Online metric algorithms with untrusted predictions[*]

Antonios Antoniadis[†]
Saarland University and
Max-Planck Institute for Informatics

Christian Coester[‡]
CWI

Marek Eliáš[§]
EPFL

Adam Polak[¶]
Jagiellonian University

Bertrand Simon[‖]
University of Bremen

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 and Tarjan, 1985; Manasse et al., 1990), movements of service vehicles (Dehghani et al., 2017; Coester and 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 and Mansour, 2019; Blum and 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 $t = 1, 2, \dots,$ 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 $\mathit{dist}(x_{t-1}, x_t) + \ell_t(x_t)$, where $\mathit{dist}(x_{t-1}, x_t)$ 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

[*] The authors would like to thank IGAFIT for the organization of the AlgPiE workshop which made this project possible.

[†] Supported by DFG Grant AN 1262/1-1.

[‡] Supported by NWO VICI grant 639.023.812 of Nikhil Bansal.

[§] Supported by ERC Starting Grant 759471 of Michael Kapralov.

[¶] Supported by National Science Center of Poland grant 2017/27/N/ST6/01334.

[‖] Supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project Number 146371743 – TRR 89 Invasive Computing

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 and Lee (2019); Bubeck and 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 and 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 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 and 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 and 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 and 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 and 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 $p_1, \dots, p_T$ 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 $\textsc{Off} = 0.1$.

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*

$$9 \cdot \min\{\alpha,\ 1 + 4\eta/\textsc{Off}\}$$

against any offline algorithm $\textsc{Off}$, where η is the prediction error with respect to $\textsc{Off}$.

Roughly speaking, the competitive ratio (formally defined in Section 2) is the worst case ratio between the cost of two algorithms. If $\textsc{Off}$ 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*

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

where η is the prediction error with respect to an offline algorithm $\textsc{Off}$. Thus, if $\textsc{Off}$ is (near-)optimal and $\eta \ll \textsc{Off}$, 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 and Burch (2000). In Theorem 9, we show that the dependence on $\eta/\textsc{Off}$ in the preceding theorems is tight up to constant factors for some MTS instance.

For some other specific MTS, however, the dependence on $\eta/\textsc{Off}$ 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 $\eta/\textsc{Off}$. 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 $O(\min\{1 + \log(1 + \frac{\eta}{\textsc{Off}}), \log k\})$ against any algorithm $\textsc{Off}$, 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 in Section 5 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 $O(\min\{\log n, 1 + \eta/\textsc{Off}\})$, where η is the prediction error with respect to some offline algorithm $\textsc{Off}$.*

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

We show in Appendix B that the predictions used by Lykouris and 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.

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 and Vassilvitskii (2018) on caching, Purohit et al. (2018) on ski rental and non-clairvoyant scheduling, Gollapudi and Panigrahi (2019) on ski rental with multiple predictors, Mitzenmacher (2020) on scheduling/queuing, and Medina and 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 and Vassilvitskii (2018) and Rohatgi (2020), who study the caching problem (a special case of MTS) with machine learned predictions. Lykouris and 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 $O\big(1+\min\{\sqrt{\eta/\,\textsc{Opt}},\log k\}\big)$ and their algorithm was shown to perform well in experiments. Later, Rohatgi (2020) achieved a better dependency on the prediction error: $O(1+\min\{\frac{\log k}{k}\frac{\eta}{\textsc{Opt}},\log k\})$. 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 $O(1+\min\{\frac{1}{k}\frac{\eta}{\textsc{Opt}},\log k\})$ 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 and Vassilvitskii

Although the work of Lykouris and 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 and 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 and Vassilvitskii (2018) and Rohatgi (2020) in empirical experiments in Section 6.

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

(1) If we modify the perfect predictions in the setup of Lykouris and 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 and 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 and 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

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

algorithm above, are $O(1 + \min\{\frac{1}{k}\frac{\eta}{\text{OPT}}, \log k\})$-competitive for caching in the setup of Lykouris and 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 $O(1 + \frac{1}{k}\frac{\eta}{\text{OPT}})$ by Wei (2020), our algorithms are $O(\min\{1 + \frac{1}{k}\frac{\eta}{\text{OPT}}, \log k\})$-competitive, where η denotes the prediction error in the setup of Lykouris and 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 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 and 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 $t = 1, 2, \dots$, 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 $\mathit{dist}(x_{t-1}, x_t) + \ell_t(x_t)$. Note that $\mathit{dist}(x_{t-1}, x_t) = 0$ 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 and 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 and 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*

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

where α is a constant independent of the input sequence, $\mathcal{A}(I)$ is the solution produced by the online algorithm and $\textit{Opt}_I$ 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 $\textit{Opt}_I$ is replaced by the cost of some specific algorithm $\textit{Off}$, we say that $\mathcal{A}$ is r-competitive against $\textit{Off}$.

2.1 Combining Online Algorithms

Consider m algorithms $A_0, \dots, A_{m-1}$ 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 and Vassilvitskii (2018). We simulate the execution of $A_0, \dots, A_{m-1}$ 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 $A_0, \dots A_{m-1}$ 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 Appendix A. 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 and Burch (2000) proposed the following way to combine online algorithms based on the WMR (Littlestone and 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 probability 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 and Burch, 2000)

```
$\beta := 1 - \frac{\epsilon}{2}$;                                  // for parameter $\epsilon < 1/2$
$w_i^0 := 1$ for each $i = 0, \dots, m-1$;
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. $\tau_{ij}/p_i^t$;
```

Theorem 7 (Blum and Burch (2000))**.** *Given m on-line algorithms $A_0, \dots A_{m-1}$ 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*.*

The proof of this lemma will follow the formal description of the FtP algorithm.

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

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

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.

Proof of Lemma 8. At each time t, the FtP algorithm is located at configuration x_{t-1} and needs to choose x_t after receiving task ℓ_t and prediction p_t. Let us consider some offline algorithm Off. We denote $x_0, o_1, \dots, o_T$ the states of Off, where the initial state x_0 is common for Off and for FtP, and T denotes the length of the sequence.

We define A_t to be the algorithm which agrees with FtP in its first t configurations $x_0, x_1, \dots, x_t$ and then agrees with the states of Off, i.e., $o_{t+1}, \dots, o_T$. Note that $\mathit{cost}(A_0) = \text{Off}$ and $\mathit{cost}(A_T) = \mathit{cost}(\text{FtP})$. We claim that $\mathit{cost}(A_t) \le \mathit{cost}(A_{t-1}) + 4\eta_t$ for each t, where $\eta_t = \mathit{dist}(p_t, o_t)$. The algorithms A_t and A_{t-1} are in the same configuration at each time except t, when A_t is in x_t while A_{t-1} is in o_t. By triangle inequality, we have

$$\begin{aligned}\mathit{cost}(A_t) &\le \mathit{cost}(A_{t-1}) + 2\mathit{dist}(o_t, x_t) + \ell_t(x_t) - \ell_t(o_t)\\ &\le \mathit{cost}(A_{t-1}) + 2\mathit{dist}(o_t, p_t) - \ell_t(o_t) + \ 2\mathit{dist}(p_t, x_t) + \ell_t(x_t)\\ &\le \mathit{cost}(A_{t-1}) + 4\mathit{dist}(o_t, p_t),\end{aligned}$$

The last inequality follows from (2): we have $2\mathit{dist}(p_t, x_t) + \ell_t(x_t) \le 2\mathit{dist}(p_t, o_t) + \ell_t(o_t)$. By summing over all times $t = 1, \dots, T$, we get

$$\mathit{cost}(\text{FtP}) = \mathit{cost}(A_T) \le \mathit{cost}(A_0) + 4\textstyle\sum_{t=1}^{T}\eta_t,$$

which equals $\text{Off} + 4\eta$. □

3.1 Lower bound

We show that our upper bounds for general metrical task systems (Theorems 1 and 2) are tight up to constant factors. We show this for MTS on a uniform metric, i.e., the metric where the distance between any two points is 1.

Theorem 9. *For* $\bar\eta \ge 0$ *and* $n \in \mathbb{N}$*, every deterministic (or randomized) online algorithm for MTS on the* n*-point uniform metric with access to a prediction oracle with error at most* $\bar\eta\,\text{Opt}$ *with respect to some optimal offline algorithm has competitive ratio* $\Omega(\min\{\alpha_n, 1+\bar\eta\})$*, where* $\alpha_n = \Theta(n)$ *(or* $\alpha_n = \Theta(\log n)$*) is the optimal competitive ratio of deterministic (or randomized) algorithms without prediction.*

Proof. For deterministic algorithms, we construct an input sequence consisting of phases defined as follows. We will ensure that the online and offline algorithm be located at the same point at the beginning of a phase. The first $\min\{n-2, \lfloor\bar{\eta}\rfloor\}$ cost functions of a phase always take value ∞ at the old position of the online algorithm and value 0 elsewhere, thus forcing the algorithm to move. Let p be a point that the online algorithm has not visited since the beginning of the phase. Only one more cost function will be issued to conclude the phase, which takes value 0 at p and ∞ elsewhere, hence forcing both the online and offline algorithm to p. The optimal offline algorithm suffers cost exactly 1 per phase because it can move to p already at the beginning of the phase. The error is at most $\bar{\eta}$ per phase provided that point p is predicted at the last step of the phase, simply because there are only at most $\bar{\eta}$ other steps in the phase, each of which can contribute at most 1 to the error. Thus, the total error is at most $\bar{\eta}\,\textsc{Opt}$. The online algorithm suffers cost $\min\{n-1, 1+\lceil\bar{\eta}\rceil\}$ during each phase, which proves the deterministic lower bound.

For randomized algorithms, let $k := \lfloor\log_2 n\rfloor$ and fix a subset F_0 of the metric space of 2^k points. We construct again an input sequence consisting of phases: For $i = 1, \dots, \min\{k, \lfloor\bar{\eta}\rfloor\}$, the ith cost function of a phase takes value 0 on some set F_i of feasible states and ∞ outside of F_i. Here, we define $F_i \subset F_{i-1}$ to be the set consisting of that half of the points of F_{i-1} where the algorithm's probability of residing is smallest right before the ith cost function of the phase is issued (breaking ties arbitrarily). Thus, the probability of the algorithm already residing at a point from F_i when the ith cost function arrives is at most $1/2$, and hence the expected cost per step is at least $1/2$. We assume that $\bar{\eta} \geq 1$ (otherwise the theorem is trivial). Similarly to the deterministic case, the phase concludes with one more cost function that forces the online and offline algorithm to some point p in the final set F_i. Again, the optimal cost is exactly 1 per phase, the error is at most $\bar{\eta}$ in each phase provided the last prediction of the phase is correct, and the algorithm's expected cost per phase is at least $\frac{1}{2}\min\{k, \lfloor\bar{\eta}\rfloor\} = \Omega(\min(\log n, 1+\bar{\eta}))$, concluding the proof. □

In light of the previous theorem it may seem surprising that our algorithm Trust&Doubt for caching (see Section 4) achieves a competitive ratio logarithmic rather than linear in the prediction error, especially considering that the special case of caching when there are only $k+1$ distinct pages corresponds to an MTS on the uniform metric. However, the construction of the randomized lower bound in Theorem 9 requires cost functions that take value ∞ at several points at once, whereas in caching only one page is requested per time step.

4 Logarithmic Error Dependence for Caching

We describe in this section a new algorithm for the (unweighted) caching problem, which we call Trust&Doubt, and prove Theorem 3. The algorithm 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 $O(\log k)$. Complementing the description of Trust&Doubt that will follow below, we also provide a pseudo-code formulation in Algorithm 3.

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[2]. 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 and Vassilvitskii (2018), belong to the class of so-called *marking algorithms*, which evict a page only if it is unmarked. In

[2] Subdividing the input sequence into such phases is a very common technique in the analysis of caching algorithms, see for example Borodin and El-Yaniv (1998) and references therein.

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 6, 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 associates 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 $\mathit{trusted}(q)$ 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 $T = \{p_q \mid q \in C, \mathit{trusted}(q) = \mathit{true}\}$ and $D = \{p_q \mid q \in C, \mathit{trusted}(q) = \mathit{false}\}$ 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 $\mathit{trusted}(r) := \mathit{true}$ 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 $\mathit{trusted}(q) := \mathit{false}$.

4. For each $q \in C$ for which a new q-interval began with this request: If $\mathit{trusted}(q) = \mathit{false}$, set $t_q := 2t_q$ and $\mathit{trusted}(q) := \mathit{true}$. 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.

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*

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

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

Algorithm 3: TRUST&DOUBT's action when page r requested and cache P predicted

```
if $|marked| = k$ and $r \notin marked$ then // Start new phase
    $A \leftarrow \{p \in cache \mid p \notin marked\}$
    $stale \leftarrow cache - A$
    $marked \leftarrow \emptyset$; $U \leftarrow stale$; $M \leftarrow \emptyset$; $T \leftarrow \emptyset$; $D \leftarrow \emptyset$; $C \leftarrow \emptyset$
    assign random priorities to pages in $U$
if $r \notin marked$ then // Arrival of $r$
    $isArrival \leftarrow true$
    $marked \leftarrow marked + \{r\}$
    if $r \notin T$ then $M \leftarrow M + \{r\}$
    $U \leftarrow U - \{r\}$ // this has no effect if $r \notin U$
else $isArrival \leftarrow false$
if $A \neq \emptyset$ then
    if $r \in A$ then
        $A \leftarrow A - \{r\}$
    else if $r \notin cache$ then
        Select $p \in A$ // e.g., choose $p \notin P$ if $A \not\subseteq P$, then select the least
            recently used
        $A \leftarrow A - \{p\}$
        $cache \leftarrow cache - \{p\} + \{r\}$
else
    // step 1:
    if $r \notin cache$ then
        Let $p$ be the page from $U \cap cache$ with lowest priority
        $cache \leftarrow cache - \{p\} + \{r\}$
    // step 2:
    if $r \notin stale$ and $isArrival$ then
        $C \leftarrow C + \{r\}$
        Define $p_r$ to be any page from $U + M - D - P$ // e.g., least recently used
        $trusted(r) \leftarrow true$; $T \leftarrow T + \{p_r\}$; $U \leftarrow U - \{p_r\}$; $M \leftarrow M - \{p_r\}$
        $t_r \leftarrow 1$
        $r\text{-}interval_{change} \leftarrow |marked|$ // To recognize new r-interval in step 4
    // step 3:
    if $\exists q \in C \colon p_q = r$ then
        $D \leftarrow D - \{p_q\}$; $T \leftarrow T - \{p_q\}$
        if $p_q \in marked$ then $M \leftarrow M + \{p_q\}$ else $U \leftarrow U + \{p_q\}$ // No effect if $p_q$
            was in $D$
        Redefine $p_q$ to be any page from $U + M - D - P$ // e.g., least recently used
        $trusted(q) \leftarrow false$; $D \leftarrow D + \{p_q\}$
    // step 4:
    for $q \in C$ do
        if $isArrival$ and $q\text{-}interval_{change} = |marked|$ then
            if $trusted(q) = false$ then
                $t_q \leftarrow 2t_q$
                $trusted(q) \leftarrow true$; $T \leftarrow T + \{p_q\}$; $D \leftarrow D - \{p_q\}$; $U \leftarrow U - \{p_q\}$; $M \leftarrow$
                  $M - \{p_q\}$;
            $q\text{-}interval_{change} = |marked| + t_q$ // time (in #arrivals) of the next
                q-interval
            if $p_q \in cache$ then
                Let $p$ be the page from $U - cache$ with highest priority
                $cache \leftarrow cache - \{p_q\} + \{p\}$
```

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 following lemma shows that TRUST&DOUBT is well-defined and proves properties that its cache invariantly satisfies.

Lemma 11. *The pages possibly to be defined as p_r or p_q in steps 2 and 3 and any page to be loaded in step 4 exist. Before each request when $A = \emptyset$, TRUST&DOUBT's configuration is a subset of $U \cup M$ and contains M, and $|U \cup M| = k + |D|$.*

Proof. The steps 1–4 are only executed when $A = \emptyset$. We first show that the first sentence of the lemma is true before a request assuming the second sentence is true before this request. Notice that $D \subseteq U \cup M$, so the set $(U \cup M) \setminus D$ has size exactly k before the request due to the second sentence.

In step 2, the page to be chosen from $(U \cup M) \setminus D$ exists because the clean page r just became element of this set, thus making the size of this set equal to $k + 1$. At least one of these $k + 1$ elements must be missing from the predictor's cache.

In step 3, one of the k elements of $(U \cup M) \setminus D$ must be missing from the predictor's cache because r is not contained in this set but r is in the predictor's cache.

In step 4, p_q is in the algorithm's cache only if it was in D before the request. But since the second sentence of the lemma was true before the request by assumption, $|D|$ pages from U are missing from the cache. Thus, sufficiently many pages from U are missing as need to be evicted in step 4.

It remains to prove that the second sentence of the lemma holds. When A first becomes empty in a phase, this is clearly true with $D = \emptyset$.

Since TRUST&DOUBT loads a page to its cache only if it is in $M \cup U$, and whenever a page is removed from $M \cup U$ (by becoming element of T) it is also evicted, the cache content remains a subset of $U \cup M$. Since it evicts marked pages only if they are in T, the configuration contains M.

Finally, let us show that $|U \cup M| = k + |D|$ is maintained. In step 1, the size of $U \cup M$ increases by 1 if the condition of step 2 is true and is unchanged otherwise. But in the positive case, a page p_q will be moved to T in step 2 and hence removed from $|U \cup M|$ again. So $|U \cup M| = k + |D|$ holds after step 2. In step 3, if $r \in D$, then membership in $U \cup M$ is unchanged, r will be removed from D, but the new page p_q added to D, thus keeping the size of D also unchanged. If $r \in T$, then r will be added to $U \cup M$, but the new p_q is also added to D. Still, $|U \cup M| = k + |D|$ holds. In step 4, any p_q that might be removed from D is added to T at the same time, and therefore removed from $U \cup M$. □

Lemma 12. *Each page from U is missing from TRUST&DOUBT's cache with probability $\frac{|D|}{|U|}$.*

Proof. While $A \neq \emptyset$, the set D is empty and all pages of U are in the cache. When $A = \emptyset$, by Lemma 11 there are $|D|$ pages from U that are missing from the algorithm's cache. Since these are the ones with lowest priority, and priorities were chosen uniformly at random, the missing elements of U are a uniformly random subset of size $|D|$ of the current set U. □

Let C_ℓ denote the set C at the end of phase ℓ. The next lemma and its proof follow closely Fiat et al. (1991). However, since our definition of a clean page is different, we cannot use their result directly, and we need to reprove it in our setting.

Lemma 13. *Any offline algorithm suffers cost at least*

$$\textsc{Off} \geq \Omega\left(\sum_{\ell} |C_\ell|\right).$$

Proof. We first claim that at least $k+|C_\ell|$ distinct pages are requested in phases $\ell-1$ and ℓ together. This claim is trivial if C_ℓ is empty, so suppose it is non-empty. Then A was emptied during phase ℓ. Let a be the size of A at the beginning of phase ℓ. There are $k - a$ stale pages in phase ℓ, and all of these pages were also requested in phase $\ell - 1$. Moreover, there were a non-stale pages requested until A was emptied in phase ℓ and another $|C_\ell|$ non-stale pages afterwards. Overall, the number of distinct pages requested in phases $\ell - 1$ and ℓ is at least $k - a + a + |C_\ell| = k + |C_\ell|$.

Thus, any algorithm must suffer at least cost $|C_\ell|$ during these two phases. Hence, Off is lower bounded by the sum of $|C_\ell|$ over all even phases and, up to a constant, by the according sum over all odd phases. The lemma follows. □

Finally, we obtain the main result of this section:

Theorem (Restated Theorem 3)**.** *Trust&Doubt has competitive ratio* $O(\min\{1 + \log(1 + \frac{\eta}{\text{Off}}), \log k\})$ *against any offline algorithm Off, where* η *is the prediction error with respect to Off.*

Proof. We call a q-interval *doubted* if *trusted*$(q) =$ *false* at the end of the interval. Let d_q be the number of doubted q-intervals in phase ℓ. Note that the length (in terms of number of arrivals) of the ith doubted q-intervals is 2^{i-1}, except that the last one might be shorter. We use variables r_q, e_q, o_q to count the number of doubted q-intervals in phase ℓ with certain characteristics:

- r_q is the number of doubted q-intervals at whose end p_q is *not* in the offline algorithm's cache.
- e_q is the number of doubted q-intervals such that throughout the entire interval, p_q is in the offline algorithm's cache.
- o_q is the number of doubted q-intervals during which the offline algorithm has a page fault upon a request to p_q.

We may sometimes write $d_{q,\ell}$ instead of d_q and similarly for the other variables to emphasize the dependence on the phase. Another lower bound for the offline cost is

$$\text{Off} \geq \sum_\ell \sum_{q \in C_\ell} o_{q,\ell}. \tag{3}$$

Since the current page p_q is never in the predictor's cache, and the ith doubted q-interval contains 2^{i-1} arrivals for $i < d_q$, a lower bound on the prediction error is given by

$$\eta \geq \sum_\ell \sum_{q \in C_\ell} (2^{e_{q,\ell}-1} - 1). \tag{4}$$

We claim that for each $q \in C_\ell$, the following inequalities hold:

$$r_q \leq o_q + 1 \tag{5}$$

$$d_q \leq r_q + e_q + o_q \tag{6}$$

$$d_q \leq O(\log k) \tag{7}$$

The reason for (5) is that if some interval counts towards r_q, then the next interval that counts towards r_q can be no earlier than the next interval that counts towards o_q (because p_q will be in T instead of D at the beginning of the next interval and remain in T until the offline algorithm has a page fault on p_q). To prove inequality (6), we show that each doubted q-interval counts towards at least one of r_q, e_q or o_q. If a doubted q-interval does *not* count towards r_q, then p_q is in the offline cache at the end of the interval. If also all previous pages that were defined as p_q during the interval were in the offline cache at their respective times, then the interval counts towards e_q. Otherwise, some previous p_q was missing from the offline cache; but since this is not the last p_q, this page was requested during the interval, so the offline algorithm suffered a page fault and the interval counts towards o_q. The bound (7) holds because if $d_q \geq 2$, then the $(d_q - 1)$st doubted q-interval contains 2^{d_q-2} arrivals, but there are only k arrivals per phase.

Let us use these connections to bound the cost of the algorithm. The number of page faults suffered while A is non-empty is at most the initial size a of A. Since a marked pages were evicted in the previous phase, this cost can be charged against the cost of the previous phase. It remains to bound the cost during the steps 1–4. Consider some phase ℓ. In step 1, there are three cases in which the algorithm can suffer a page fault: (1) If a clean page arrives, (2) if a page arrives that was in U right before the arrival and is missing in the algorithm's cache and (3) if a page from T is requested. The cost due to case (1) is $|C_\ell|$. In case (2), if a page from U arrives, then the expected cost is $\frac{|D|}{|U|}$ by Lemma 12. We account for this cost by charging $1/|U|$ to each $q \in C$ with $p_q \in D$. Over the whole phase, the number of times we charge to $q \in C$ in this way is at most the total number of arrivals during doubted q-intervals, which is at most 2^{d_q}. By Lemma 11, $|U| \geq k + |D| - |M| \geq k + 1 - |M|$ and if there are i more arrivals to come, then $|M| \leq k - i$, so $|U| \geq i + 1$. Thus, the value of $|U|$ can be lower bounded by $1, 2, 3, \ldots, 2^{d_q}$ during the at most 2^{d_q} arrivals when $1/|U|$ is charged to q. Hence the total cost charged to q is at most $O(d_q)$. The total expected cost due to case (2) in phase ℓ is therefore $O(\sum_{q \in C_\ell} d_q)$. For case (3), we charge the cost due to a request to $p_q \in T$ to the corresponding $q \in C$. Since this can happen at most once for each q during each q-interval, and such a q-interval will be doubted, the cost due to case (3) is also bounded by $\sum_{q \in C_\ell} d_q$. The only other time that the algorithm suffers cost is in step 4 of the algorithm, which is again bounded by $\sum_{q \in C_\ell} d_q$. Thus, the total expected cost during phase ℓ is

$$O(|C_\ell| + \sum_{q \in C_\ell} d_q).$$

Due to (7) and Lemma 13, this shows already that the algorithm is $O(\log k)$ competitive. It remains to bound the competitive ratio in terms of the error. Applying (5), (6) and (3), we can bound the total cost of the algorithm by

$$O(\textsc{Off} + \sum_\ell |C_\ell| + \sum_\ell \sum_{q \in C_\ell} e_{q,\ell}).$$

The summands $e_{q,\ell}$ can be rewritten as $\log_2 \big(1 + [2^{e_{q,\ell}-1} - 1]\big)$. By concavity of $x \mapsto \log(1+x)$, while respecting the bound (4) the sum of these terms is maximized when each term in brackets equals $\frac{\eta}{\sum_\ell |C_\ell|}$, giving a bound on the cost of

$$O\left(\textsc{Off} + \sum_\ell |C_\ell| \left(1 + \log\left(1 + \frac{\eta}{\sum_\ell |C_\ell|}\right)\right)\right).$$

Since this quantity is increasing in $\sum_\ell |C_\ell|$, applying Lemma 13 completes the proof of the theorem. □

Since the performance of Trust&Doubt already matches the lower bound $\Omega(\log k)$ on the competitive ratio of randomized online algorithms without prediction Fiat et al. (1991), an additional combination using the methods from Section 2.1 is not needed here. The competitive ratio of Trust&Doubt when expressed only as a function of the error, $O(1 + \log(1 + \frac{\eta}{\textsc{Off}}))$, is also tight due to the following theorem. It should be noted, though, that for the competitive ratio as a function of both k and $\frac{\eta}{\textsc{Off}}$ it is still plausible that a better bound can be achieved when $\frac{\eta}{\textsc{Off}}$ is relatively small compared to k.

Theorem 14. *If an online caching algorithm achieves competitive ratio at most $f(\frac{\eta}{\textsc{Opt}})$ for arbitrary k when provided with predictions with error at most η with respect to an optimal offline algorithm, then $f(x) = \Omega(\log x)$ as $x \to \infty$.*

Proof. Fix some $k+1$ pages and consider the request sequence where each request is to a uniformly randomly chosen page from this set. We define phases in the same way as in the description of Trust&Doubt. By a standard coupon collector argument, each phase lasts $\Theta(k \log k)$ requests in expectation. An optimal offline algorithm can suffer only one page fault per phase by evicting only the one page that is not requested in each phase. On the other hand, since requests are chosen uniformly at random, any online algorithm suffers a page fault with probability $1/(k+1)$ per request, giving a cost of $\Theta(\log k)$ per phase. Since $\frac{\eta}{\textsc{Opt}} = O(k \log k)$ due to the duration of phases, the competitive ratio of the algorithm is $\Omega(\log k) = \Omega(\log \frac{\eta}{\textsc{Opt}})$. □

5 Online matching on the line

In the *online matching on the line* problem, we are given a set $S = \{s_1, s_2, \ldots s_n\}$ of server locations on the real line. A set of requests $R = \{r_1, r_2, \ldots r_n\}$ which are also locations on the real line, arrive over time. Once request r_i arrives, it has to be irrevocably matched to some previously unmatched server s_j. The cost of this edge in the matching is the distance between r_i and s_j, i.e., $|s_j - r_i|$ and the total cost is given by the sum of all such edges in the final matching, i.e., the matching that matches every request in R to some unique server in S. The objective is to minimize this total cost.

The best known lower bound on the competitive ratio of any deterministic algorithm is 9.001 (Fuchs et al., 2005) and the best known upper bound for any algorithm is $O(\log n)$, due to Raghvendra (2018).

We start by defining the notion of *distance* between two sets of servers.

Definition 2. *Let P_i^1 and P_i^2 be two sets of points in a metric space, of size i each. We then say that their distance $dist(P_i^1, P_i^2)$ is equal to the cost of a minimum-cost perfect matching in the bipartite graph having P_i^1 and P_i^2 as the two sides of the bipartition.*

In *online matching on the line with predictions* we assume that, in each round i along with request r_i, we obtain a prediction $P_i \subseteq S$ with $|P_i| = i$ on the server set that the offline optimal algorithm is using for the first i many requests. Note that it may be the case that $P_i \not\subseteq P_{i+1}$. The error in round i is given by $\eta_i := dist(P_i, \textsc{Off}_i)$, where $\textsc{Off}_i$ is the server set of a (fixed) offline algorithm on the instance. The total prediction error is $\eta = \sum_{i=1}^{n} \eta_i$.

Since a request has to be irrevocably matched to a server, it is not straightforward that one can switch between configurations of different algorithms. Nevertheless, we are able to simulate such a switching procedure. By applying this switching procedure to the best known classic online algorithm for the problem, due to Raghvendra (2018) and designing a Follow-The-Prediction algorithm that achieves a competitive ratio of $1 + 2\eta/\textsc{Off}$, we can apply the combining method of Theorem 6 to get the following result.

Theorem (Restated Theorem 4)**.** *There exists a deterministic algorithm for the online matching on the line problem with predictions that attains a competitive ratio of*

$$\min\{O(\log n), 9 + \frac{8e\eta}{\textsc{Off}}\},$$

for any offline algorithm $\textsc{Off}$.

We note that for some instances the switching cost between these two algorithms (and therefore, in a sense, also the metric space diameter) can be as high as $\Theta(\textsc{Opt})$ which renders the randomized combination uninteresting for this particular problem.

5.1 A potential function

We define the *configuration* of an algorithm at some point in time as the set of servers which are currently matched to a request.

For each round of the algorithm, we define S_i as the current configuration and P_i as the predicted configuration, which verify $|S_i| = |P_i| = i$. We define a potential function after each round i to be $\Phi_i = dist(S_i, P_i)$, and let μ_i be the associated matching between S_i and P_i that realizes this distance, such that all servers in $S_i \cap P_i$ are matched to themselves for zero cost. We extend μ_i to the complete set of severs S by setting $\mu_i(q) = q$ for all $q \notin S_i \cup P_i$. The intuition behind the potential function is that after round i one can simulate being in configuration P_i instead of the actual configuration S_i, at an additional expense of Φ_i.

5.2 Distance among different configurations

The purpose of this section is to show that the distance among the configurations of two algorithms is at most the sum of their current costs. As we will see, this will imply that we can afford switching between any two algorithms.

We continue by bounding the distance between any two algorithms as a function of their costs.

Lemma 15. *Consider two algorithms A and B, and fix the set of servers S as well as the request sequence R. Let A_i and B_i be the respective configurations of the algorithms (i.e., currently matched servers) after serving the first i requests of R with servers from S. Furthermore, let $\textsc{Opt}_i^A$ (resp. $\textsc{Opt}_i^B$) be the optimal matching between $\{r_1, r_2, \ldots r_i\}$ and A_i (resp. B_i), and let M_i^A (resp. M_i^B) be the corresponding matching produced by A (resp. B). Then:*

$$\begin{aligned} \mathit{dist}(A_i, B_i) &\leq \mathit{cost}(\textsc{Opt}_i^A) + \mathit{cost}(\textsc{Opt}_i^B) \\ &\leq \mathit{cost}(M_i^A) + \mathit{cost}(M_i^B). \end{aligned}$$

Proof. The second inequality follows by the optimality of $\textsc{Opt}_i^A$ and $\textsc{Opt}_i^B$. For the first inequality let s_j^A (resp. s_j^B) be the server matched to r_j by $\textsc{Opt}_i^A$ (resp. $\textsc{Opt}_i^B$), for all $j \in \{1, \ldots, i\}$. Therefore, there exists a matching between A_i and B_i that matches for all $j \in \{1, \ldots, i\}$, s_j^A to s_j^B which has a total cost of

$$\begin{aligned} \sum_{j=1}^{i} \mathit{dist}(s_j^A - s_j^B) &\leq \sum_{j=1}^{i} \mathit{dist}(s_j^A - r_j) + \sum_{j=1}^{i} \mathit{dist}(s_j^B - r_j) \\ &= \mathit{cost}(\textsc{Opt}_i^A) + \mathit{cost}(\textsc{Opt}_i^B), \end{aligned}$$

where the inequality follows by the triangle inequality. By the definition of distance we have that $\mathit{dist}(A_i, B_i) \leq \sum_{j=1}^{i} \mathit{dist}(s_j^A - s_j^B)$, which concludes the proof. □

5.3 Follow-The-Prediction

Since *online matching on the line* is not known to be in MTS, we start by redefining the algorithm Follow-The-Prediction for this particular problem. In essence, the algorithm virtually switches from predicted configuration P_i to predicted configuration P_{i+1}.

Let S_i be the actual set of servers used by Follow-The-Prediction after round i. Follow-The-Prediction computes the optimal matching among P_{i+1} and the multiset $P_i \cup \{r_{i+1}\}$ which maps the elements of $P_{i+1} \cap P_i$ to themselves. Note that if $r_{i+1} \in P_i$, then $P_i \cup \{r_{i+1}\}$ is a multiset where r_{i+1} occurs twice. Such matching will match r_{i+1} to some server $s \in P_{i+1} \setminus P_i$. Recall that μ_i is the minimum cost bipartite matching between S_i and P_i extended by zero-cost edges to the whole set of servers. Follow-The-Prediction matches r_{i+1} to the server $\mu_i(s)$, i.e., to the server to which s is matched to under μ_i. We can show easily that $\mu(s) \notin S_i$. Since $s \notin P_i$, there are two possibilities: If $s \notin S_i$, then $\mu(s) = s \notin S_i$ by extension of μ_i to elements which do not belong to S_i nor P_i. Otherwise, $s \in S_i \setminus P_i$ and, since μ_i matches all the elements of $S_i \cap P_i$ to themselves, we have $\mu(s) \in P_i \setminus S_i$.

Theorem 16. *Follow-The-Prediction has total matching cost at most $\textsc{Off} + 2\eta$ and therefore the algorithm has a competitive ratio of*

$$1 + 2\eta/\textsc{Off}$$

against any offline algorithm $\textsc{Off}$.

Proof. The idea behind the proof is that, by paying the switching cost of $\Delta\Phi_i$ at each round, we can always virtually assume that we reside in configuration P_i. So whenever a new request r_{i+1} and a new predicted configuration P_{i+1} arrive, we pay the costs for switching from P_i to P_{i+1} and for matching r_{i+1} to a server in P_{i+1}.

We first show that, for every round i, we have:

$$\begin{aligned} FtP_i + \Delta\Phi_i &\leq \mathit{dist}(P_{i+1}, P_i \cup \{r_{i+1}\}) \\ \Leftrightarrow \quad \mathit{dist}(r_{i+1}, \mu(s)) + \Phi_{i+1} &\leq \mathit{dist}(P_{i+1}, P_i \cup \{r_{i+1}\}) + \Phi_i. \end{aligned}$$

Note that for all j, $\Phi_j = \mathit{dist}(S_j, P_j) = \mathit{dist}(\bar{S}_j, \bar{P}_j)$, where $\bar{S}_j$ and $\bar{P}_j$ denote the complements of S_j and P_j respectively.

We have in addition $\mathit{dist}(\bar{S}_i, \bar{P}_i) = \mathit{dist}(\bar{S}_i \setminus \{\mu_i(s)\}, \bar{P}_i \setminus \{s\}) + \mathit{dist}(s, \mu_i(s))$ as $s \notin P_i$ and $\mu_i(s) \notin S_i$, and $(s, \mu_i(s))$ is an edge in the min-cost matching between $\bar{S}_i$ and $\bar{P}_i$. Note that $S_{i+1} = S_i \cup \{\mu_i(s)\}$ so $\bar{S}_i \setminus \{\mu_i(s)\} = \bar{S}_{i+1}$. Therefore, we get:

$$\Phi_i = \mathit{dist}(\bar{S}_i, \bar{P}_i) = \mathit{dist}(\bar{S}_{i+1}, \bar{P}_i \setminus \{s\}) + \mathit{dist}(s, \mu_i(s)) = \mathit{dist}(S_{i+1}, P_i \cup \{s\}) + \mathit{dist}(s, \mu_i(s)).$$

In addition, we have $\mathit{dist}(P_{i+1}, P_i \cup \{r_{i+1}\}) = \mathit{dist}(s, r_{i+1}) + \mathit{dist}(P_{i+1} \setminus \{s\}, P_i)$ because by definition of s, s is matched to r_{i+1} in a minimum cost matching between P_{i+1} and $P_i \cup \{r_{i+1}\}$. Now, $s \notin P_i$, so $\mathit{dist}(P_{i+1} \setminus \{s\}, P_i) = \mathit{dist}(P_{i+1}, P_i \cup \{s\})$ as this is equivalent to adding a zero-length edge from s to itself to the associated matching. Therefore, we get:

$$\mathit{dist}(P_{i+1}, P_i \cup \{r_{i+1}\}) = \mathit{dist}(s, r_{i+1}) + \mathit{dist}(P_{i+1}, P_i \cup \{s\}).$$

Combining the results above, we obtain:

$$\begin{aligned} & FtP_i + \Delta\Phi_i \leq \mathit{dist}(P_{i+1}, P_i \cup \{r_{i+1}\}) \\ \Leftrightarrow \quad & \mathit{dist}(r_{i+1}, \mu_i(s)) + \Phi_{i+1} \leq \mathit{dist}(P_{i+1}, P_i \cup \{r_{i+1}\}) + \Phi_i \\ \Leftrightarrow \quad & \mathit{dist}(r_{i+1}, \mu_i(s)) + \mathit{dist}(S_{i+1}, P_{i+1}) \\ & \leq \mathit{dist}(S_{i+1}, P_i \cup \{s\}) + \mathit{dist}(s, \mu_i(s)) + \mathit{dist}(s, r_{i+1}) + \mathit{dist}(P_{i+1}, P_i \cup \{s\}) \end{aligned}$$

The last equation holds by the triangle inequality.

Finally, we bound $\mathit{dist}(P_{i+1}, P_i \cup \{r_{i+1}\})$ using the triangle inequality. In the following $\textsc{Off}_i$ refers to the configuration of offline algorithm $\textsc{Off}$ after the first i requests have been served.

$$\begin{aligned} & \mathit{dist}(P_{i+1}, P_i \cup \{r_{i+1}\}) \\ & \leq \mathit{dist}(P_i \cup \{r_{i+1}\}, \textsc{Off}_i \cup \{r_{i+1}\}) + \mathit{dist}(\textsc{Off}_i \cup \{r_{i+1}\}, \textsc{Off}_{i+1}) + \mathit{dist}(\textsc{Off}_{i+1}, P_{i+1}) \\ & \leq \eta_i + |\textsc{Off}_i| + \eta_{i+1}. \end{aligned}$$

Summing up over all rounds, and using that $\Phi_1 = \Phi_n = 0$ completes the proof of the theorem. □

5.4 The main theorem

Goal of this subsection is to prove Theorem 4.

Proof of Theorem 4. The main idea behind the proof is to show that we can apply Theorem 6 and virtually simulate the two algorithms (Follow-The-Prediction and the online algorithm of Raghvendra (2018)).

We need to show that we can assume that we are in some configuration and executing the respective algorithm, and that the switching cost between these configurations is upper bounded by the cost of the two algorithms. Similarly to the analysis of Follow-The-Prediction, we can virtually be in any configuration as long as we pay for the distance between any two consecutive configurations. When we currently simulate an algorithm A, the distance between the two consecutive configurations is exactly the cost of the edge that A introduces in this round. When we switch from the configuration of some algorithm A to the configuration of some algorithm B, then by Lemma 15, the distance between the two configurations is at most the total current cost of A and B.

This along with Theorem 18 (which is generalizing Theorem 6 beyond MTS and can be found in Appendix A) concludes the proof. □

5.5 Bipartite metric matching

Bipartite metric matching is the generalization of online matching on the line where the servers and requests can be points of any metric space. The problem is known to have a tight $(2n-1)$-competitive algorithm, due to Kalyanasundaram and Pruhs (1993) as well as Khuller et al. (1994).

We note that our arguments in this section are not line-specific and apply to that problem as well. This gives the following result:

Theorem 17. *There exists a deterministic algorithm for the online metric bipartite matching problem with predictions that attains a competitive ratio of*

$$\min\{2n-1, 9 + \frac{8e\eta}{\textsc{Off}}\},$$

against any offline algorithm Off.

6 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.

6.1 The Caching Problem

Datasets. For the sake of comparability, we used the same two datasets as Lykouris and 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 and 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`

Algorithm	Competitive ratio	Property	Reference
LRU	k		(Sleator and Tarjan, 1985)
Marker	$O(\log k)$	Robust	(Fiat et al., 1991)
FtP	$1 + 4\frac{\eta}{\textsc{Opt}}$	Consistent	Lemma 8
L&V	$2 + O(\min\{\sqrt{\frac{\eta'}{\textsc{Opt}}}, \log k\})$	Both	(Lykouris and Vassilvitskii, 2018)
RobustFtP	$(1+\epsilon)\min\{1 + 4\frac{\eta}{\textsc{Opt}}, O(\log k)\}$	Both	Theorem 2
LMarker	$O(1 + \min\{\log \frac{\eta'}{\textsc{Opt}}, \log k)\}$	Both	(Rohatgi, 2020)
LNonMarker	$O(1 + \min\{1, \frac{\eta'}{k \cdot \textsc{Opt}}\} \log k)$	Both	(Rohatgi, 2020)
Trust&Doubt	$O(\min\{1 + \log(1 + \frac{\eta}{\textsc{Opt}}), \log k\})$	Both	Theorem 3

Table 1: Summary of caching algorithms evaluated in experiments. Note that η and η' are different measures of prediction error, so their functions should not be compared directly.

- 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 and 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 and 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, whose competitive ratios are reported in Table 1. 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 and 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 $MIN^{rand}(\textsc{FtP}, \text{Marker})$, 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 $\{10^{-i} : i = 0, \dots, 4\}$, happen to be consistently the best choice in all our experimental settings.

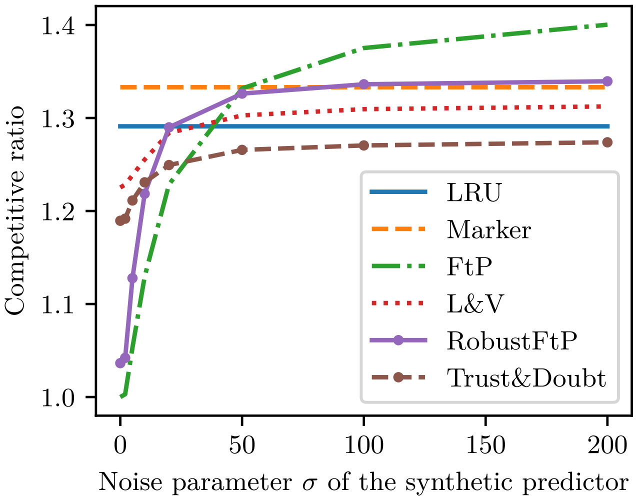

Figure 1: Comparison of caching algorithms augmented with synthetic predictions on the `BK` dataset.

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

Table 2: Competitive ratios of caching algorithms using PLECO, POPU, and LRU predictions on both datasets.

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. Figures 2 and 3 present the performance of all algorithms on the `BK` and `Citi` datasets, respectively. The experiment suggests that our algorithm Trust&Doubt outperforms previous prediction-based algorithms as well as LRU. In Table 2 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

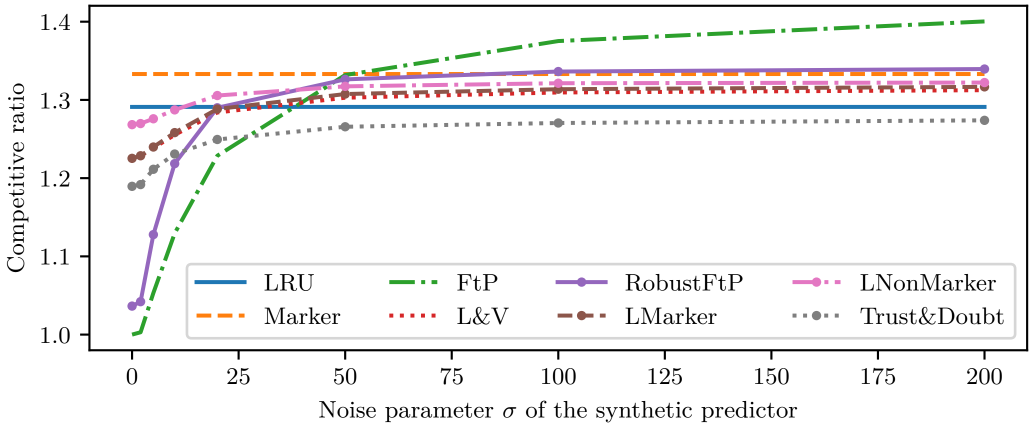

Figure 2: Comparison of caching algorithms augmented with synthetic predictions on the `BK` dataset.

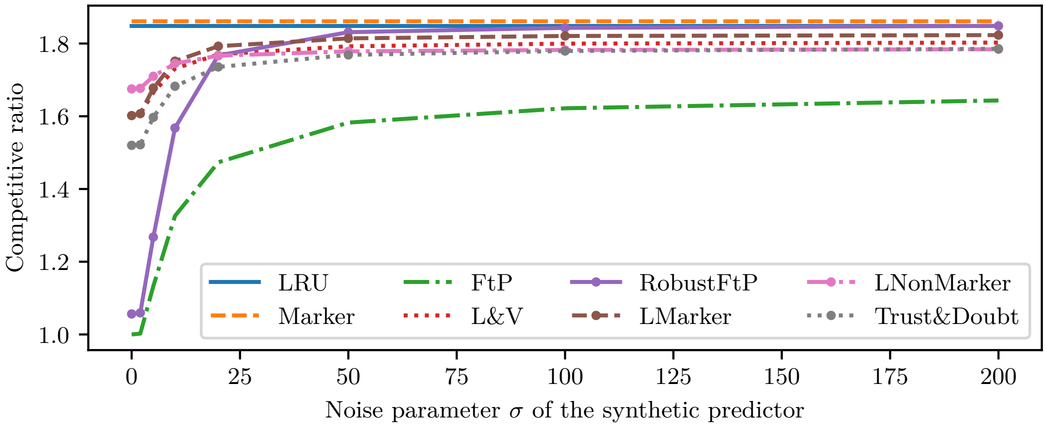

Figure 3: Comparison of caching algorithms augmented with synthetic predictions on the `Citi` dataset.

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 and Vassilvitskii (2018) also discuss how their algorithm framework performs when using LRU predictions, but did not provide both of these theoretical guarantees simultaneously.

6.2 A Simple MTS: the *Ice Cream* Problem

We consider a simple MTS example from Chrobak and 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 is motivated by an ice cream

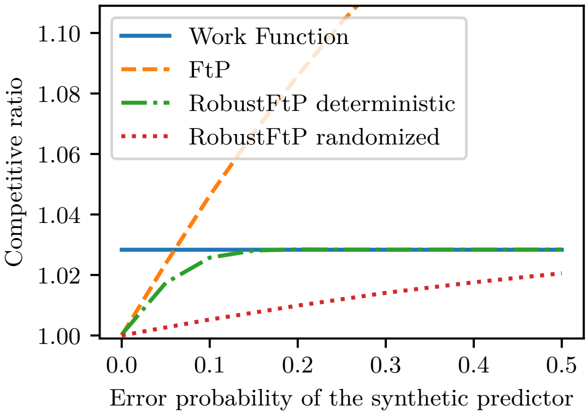

Figure 4: Performance on the ice cream problem with synthetic predictions.

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 and 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 4 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.

7 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.

References

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

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

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

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

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

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

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

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

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

S. Bubeck, M. B. Cohen, J. R. Lee, and Y. T. Lee. 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, Califormia, USA, January 6-9, 2019*, pages 89–97, 2019. doi: 10.1137/1.9781611975482.6.

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

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

T. H. Chung. Approximate methods for sequential decision making using expert advice. In *Proceedings of COLT'94*, COLT '94, pages 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.

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

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

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

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

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

A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young. Competitive paging algorithms. *J. Algorithms*, 12(4):685–699, 1991. doi: 10.1016/0196-6774(91)90041-V.

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

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

Y. Freund and R. E. Schapire. 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.

B. Fuchs, W. Hochstättler, and W. Kern. Online matching on a line. *Theor. Comput. Sci.*, 332 (1-3):251–264, 2005. doi: 10.1016/j.tcs.2004.10.028.

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

S. Irani, S. Shukla, and R. Gupta. 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.

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

B. Kalyanasundaram and K. Pruhs. Online weighted matching. *J. Algorithms*, 14(3):478–488, 1993. doi: 10.1006/jagm.1993.1026.

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

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

S. Khuller, S. G. Mitchell, and V. V. Vazirani. On-line algorithms for weighted bipartite matching and stable marriages. *Theor. Comput. Sci.*, 127(2):255–267, 1994. doi: 10.1016/0304-3975(94)90042-6.

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

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

J. R. Lee. Lower bounds for MTS. Lecture notes, 2018. URL https://tcsmath.github.io/online/2018/04/20/mts-lower-bounds/. Accessed: 02/02/2020.

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

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

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

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

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

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

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

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

S. Raghvendra. Optimal analysis of an online algorithm for the bipartite matching problem on a line. In *Proceedings of SoCG'18*, pages 67:1–67:14, 2018. doi: 10.4230/LIPIcs.SoCG.2018.67.

D. Rohatgi. 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, pages 1834–1845, 2020.

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

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

A Deterministic combination of a collection of algorithms

We consider a problem P and m algorithms $A_0, A_1, \ldots, A_{m-1}$ for this problem which fulfill the following requirements.

- $A_0, \ldots, A_{m-1}$ start at the same state and we are able to simulate the run of all of them simultaneously
- for two algorithms A_i and A_j, the cost of switching between their states is bounded by $\mathit{cost}(A_i) + \mathit{cost}(A_j)$.

Theorem 18 (Restated Theorem 6; generalization of Theorem 1 in Fiat et al. (1994))**.** *Given m on-line algorithms $A_0, \ldots A_{m-1}$ for a problem P which satisfy the requirements above, the algorithm MIN^{det} with parameter $1 < \gamma \leq 2$ incurs cost at most*

$$\left(\frac{2\gamma^m}{\gamma - 1} + 1\right) \cdot \min_i \{\mathit{cost}(A_i(I))\},$$

on any input instance I such that $\textsc{Opt}_I \geq 1$. If we choose $\gamma = \frac{m}{m-1}$, the coefficient $\frac{2\gamma^m}{\gamma-1} + 1$ equals 9 if $m = 2$ and can be bounded by $2em$.

Note that assumption on $\textsc{Opt}_I \geq 1$ is just to take care of the corner-case instances with very small costs. If we can only assume $\textsc{Opt}_I \geq c$ for some $0 < c < 1$, then we scale all the costs fed to MIN^{det} by $1/c$ and instances with $\textsc{Opt}_I = 0$ are usually not very interesting. The value of c is usually clear from the particular problem in hand, e.g., for caching we only care about instances which need at least one page fault, i.e., $\textsc{Opt}_I \geq 1$.

Proof. Let us consider the ℓ-th cycle of the algorithm and denote $i = \ell \bmod m$ and $i' = (\ell - 1) \bmod m$. We are switching from algorithm $A_{i'}$, whose current cost we denote $\mathit{cost}'(A_{i'}) = \gamma^{\ell-1}$ to A_i, whose current cost we denote $\mathit{cost}'(A_i)$, and its cost at the end of this cycle will become $\mathit{cost}(A_i) = \gamma^\ell$. Our cost during this cycle, i.e., for switching and for execution of A_i, is at most

$$\mathit{cost}'(A_{i'}) + \mathit{cost}'(A_i) + (\mathit{cost}(A_i) - \mathit{cost}'(A_i)) = \mathit{cost}'(A_{i'}) + \mathit{cost}(A_i) = \gamma^{\ell-1} + \gamma^\ell.$$

Now, let us consider the last cycle L, when we run the algorithm number $i = L \bmod m$. By the preceding equation, the total cost of MIN^{det} can be bounded as

$$\mathit{cost}(MIN^{det}) \leq 2 \cdot \sum_{\ell}^{L-1} \gamma^\ell + \mathit{cost}(A_i) = 2\frac{\gamma^L - 1}{\gamma - 1} + \mathit{cost}(A_i) \leq 2\frac{\gamma^L}{\gamma - 1} + \mathit{cost}(A_i).$$

If $L < m$, we use the fact that $\textsc{Opt} \geq 1$ and therefore the cost of each algorithm processing the whole instance would be at least one. Therefore, we have

$$\mathit{cost}(MIN^{det}) \leq 2\frac{\gamma^L}{\gamma - 1} + \gamma^L \leq 2\frac{\gamma^m}{\gamma - 1} \cdot \min_i\{\mathit{cost}(A_i)\},$$

because $\frac{\gamma^L}{\gamma-1} + \gamma^L = \frac{\gamma^{L+1}}{\gamma-1}$ and $L + 1 \leq m$.

Now, we have $L \geq m$, denoting $i = L \bmod m$, and we distinguish two cases.

(1) If $\min_j\{\mathit{cost}(A_j)\} = \mathit{cost}(A_i)$, then $\mathit{cost}(A_i) \geq \gamma^{L-m}$ for each i, and therefore

$$\frac{\mathit{cost}(MIN^{det})}{\min_j\{\mathit{cost}(A_j)\}} \leq \frac{2\frac{\gamma^L}{\gamma-1} + \mathit{cost}(A_i)}{\min_j\{\mathit{cost}(A_j)\}}$$

Note that $\mathit{cost}(A_i) \geq \gamma^{L-m}$, its cost from the previous usage. Since $\min_j\{\mathit{cost}(A_j)\} = \mathit{cost}(A_i)$, we get

$$\frac{\mathit{cost}(MIN^{det})}{\min_j\{\mathit{cost}(A_j)\}} \leq 2\frac{\gamma^m}{\gamma - 1} + 1.$$

(2) Otherwise, we have $\min_j\{cost(A_j)\} \geq \gamma^{L-m+1}$ and $cost(A_j) \leq \gamma^L$ and therefore

$$\frac{cost(MIN^{det})}{\min_j\{cost(A_j)\}} \leq 2\frac{\gamma^{m-1}}{\gamma-1} + \gamma^{m-1} \leq 2\frac{\gamma^m}{\gamma-1}.$$

For $\gamma = \frac{m}{m-1}$ we have

$$2\frac{\gamma^m}{\gamma-1} + 1 = 2(m-1)\left(\frac{m}{m-1}\right)^m + 1,$$

which equals 9 for $m = 2$ and can be bounded by $2em$.

□

B Limitations of the previous predictions for caching

In this section, we prove Theorem 5.

In previous prediction setups for the caching problem (Lykouris and Vassilvitskii, 2018; Rohatgi, 2020) the predictions are the time of the next request to each page. It is natural to try extending this type of predictions to other problems, such as weighted caching. In weighted caching each page has a weight/cost that is paid each time the page enters the cache. However, it turns out that even with perfect predictions of this type for weighted caching, one cannot improve upon the competitive ratio $\Theta(\log k)$, which can already be attained without predictions (Bansal et al., 2012). Our proof is based on a known lower bound for MTS on a so-called "superincreasing" metric (Karloff et al., 1994). Following a presentation of this lower bound by Lee (2018), we modify the lower bound so that the perfect predictions provide no additional information.

We call an algorithm for weighted caching *semi-online* if it is online except that it receives in each round, as an additional input, the time of the next request to each page (guaranteed to be without prediction error). We prove the following result:

Theorem (Restated Theorem 5). *Every randomized semi-online algorithm for weighted caching is* $\Omega(\log k)$*-competitive.*

Proof. Let $\tau > 0$ be some large constant. Consider an instance of weighted caching with cache size k and $k+1$ pages, denoted by the numbers $0, \dots, k$, and such that the weight of page i is $2\tau^i$. It is somewhat easier to think of the following equivalent *evader* problem: Let S_k be the weighted star with leaves $0, 1, \dots, k$ and such that leaf i is at distance τ^i from the root. A single evader is located in the metric space. Whenever there is a request to page i, the evader must be located at some leaf of S_k other than i. The cost is the distance traveled by the evader. Any weighted caching algorithm gives rise to the evader algorithm that keeps its evader at the one leaf that is *not* currently in the algorithm's cache. The cost between the two models differs only by an additive constant (depending on k and τ).

For $h = 1, \dots, k$ and a non-empty time interval (a, b), we will define inductively a random sequence $\sigma_h = \sigma_h(a, b)$ of requests to the leaves $0, \dots, h$, such that each request arrives in the time interval (a, b) and

$$A_h \geq 4\alpha_{h-1}\tau^h \geq \alpha_h \cdot \textsc{Opt}_h, \tag{8}$$

where A_h denotes the expected cost of an arbitrary semi-online algorithm to serve the random sequence σ_h while staying among the leaves $0, \dots, h$, $\textsc{Opt}_h$ denotes the expected optimal offline cost of doing so with an offline evader that starts and ends at leaf 0, $\alpha_0 = \frac{1+\tau}{4\tau}$, and $\alpha_h = 1/2 + \beta \log h$ for $h \geq 1$, where $\beta > 0$ is a constant determined later. The inequality between the first and last term in (8) implies the theorem. We will also ensure that $(0, 1, \dots, h)$ is both a prefix and a suffix of the sequence of requests in σ_h.

For the base case $h = 1$, the inequality is satisfied by the request sequence σ_1 that requests first 0 and then 1 at arbitrary times within the interval (a, b).

For $h \geq 2$, the request sequence σ_h consists of subsequences (iterations) of the following two types (we will only describe the sequence of request locations for now and later how to choose the exact arrival times of these requests): A *type 1 iteration* is the sequence $(0, 1, \ldots, h)$. A *type 2 iteration* is the concatenation of $\lceil \frac{\tau^h}{\alpha_{h-1} \textsc{Opt}_{h-1}} \rceil$ independent samples of a random request sequence of the form σ_{h-1}. The request sequence σ_h is formed by concatenating $\lceil 8\alpha_{h-1} \rceil$ iterations, where each iteration is chosen uniformly at random to be of type 1 or type 2. If the last iteration is of type 2, an additional final request at h is issued. Thus, by induction, $(0, \ldots, h)$ is both a prefix and a suffix of σ_h.

We next show (8) under the assumption that at the start of each iteration, the iteration is of type 1 or 2 each with probability 1/2 even when conditioned on the knowledge of the semi-online algorithm at that time. We will later show how to design the arrival times of individual requests so that this assumption is satisfied. We begin by proving the first inequality of (8). We claim that in each iteration of σ_h, the expected cost of any semi-online algorithm (restricted to staying at the leaves $0, \ldots, h$) is at least $\tau^h/2$. Indeed, if the evader starts the iteration at leaf h, then with probability 1/2 we have a type 1 iteration forcing the evader to vacate leaf h for cost τ^h, giving an expected cost of $\tau^h/2$. If the evader is at one of the leaves $0, \ldots, h-1$, then with probability 1/2 we have a type 2 iteration. In this case, it must either move to h for cost at least τ^h, or $\lceil \frac{\tau^h}{\alpha_{h-1} \textsc{Opt}_{h-1}} \rceil$ times it suffers expected cost at least $\alpha_{h-1} \textsc{Opt}_{h-1}$ by the induction hypothesis. So again, the expected cost is at least $\tau^h/2$. Since σ_h consists of $\lceil 8\alpha_{h-1} \rceil$ iterations, we have

$$A_h \geq 4\alpha_{h-1}\tau^h,$$

giving the first inequality of (8).

To show the second inequality of (8), we describe an offline strategy. With probability $2^{-\lceil 8\alpha_{h-1} \rceil}$, all iterations of σ_h are of type 2. In this case, the offline evader moves to leaf h at the beginning of σ_h and back to leaf 0 upon the one request to h at the end of σ_h, for total cost $2(1 + \tau^h)$. With the remaining probability, there is at least one type 1 iteration. Conditioned on this being the case, the expected number of type 1 iterations is $\lceil 8\alpha_{h-1} + 1 \rceil/2$, and the expected number of type 2 iterations is $\lceil 8\alpha_{h-1} - 1 \rceil/2$. The offline evader can serve each type 1 iteration for cost $2(1 + \tau)$ and each type 2 iteration for expected cost $\lceil \frac{\tau^h}{\alpha_{h-1} \textsc{Opt}_{h-1}} \rceil \textsc{Opt}_{h-1}$, and it finishes each iteration at leaf 0. (Thus, if the last iteration is of type 2, then the final request to h incurs no additional cost.) By the induction hypothesis, $\textsc{Opt}_{h-1} \leq O(\tau^{h-1})$. Hence, we can rewrite the expected cost of a type 2 iteration as

$$\left\lceil \frac{\tau^h}{\alpha_{h-1} \textsc{Opt}_{h-1}} \right\rceil \textsc{Opt}_{h-1} = (1 + o(1)) \frac{\tau^h}{\alpha_{h-1}},$$

as $\tau \to \infty$. Since $h \geq 2$, the expected cost of all type 1 iterations is only an $o(1)$ fraction of the expected cost of the type 2 iterations. Overall, we get

$$\begin{aligned}
\textsc{Opt}_h &\leq 2^{-\lceil 8\alpha_{h-1} \rceil} 2(1 + \tau^h) + \\
&\qquad (1 + o(1)) \left(1 - 2^{-\lceil 8\alpha_{h-1} \rceil}\right) \frac{\lceil 8\alpha_{h-1} - 1 \rceil}{2} \frac{\tau^h}{\alpha_{h-1}} \\
&\leq (1 + o(1)) \left[2^{-\lceil 8\alpha_{h-1} \rceil} 2\tau^h + \left(1 - 2^{-\lceil 8\alpha_{h-1} \rceil}\right) 4\tau^h\right] \\
&= (1 + o(1)) \left(1 - 2^{-\lceil 8\alpha_{h-1} \rceil - 1}\right) 4\tau^h \\
&\leq \frac{4\tau^h}{1 + 2^{-\lceil 8\alpha_{h-1} \rceil - 1}}.
\end{aligned}$$

We obtain the second inequality in (8) by

$$\begin{aligned}
\frac{4\alpha_{h-1}\tau^h}{\textsc{Opt}_h} &\geq \alpha_{h-1}\left(1 + 2^{-\lceil 8\alpha_{h-1}\rceil - 1}\right) \\
&\geq \frac{1}{2} + \beta\log(h-1) + 2^{-8\beta\log(h-1)-7} \\
&\geq \frac{1}{2} + \beta\log(h-1) + \frac{\beta}{h-1} \\
&\geq \frac{1}{2} + \beta\log h \\
&= \alpha_h,
\end{aligned}$$

where the third inequality holds for $\beta = 2^{-7}$.

It remains to define to define the arrival times for the requests of sequence σ_h within the interval (a, b). We do this as follows: Let $m \geq 1$ be the number of requests to leaf h in σ_h. These requests to h will be issued at times $a + (b-a)\sum_{i=1}^{j} 2^{-i}$ for $j = 1, \ldots, m$.

To define the arrival times of the other requests, we will maintain a time variable $c \in [a, b)$ indicating the *current* time, and a variable $n > c$ indicating the time of the *next* request to leaf h after time c. Initially, $c := a$ and $n := (a+b)/2$. Consider the first iteration for which the arrival times have not been defined yet. If the iteration is of type 2, we choose the arrival times according to the induction hypothesis so that all subsequences σ_{h-1} within the iteration fit into the time window $(c, (c+n)/2)$, and we update $c := (c+n)/2$. If the iteration is of type 1, sample a type 2 iteration I and let $t_1, \ldots, t_{h-1}$ be such that t_i would be the time of the next request to page i if the next iteration were this iteration I of type 2 instead of a type 1 iteration. We define the arrival times of the (single) request to leaf $i < h$ in this type 1 iteration to be t_i. If this was not the last iteration, we update $c := n$ and increase n to the time of the next request to h (as defined above).

Notice that at the beginning of each iteration within σ_h, ordering the pages by the time of their next request always yields the sequence $0, 1, \ldots, k$, and the time of the next request to each page is independent of whether the next iteration is of type 1 or type 2. Thus, as promised, whether the next iteration is of type 1 or type 2 is independent of the knowledge of the semi-online algorithm. □