. Non-ancient pages that are in TRUST&DOUBT's cache at the beginning of a phase will be called *stale* for the remainder of the phase. A page that is not stale and arrives in a phase *after* A becomes empty will be called *clean*. By C we denote the set of clean pages that have arrived so far in the current phase. TRUST&DOUBT asso-

[2] We provide a pseudocode in the supplementary material.

ciates with each clean page $q \in C$ a page p_q that is missing from the predictor's cache. We also maintain a Boolean variable *trusted*(q) for each $q \in C$, indicating whether TRUST&DOUBT has decided to trust the predictor's advice to evict p_q (a trusted advice may still be wrong though). Denote by $T = \{p_q \mid q \in C, \textit{trusted}(q) = \textit{true}\}$ and $D = \{p_q \mid q \in C, \textit{trusted}(q) = \textit{false}\}$ the sets of these predicted evictions that are currently trusted and doubted, respectively. We will ensure that no page from T is in the algorithm's cache, whereas pages in D may or may not be in the algorithm's cache. Let U be the set of unmarked stale pages that are not in T. Let M be the set of marked pages that are not in T. For each clean page $q \in C$, we also maintain a *threshold* t_q. Roughly speaking, a larger threshold indicates that TRUST&DOUBT is willing to trust the prediction to evict p_q less frequently. We partition the time from the arrival of $q \in C$ until the end of the phase into intervals, which we call *q-intervals*. The first q-interval begins right before the arrival of q. The current q-interval lasts as long as the number of arrivals in this q-interval is at most t_q, and a new q-interval begins once this would stop being the case. As will be specified below, the threshold t_q starts at 1 for each clean page q of the phase and doubles after those intervals in which a request to p_q occurs (i.e., the prediction to evict p_q was ill-advised and hence we increase the threshold). The algorithm trusts to evict p_q at the start of each q-interval, but if p_q is requested during the q-interval, then p_q will be redefined to be a different page and this prediction will be doubted for the remainder of the current q-interval.

As mentioned above, whenever a page fault occurs while $A \neq \emptyset$, TRUST&DOUBT evicts an arbitrary[3] ancient page. Once A becomes empty, we sample a permutation of the pages in U uniformly at random, and define the *priority* of each page in U to be its rank in this permutation. Hereafter, when a page r is requested, TRUST&DOUBT proceeds as follows:

1. If r is not in TRUST&DOUBT's cache, evict the unmarked page with lowest priority and load r.
2. If $r \in C$ and this is the arrival of r, define p_r to be an arbitrary[4] page from $(U \cup M) \setminus D$ that is missing from the predictor's cache and set *trusted*$(r) :=$ *true* and $t_r := 1$.
3. If $r = p_q \in T \cup D$ for some $q \in C$, redefine p_q to be an arbitrary page from $(U \cup M) \setminus D$ that is missing from the predictor's cache, and set *trusted*$(q) :=$ *false*.
4. For each $q \in C$ for which a new q-interval began with this request: If *trusted*$(q) =$ *false*, set $t_q := 2t_q$ and *trusted*$(q) :=$ *true*. If p_q is in TRUST&DOUBT's cache, evict p_q and, of the pages in U that are missing from the cache, load the one with highest priority.

[3] e.g., prioritize those missing from the predictor's cache, then the least recently used

[4] e.g., the least recently used

Remark 10. *To simplify analysis, the algorithm is defined non-lazily here in the sense that it may load pages even when they are not requested. For instance, the page evicted in step 1 might be immediately reloaded in step 4 (in particular, this will always be the case when r is clean). An implementation should only simulate this non-lazy algorithm in the background and, whenever the actual algorithm has a page fault, it evicts an arbitrary (e.g., the least recently used) page that is present in its own cache but missing from the simulated cache.*

The proof of Theorem 3 can be found in supplementary material.

5. Experiments

We evaluate the practicality of our approach on real-world datasets for two MTS: *caching* and *ice cream* problem. The source code and datasets are available at GitHub[5]. Each experiment was run 10 times and we report the mean competitive ratios. The maximum standard deviation we observed was of the order of 0.001.

5.1. The Caching Problem

Datasets. For the sake of comparability, we used the same two datasets as Lykouris & Vassilvitskii (2018).

- `BK` dataset comes from a former social network BrightKite (Cho et al., 2011). It contains checkins with user IDs and locations. We treat the sequence of checkin locations of each users as a separate instance of caching problem. We filter users with the maximum sequence length (2100) who require at least 50 evictions in an optimum cache policy. Out of those we take the first 100 instances. We set the cache size to $k = 10$.
- `Citi` dataset comes from a bike sharing platform CitiBike. For each month of 2017, we consider the first 25 000 bike trips and build an instance where a request corresponds to the starting station of a trip. We set the cache size to $k = 100$.

Predictions. We first generate predictions regarding the next time that the requested page will appear, this prediction being used by previous prediction-augmented algorithms. To this purpose, we use the same two predictors as Lykouris & Vassilvitskii (2018). Additionally we also consider a simple predictor, which we call POPU (from *popularity*), and the LRU heuristic adapted to serve as a predictor.

[5] https://github.com/adampolak/mts-with-predictions

- Synthetic predictions: we first compute the exact next arrival time for each request, setting it to the end of the instance if it does not reappear. We then add some noise drawn from a lognormal distribution, with the mean parameter 0 and the standard deviation σ, in order to model rare but large failures.
- PLECO predictions: we use the PLECO model described in Anderson et al. (2014), with the same parameters as Lykouris & Vassilvitskii (2018), which were fitted for the `BK` dataset (but not refitted for `Citi`). This model estimates that a page requested x steps earlier will be the next request with a probability proportional to $(x+10)^{-1.8}e^{-x/670}$. We sum the weights corresponding to all the earlier appearances of the current request to obtain the probability p that this request is also the next one. We then estimate that such a request will reappear $1/p$ steps later.
- POPU predictions: if the current request has been seen in a fraction p of the past requests, we predict it will be repeated $1/p$ steps later.
- LRU predictions: Lykouris & Vassilvitskii (2018) already remarked on (but did not evaluate experimentally) a predictor that emulates the behavior of the LRU heuristic. A page requested at time t is predicted to appear at time $-t$. Note that the algorithms only consider the order of predicted times among pages, and not their values, so the negative predictions pointing to the past are not an issue.

We then transform these predictions (which are tailored to the caching problem) to our prediction setup (which is designed for general MTS) by simulating the algorithm that evicts the element predicted to appear the furthest in the future. In each step the prediction to our algorithm is the configuration of this algorithm. Note that in the case of LRU predictions, the predicted configuration is precisely the configuration of the LRU algorithm.

Algorithms. We considered the following algorithms. Two online algorithms: the heuristic LRU, which is considered the gold standard for caching, and the $O(\log k)$-competitive Marker (Fiat et al., 1994). Three robust algorithms from the literature using the "next-arrival time" predictions: L&V (Lykouris & Vassilvitskii, 2018), LMarker (Rohatgi, 2020), and LNonMarker (Rohatgi, 2020). Three algorithms using the prediction setup which is the focus of this paper: FtP, which naively follows the predicted state, RobustFtP, which is defined as $MIN^{rand}(\text{FtP}, \text{Marker})$, and is an instance of the general MTS algorithm described in Section 3, and Trust&Doubt, the caching algorithm described in Section 4.

We implemented the deterministic and randomized combination schemes described in Section 2 with a subtlety for the caching problem: we do not flush the whole cache when switching algorithms, but perform only a single eviction per page fault in the same way as described in Remark 10. We set the parameters to $\gamma = 1 + 0.01$ and $\epsilon = 0.01$. These values, chosen from $\{10^{-i} : i = 0, \ldots, 4\}$, happen to be consistently the best choice in all our experimental settings.

Results. For both datasets, for each algorithm and each prediction considered, we computed the total number of page faults over all the instances and divided it by the optimal number in order to obtain a *competitive ratio*. Figure 1 presents the performance of a selection of the algorithms depending on the noise of synthetic predictions for the `BK` dataset. We omit LMarker and LNonMarker for readability since they perform no better than L&V. In supplementary material, we present the performance of all the algorithms on the `BK` and `Citi` datasets. The experiment suggests that our algorithm Trust&Doubt outperforms previous prediction-based algorithms as well as LRU. In Table 1 we provide the results obtained on both datasets using PLECO, POPU, and LRU predictions. We observe that PLECO predictions are not accurate enough to allow previously known algorithms to improve over the Marker algorithm. This may be due to the sensitivity of this predictor to consecutive identical requests, which are irrelevant for the caching problem. However, using the simple POPU predictions enables the prediction-augmented algorithms to significantly improve their performance compared to the classical online algorithms. Using Trust&Doubt with either of the predictions is however sufficient to get a performance similar or better than LRU (and than all other alternatives, excepted for POPU predictions on the BK dataset). RobustFtP, although being a very generic algorithm with worse theoretical guarantees, achieves a performance which is not that far from previously known algorithms. Note that we did not use a prediction model tailored to our setup, which suggests that even better results can be achieved. When we use the LRU heuristic as a predictor, all the prediction-augmented algorithms perform comparably to the bare LRU algorithm. For Trust&Doubt and RobustFTP, there is a theoretical guarantee that this must be the case: Since the prediction error with respect to LRU is 0, these algorithms are $O(1)$-competitive against LRU. Thus, Trust&Doubt achieves both the practical performance of LRU with an exponentially better worst-case guarantee than LRU. Note that Lykouris & Vassilvitskii (2018) also discuss how their algorithm framework performs when using LRU predictions, but did not provide both of these theoretical guarantees simultaneously.

5.2. A Simple MTS: the *Ice Cream* Problem

We consider a simple MTS example from Chrobak & Larmore (1998), named *ice cream* problem. It it an MTS with two states, named v and c, at distance 1 from each other, and two types of requests, V and C. Serving a request while being in the matching state costs 1 for V and 2 for C, and the costs are doubled for the mismatched state. The problem

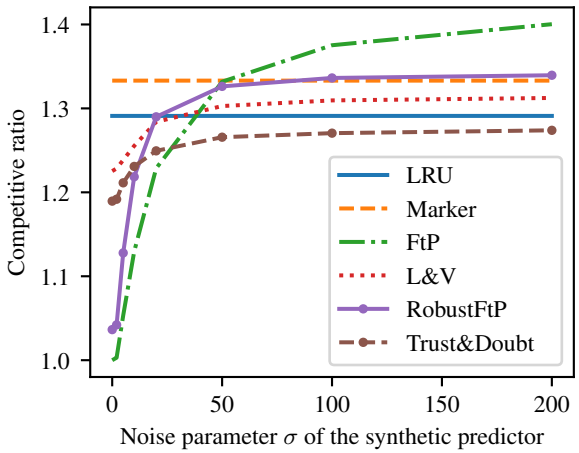

Figure 1. Comparison of caching algorithms augmented with synthetic predictions on the BK dataset.

Table 1. Competitive ratios of caching algorithms using PLECO, POPU, and LRU predictions on both datasets.

Dataset	BK			Citi		
LRU	1.291			1.848		
Marker	1.333			1.861		
Predictions	PLECO	POPU	LRU	PLECO	POPU	LRU
FtP	2.081	1.707	1.291	2.277	1.739	1.848
L&V	1.340	1.262	1.291	1.877	1.776	1.848
LMarker	1.337	1.264	1.291	1.876	1.780	1.848
LNonMarker	1.339	1.292	1.311	1.882	1.800	1.855
RobustFtP	1.351	1.316	1.301	1.885	1.831	1.859
Trust&Doubt	1.292	1.274	1.291	1.847	1.774	1.848

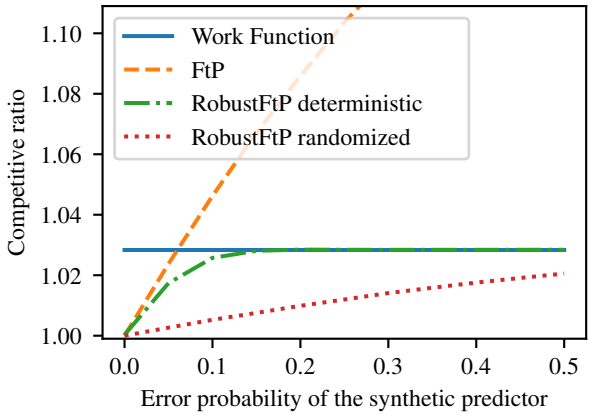

Figure 2. Performance on the ice cream problem with synthetic predictions.

is motivated by an ice cream machine which operates in two modes (states) – vanilla or chocolate – each facilitating a cheaper production of a type of ice cream (requests).

We use the BrightKite dataset to prepare test instances for the problem. We extract the same 100 users as for caching. For each user we look at the geographic coordinates of the checkins, and we issue a V request for each checkin in the northmost half, and a C request for each checkin in the southmost half.

In order to obtain synthetic predictions, we first compute the optimal offline policy, using dynamic programming. Then, for an error parameter p, for each request we follow the policy with probability $1 - p$, and do the opposite with probability p.

We consider the following algorithms: the Work Function algorithm (Borodin et al., 1992; Borodin & El-Yaniv, 1998), of competitive ratio of 3 in this setting ($2n - 1$ in general); FtP, defined in Section 3 (in case of ties in Equation (2), we follow the prediction); and the deterministic and randomized combination of the two above algorithms (with the same ϵ and γ as previously) as proposed in Section 3.

Figure 2 presents the competitive ratios we obtained. We can see that the general MTS algorithms we propose in Section 3 allow to benefit from good predictions while providing the worst-case guarantee of the classical online algorithm. The deterministic combination is comparable to the best of the algorithms combined. Quite surprisingly, the randomized combination performs even better, even when predictions are completely random. A likely reason for this phenomenon is that the randomized combination, when following at the moment the Work Function algorithm, overrides its choice with non-zero probability only when Work Function makes a non-greedy move. This makes the combined algorithm more greedy, which is beneficial in the case of the ice cream problem.

6. Conclusion

In this paper, we proposed a prediction setup that allowed us to design a general prediction-augmented algorithm for a large class of problems encompassing MTS. For the MTS problem of caching in particular, the setup requires less information than previously studied ones. Nevertheless, we can design a specific algorithm for the caching problem in our setup which offers guarantees similar to previous algorithms and even performs better in most of our experiments. Future work includes designing specific algorithms for other MTS problems in our setup, e.g., weighted caching, k-server and convex body chasing. Another research direction is to identify more sophisticated predictors for caching and other problems that will further enhance the performance of prediction-augmented algorithms.

Acknowledgements

The authors would like to thank IGAFIT for the organization of the AlgPiE workshop which made this project possible. A. Antoniadis was supported by DFG Grant AN 1262/1-1, C. Coester was supported by NWO VICI grant 639.023.812 of Nikhil Bansal, M. Elias was supported by ERC Starting Grant 759471 of Michael Kapralov, A. Polak was supported by National Science Center of Poland grant 2017/27/N/ST6/01334, and B. Simon was supported by DFG project number 146371743 – TRR89 Invasive Computing.

References

Anderson, A., Kumar, R., Tomkins, A., and Vassilvitskii, S. The dynamics of repeat consumption. In *Proceedings of conference World Wide Web '14*, pp. 419–430, 2014. doi: 10.1145/2566486.2568018.

Angelopoulos, S., Dürr, C., Jin, S., Kamali, S., and Renault, M. Online Computation with Untrusted Advice. In *Proceedings of ITCS'20*, volume 151, pp. 52:1–52:15, 2020. doi: 10.4230/LIPIcs.ITCS.2020.52.

Antoniadis, A., Coester, C., Elias, M., Polak, A., and Simon, B. Online metric algorithms with untrusted predictions, 2020. URL https://arxiv.org/abs/2003.02144.

Bansal, N., Buchbinder, N., and Naor, J. A primal-dual randomized algorithm for weighted paging. *J. ACM*, 59 (4):19:1–19:24, 2012. doi: 10.1145/2339123.2339126.

Belady, L. A. A study of replacement algorithms for a virtual-storage computer. *IBM Syst. J.*, 5(2):78–101, 1966. doi: 10.1147/sj.52.0078.

Blum, A. and Burch, C. On-line learning and the metrical task system problem. *Machine Learning*, 39(1):35–58, 2000. doi: 10.1023/A:1007621832648.

Borodin, A. and El-Yaniv, R. *Online computation and competitive analysis*. Cambridge University Press, 1998.

Borodin, A., Linial, N., and Saks, M. E. An optimal online algorithm for metrical task system. *J. ACM*, 39(4): 745–763, 1992. doi: 10.1145/146585.146588.

Boyar, J., Favrholdt, L. M., Kudahl, C., Larsen, K. S., and Mikkelsen, J. W. Online algorithms with advice: A survey. *ACM Comput. Surv.*, 50(2):19:1–19:34, 2017. doi: 10.1145/3056461.

Bubeck, S. and Rabani, Y. Parametrized metrical task systems. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2020)*, 2020. To appear.

Bubeck, S., Cohen, M. B., Lee, J. R., and Lee, Y. T. Metrical task systems on trees via mirror descent and unfair gluing. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pp. 89–97, 2019. doi: 10.1137/1.9781611975482.6.

Cho, E., Myers, S. A., and Leskovec, J. Friendship and mobility: user movement in location-based social networks. In *Proceedings of SIGKDD'11*, pp. 1082–1090, 2011. doi: 10.1145/2020408.2020579. URL https://snap.stanford.edu/data/loc-brightkite.html.

Chrobak, M. and Larmore, L. L. Metrical task systems, the server problem and the work function algorithm. In *Online Algorithms*, pp. 74–96. Springer, 1998. doi: 10.1007/BFb0029565.

Chung, T. H. Approximate methods for sequential decision making using expert advice. In *Proceedings of COLT'94*, COLT '94, pp. 183–189. Association for Computing Machinery, 1994. doi: 10.1145/180139.181097.

CitiBike. Citi bike trip histories. https://www.citibikenyc.com/system-data. Accessed: 02/02/2020.

Coester, C. and Koutsoupias, E. The online k-taxi problem. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pp. 1136–1147, 2019. doi: 10.1145/3313276.3316370.

Coester, C. and Lee, J. R. Pure entropic regularization for metrical task systems. In *Conference on Learning Theory, COLT 2019*, pp. 835–848, 2019.

Daniely, A. and Mansour, Y. Competitive ratio vs regret minimization: achieving the best of both worlds. In *Proceedings of ALT 2019*, pp. 333–368, 2019. URL http://proceedings.mlr.press/v98/daniely19a.html.

Dehghani, S., Ehsani, S., Hajiaghayi, M., Liaghat, V., and Seddighin, S. Stochastic k-Server: How Should Uber Work? In *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80, pp. 126:1–126:14, 2017. doi: 10.4230/LIPIcs.ICALP.2017.126.

Emek, Y., Fraigniaud, P., Korman, A., and Rosén, A. Online computation with advice. *Theor. Comput. Sci.*, 412(24): 2642–2656, 2011. doi: 10.1016/j.tcs.2010.08.007.

Fiat, A., Rabani, Y., and Ravid, Y. Competitive k-server algorithms. *J. Comput. Syst. Sci.*, 48(3):410–428, 1994. doi: 10.1016/S0022-0000(05)80060-1.

Fiat, A., Foster, D. P., Karloff, H. J., Rabani, Y., Ravid, Y., and Vishwanathan, S. Competitive algorithms for layered graph traversal. *SIAM J. Comput.*, 28(2):447–462, 1998. doi: 10.1137/S0097539795279943.

Freund, Y. and Schapire, R. E. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997. doi: https://doi.org/10.1006/jcss.1997.1504.

Gollapudi, S. and Panigrahi, D. Online algorithms for rent-or-buy with expert advice. In *Proceedings of ICML'19*, pp. 2319–2327, 2019. URL http://proceedings.mlr.press/v97/gollapudi19a.html.

Irani, S., Shukla, S., and Gupta, R. Online strategies for dynamic power management in systems with multiple power-saving states. *ACM Trans. Embed. Comput. Syst.*, 2(3):325–346, 2003. doi: 10.1145/860176.860180.

Jiang, Z., Panigrahi, D., and Su, K. Online algorithms for weighted paging with predictions. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, 2020. To appear.

Karloff, H. J., Rabani, Y., and Ravid, Y. Lower bounds for randomized k-server and motion-planning algorithms. *SIAM J. Comput.*, 23(2):293–312, 1994. doi: 10.1137/S0097539792224838.

Khalil, E. B., Dilkina, B., Nemhauser, G. L., Ahmed, S., and Shao, Y. Learning to run heuristics in tree search. In *Proceedings of IJCAI'17*, pp. 659–666, 2017. doi: 10.24963/ijcai.2017/92.

Kraska, T., Beutel, A., Chi, E. H., Dean, J., and Polyzotis, N. The case for learned index structures. In *Proceedings of SIGMOD'18*, pp. 489–504, 2018. doi: 10.1145/3183713.3196909.

Lattanzi, S., Lavastida, T., Moseley, B., and Vassilvitskii, S. Online scheduling via learned weights. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'20, pp. 1859–1877, 2020.

Lin, M., Wierman, A., Andrew, L. L. H., and Thereska, E. Dynamic right-sizing for power-proportional data centers. *IEEE/ACM Trans. Netw.*, 21(5):1378–1391, 2013. doi: 10.1109/TNET.2012.2226216.

Littlestone, N. and Warmuth, M. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, February 1994. doi: 10.1006/inco.1994.1009.

Lykouris, T. and Vassilvitskii, S. Competitive caching with machine learned advice. In *Proceedings of ICML'18*, pp. 3302–3311, 2018. URL http://proceedings.mlr.press/v80/lykouris18a.html.

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

Manasse, M. S., McGeoch, L. A., and Sleator, D. D. Competitive algorithms for server problems. *J. ACM*, 11(2):208–230, 1990. doi: 10.1016/0196-6774(90)90003-W.

Medina, A. M. and Vassilvitskii, S. Revenue optimization with approximate bid predictions. In *Proceedings of NeurIPS'17*, pp. 1858–1866, 2017.

Mitzenmacher, M. Scheduling with predictions and the price of misprediction. In *Proceedings of ITCS'20*, pp. 14:1–14:18, 2020. doi: 10.4230/LIPIcs.ITCS.2020.14.

Purohit, M., Svitkina, Z., and Kumar, R. Improving online algorithms via ML predictions. In *Proceedings of NeurIPS'18*, pp. 9684–9693, 2018.

Rohatgi, D. Near-optimal bounds for online caching with machine learned advice. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'20, pp. 1834–1845, 2020.

Sleator, D. D. and Tarjan, R. E. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985. doi: 10.1145/2786.2793.

Wei, A. Better and simpler learning-augmented online caching. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2020)*, 2020. To appear.