# Double Coverage with Machine-Learned Advice

**Alexander Lindermayr**
Faculty of Mathematics and Computer Science, University of Bremen, Germany

**Nicole Megow**
Faculty of Mathematics and Computer Science, University of Bremen, Germany

**Bertrand Simon**
IN2P3 Computing Center, CNRS, Villeurbanne, France

## Abstract

We study the fundamental online $k$-server problem in a learning-augmented setting. While in the traditional online model, an algorithm has no information about the request sequence, we assume that there is given some advice (e.g. machine-learned predictions) on an algorithm's decision. There is, however, no guarantee on the quality of the prediction and it might be far from being correct.

Our main result is a learning-augmented variation of the well-known Double Coverage algorithm for $k$-server on the line (Chrobak et al., SIDMA 1991) in which we integrate predictions as well as our trust into their quality. We give an error-dependent competitive ratio, which is a function of a user-defined confidence parameter, and which interpolates smoothly between an optimal consistency, the performance in case that all predictions are correct, and the best-possible robustness regardless of the prediction quality. When given good predictions, we improve upon known lower bounds for online algorithms without advice. We further show that our algorithm achieves for any $k$ an almost optimal consistency-robustness tradeoff, within a class of deterministic algorithms respecting *local* and *memoryless* properties.

Our algorithm outperforms a previously proposed (more general) learning-augmented algorithm. It is remarkable that the previous algorithm crucially exploits memory, whereas our algorithm is *memoryless*. Finally, we demonstrate in experiments the practicability and the superior performance of our algorithm on real-world data.

**2012 ACM Subject Classification** Theory of computation → Design and analysis of algorithms; Mathematics of computing → Mathematical optimization

**Keywords and phrases** online k-server problem, competitive analysis, learning-augmented algorithms, untrusted predictions, consistency, robustness

**Digital Object Identifier** 10.4230/LIPIcs.ITCS.2022.99

**Related Version** *Full Version*: https://arxiv.org/abs/2103.01640

**Funding** Partially supported by the German Science Foundation (DFG) under contracts ME 3825/1 and 146371743 – TRR 89 Invasive Computing.

## 1 Introduction

The $k$-server problem is one of the most fundamental online optimization problems. Manasse et al. [36, 35] introduced it in 1988 as a generalization of other online problems, such as the prominent paging problem, and since then, it has been a corner stone for developing new models and techniques. We follow this line and investigate the $k$-server problem in the recently evolving framework of learning-augmented online computation.

We consider the *$k$-server problem on the line*, in which there are given $k$ distinct servers $s_1, \ldots, s_k$ located at initial positions on the real line. A sequence of requests $r_1, \ldots, r_n \in \mathbb{R}$ is revealed online one-by-one, that is, an algorithm only knows the current (unserved) request, serves it and only then sees the next request; it has no knowledge about future requests. To serve a request, (at least) one of the servers has to be moved to the requested point. The cost of serving a request is defined as the distance traveled by the server(s). The task is to give an online strategy of minimum total cost for serving a request sequence.

In standard competitive analysis, an online algorithm $\mathcal{A}$ is called *$\mu$-competitive* if for every instance $I$, there is some constant $c$ depending only on the initial configuration such that $\mathcal{A}(I) \leq \mu \cdot \textsc{Opt}(I) + c$, where $\mathcal{A}(I)$ denotes the cost of $\mathcal{A}$ on $I$ whereas $\textsc{Opt}(I)$ is the cost of an optimal solution that can be obtained when having full information about $I$ in advance.

Manasse et al. [36] gave a strong lower bound which rules out any deterministic online algorithm with a competitive ratio better than $k$. They also stated the famous *$k$-server conjecture* in which they conjecture that there is a $k$-competitive online algorithm for the $k$-server problem in any metric space and for any $k$. The conjecture has been proven to be true for special metric spaces such as the line [16], considered in this paper, the uniform metric space (paging problem) [43] and tree metrics [17]. For the $k$-server problem on the line, Chrobak et al. [16] devised the DoubleCoverage algorithm and proved a best possible competitive ratio $k$. For a given request, DoubleCoverage moves the (at most) two adjacent servers towards the requested point until the first of them reaches that point.

The past decades have witnessed a rapid advancement of machine learning (ML) methods, which nowadays can be expected to predict often – but not always – uncertain data with good accuracy. The lack of guarantees on the predictions and the need for trustable performance guarantees lead to the area of *learning-augmented online algorithms*. This recently emerging research area investigates online algorithms that have access to predictions, e.g., on parts of the instance or the algorithm's execution, while not making any assumption on the quality of the predictions. Formally, we assume that a prediction has a certain quality $\eta \geq 0$. In the context of learning theory one may think of the *loss* of a prediction with respect to the ground truth. Accordingly, $\eta = 0$ refers loosely speaking to the case where the prediction was correct. In the field of learning-augmented algorithm this quantity is called *prediction error*. An algorithm does not know what quality a prediction has, but we can use it in the analysis to measure an algorithm's performance depending on $\eta$. If a learning-augmented algorithm is $\mu(\eta)$-competitive for some function $\mu$, we say that the algorithm is *$\alpha$-consistent* if $\alpha = \mu(0)$ and *$\beta$-robust* if $\mu(\eta) \leq \beta$ for any prediction with prediction error $\eta$ [40].

Very recently, Antoniadis et al. [2] proposed learning-augmented online algorithms for general metrical task systems, a generalization of our problem. Their algorithm relies on simulating several online algorithms in parallel and keeping track of their solutions and cost. This technique crucially employs additional *memory* which can be a serious drawback in practice when decisions must be made without access to the history.

In this work, we introduce **memory-constrained** learning-augmented algorithms for the $k$-server problem on the line. An algorithm $\mathcal{A}$ is intuitively *memory-constrained*, if the decision for the next move of $\mathcal{A}$ only depends on the current situation (server positions, request and prediction). It is especially independent of previous requests. However, as the algorithm is allowed to move a server to any point of the real line, it could use its position to encode any information at a negligible cost. This issue is often addressed by forbidding algorithms to move several servers per request (hence, restricting to so-called *lazy* algorithms) which leads to the classical *memoryless* property, although variations of this definition exist [24]. A downside of this restriction is that deterministic memoryless algorithms cannot be competitive, and there is no distinction between the type of information gathered by DoubleCoverage and unconstrained information encoding. This difference has been nevertheless acknowledged by informally considering DoubleCoverage as memoryless [22], although noting immediately that such a definition for a non-lazy algorithm is cumbersome.

In order to allow the behavior of DoubleCoverage, we formally define *memory-constrained* algorithms as algorithms allowed to move several servers, making decisions independently of previous requests, but with an *erasable* memory: for any set of $k$ distinct points and any starting configuration, there exists a finite sequence of requests among these $k$ points after which each point contains exactly one server. We will refer to such a sequence as a *force* to these $k$ points. This definition is quite general as it allows to pre-move some servers as DoubleCoverage does, and even allows information encoding, but provides a possibility to erase any information gathered. The algorithms we design will not abuse information encoding, but our lower bounds will hold in this context.

## Further Related Work

The past few years have exhibited several demonstrations of the power of learning-augmented algorithms improving on traditional online algorithms. Studied online problems include caching [33, 41, 2, 45], ski rental [40, 19, 44, 46, 8], scheduling [40, 6, 39, 27, 46, 20, 5], secretary problems [4, 18], matching [28, 26], sorting [32], online covering [7], and possibly more by now. Learning-augmented algorithms have proven to be successful also in other areas, e.g., to speed up search queries [25], in revenue optimization [37], and bloom filters [38].

More than a decade ago, Mahdian et al. [34] demonstrated performance improvements for online allocation algorithms when there is access to an accurate solution estimation. They further bounded the case where the estimation is inaccurate. While these bounds essentially correspond to consistency and robustness, they did not precisely measure the prediction quality. Yet they introduced a parameter to express the tradeoff between both bounds. In the recent field of learning-augmented algorithms, Kumar et al. [40] initiated the use of a similar parameter $\lambda \in [0, 1]$. It can be interpreted as an algorithm's indicator of trust in the given predictions: smaller $\lambda$ indicates stronger trust and gives a higher priority to a better consistency at the cost of a worse robustness, and vice versa. Such parameterized consistency-robustness tradeoff has become standard for expressing the performance of learning-augmented algorithms when aiming for constant factors [40, 44, 46, 1, 8, 7, 4, 20].

As mentioned, Antoniadis et al. [2] provide a general learning-augmented framework for any metrical task systems which includes the $k$-server problem. Applied to the line metric, they devise a learning-augmented algorithm that crucially requires memory and obtains a 9-consistent and $9k$-robust algorithm.

The $k$-server problem has been studied also in the context of reinforcement learning (RL), originating at [21] and including hierarchical RL learning [29] as well as deep RL learning [31].

The classical online $k$-server problem without access to predictions has been studied extensively, also in general metric spaces. The best known deterministic algorithm is the WorkFunction algorithm [23] with a competitive ratio of $2k - 1$. For several special metric spaces there are even tighter bounds known for this algorithm [10, 47]. When allowing randomization, a $\Omega(\log k/ \log \log k)$ lower bound holds [9] and a $(\log k)^{\mathcal{O}(1)}$-competitive randomized algorithm is conjectured [22]. Restricting further to memoryless randomized algorithms increases the lower bound on the competitive ratio exponentially to $k$ [22] and some recent efforts focus on a more general variant in this setting [14].

The power of DoubleCoverage goes beyond its optimality for the $k$-server problem in tree metrics [17]. Recently, Buchbinder et al. [13] showed that it is a best possible deterministic algorithm for the more general $k$-taxi problem, even in general metric spaces using an embedding into hierarchically separated trees.

## Our Contribution

We design learning-augmented memory-constrained online algorithms for the $k$-server problem on the line. Firstly, we define some more notation and the precise prediction model. We denote a server's name as well as its position on the line by $s_i$, for $i \in \{1, 2 \ldots, k\}$. A configuration $C_t = (s_1, \ldots, s_k) \in \mathbb{R}^k$ is a snapshot of the server positions at a certain point in time. For a given instance, a $k$-server algorithm outputs a sequence of configurations $C_1, \ldots, C_n$ (also called *schedule*) such that for every $t = 1, \ldots, n$, we have $r_t \in C_t$. We denote the initial configuration by $C_0$. The objective function can be expressed as $\sum_{t=1}^{n} d(C_{t-1}, C_t)$, where $d(C_{t-1}, C_t)$ denotes the cost for moving the servers from $C_{t-1}$ to $C_t$. We assume w.l.o.g. $s_1 \leq \ldots \leq s_k$, as server overtakings can be uncrossed without increasing the total cost.

We employ a prediction model that predicts algorithmic choices of an optimal algorithm, that is predicting which server should serve a certain request. Given an instance $I$ composed of the request sequence $r_1, \ldots, r_n$, we define a *prediction* for $I$ as a sequence of indices $p_1, \ldots, p_n$ from the set $\{1, \ldots, k\}$. If $s_1, \ldots, s_k$ are the servers of some learning-augmented algorithm, we call $s_{p_t}$ the *predicted server* for the $t$-th request. We call the algorithm that simply *follows the predictions* FtP, that is, it serves each request by the predicted server (to simplify computations, we still remove overtakings as mentioned above, which is equivalent to relabel servers by their position order). We denote its cost by FtP$(I)$. We define the *prediction error* $\eta = \text{FtP}(I) - \text{Opt}(I)$ as quality measure for our predictions. Note that this error definition is independent of our algorithm.

Our main result is a parameterized algorithm for the $k$-server problem on the line with an error-dependent performance guarantee that – when having access to good-quality predictions – beats the known lower bound for deterministic online algorithms.

▶ **Theorem 1.** *Let $\lambda \in [0, 1]$. We define $\beta(k) = \sum_{i=0}^{k-1} \lambda^{-i}$, for $\lambda > 0$, and $\beta(k) = \infty$, for $\lambda = 0$. Further, let*

$$\alpha(k) = \begin{cases} 1 + 2\lambda + 2\lambda^2 + \ldots + 2\lambda^{(k-1)/2} & \text{if } k \text{ is odd} \\ 1 + 2\lambda + 2\lambda^2 + \ldots + 2\lambda^{k/2-1} + \lambda^{k/2} & \text{if } k \text{ is even.} \end{cases}$$

*Let $\eta$ denote the total prediction error and* Opt *the cost of an optimal solution. Then, there exists a learning-augmented memory-constrained online algorithm for the $k$-server problem on the line with a competitive ratio of at most*

$$\min\left\{\alpha(k)\left(1 + \frac{\eta}{\text{Opt}}\right), \beta(k)\right\}.$$

*In particular, the algorithm is $\alpha(k)$-consistent and $\beta(k)$-robust, for $\lambda > 0$.*

Interpreting both bounds as functions of $\lambda \in [0, 1]$ illustrates that $\alpha(k)$ interpolates monotonously between 1 and $k$ while $\beta(k)$ grows from $k$ as $\lambda$ decreases. This matches our expectation on a learning-augmented online algorithm, as it improves in consistency but loses in robustness compared to the best possible online algorithm. From another perspective, for a fixed value of $\lambda$, $\alpha(k)$ is bounded by a constant (equal to $1 + \frac{2}{1-\lambda}$) which highlights the algorithm consistency but this comes at the price of an exponential dependency on $k$ for $\beta(k)$.

To show this result, we design an algorithm that carefully balances between (i) the wish to simply follow the predictions (FtP) which is obviously optimal if the predictions are correct, i.e. is 1-consistent, and (ii) the best possible online algorithm when not having access to (good) predictions DoubleCoverage [16], which is $k$-robust. An additional challenge is to preserve the memory-constrained property. We achieve this, by generalizing the classical DoubleCoverage [16] in an intuitive way. Essentially, our algorithm LambdaDC includes the information about predicted servers and our trust into them by varying server speeds.

The analysis of our algorithm is tight. On the technical side, our analysis builds on the powerful *potential function method*, as does the analysis of the classical DoubleCoverage [16]. While LambdaDC is quite simple (a precise definition follows), the analysis is much more intricate and requires a careful re-design for the learning-augmented setting. Our main technical contribution is the definition and analysis of different parameterized potential functions for proving robustness and consistency, that capture the different speeds for moving servers and the accordingly more difficult tracing of the server moves.

We remark that our performance bound also holds [30] (with an additional factor of 2 on the error) using the error measure of Antoniadis et al. [2]. Their error definition sums up the distances between the configurations of Opt and FtP after every request, thus, it may seem more intuitive as server positions are compared instead of solution costs. However, our error definition allows to establish learnability results and also simplifies some analyses.

While our result is tailored to the $k$-server problem, the framework by Antoniadis et al. [2] is designed for more general metrical task systems. Interestingly, one of their methods is a deterministic combination of DoubleCoverage and FtP, we refer to it as FtP&DC. It is shown that FtP&DC is 9-consistent and $9k$-robust. Our methods differ substantially. While FtP&DC carefully tracks states and costs of the simulated individual algorithms, LambdaDC is a simple algorithm that only requires knowledge of the current configuration. Further, LambdaDC has a better performance for $k < 20$ and an appropriate parameter $\lambda$ (e.g., $k = 19$ and $\lambda = 0.83$), but does not offer such a good tradeoff for larger $k$. Actually, this is unavoidable for a certain class of memory-constrained algorithms, that includes LambdaDC.

Indeed, we complement our main result with an almost matching lower bound on the consistency-robustness tradeoff. We construct a non-trivial bound for the class of memory-constrained algorithms that satisfy an additional locality property; its precise definition is formulated in Section 5. Intuitively, the locality property enforces an algorithm to achieve a better competitive ratio for a subinstance served by fewer servers. Other locality restrictions have been required before to establish lower bounds, e.g., for matching on the line, see [3].

▶ **Theorem 2.** *Let $\lambda \in (0,1]$, $\rho(k) = \sum_{i=0}^{k-1} \lambda^i$ and $\beta(k) = \sum_{i=0}^{k-1} \lambda^{-i}$. Let $\mathcal{A}$ be a learning-augmented locally-consistent and memory-constrained deterministic online algorithm for the $k$-server problem on the line. Then, if $\mathcal{A}$ is $\rho(k)$-consistent, it is at least $\beta(k)$-robust.*

Algebraic transformations (see Appendix B) show that $\alpha(k) < 2\rho(k)$, which implies that LambdaDC achieves a tradeoff within a factor of at most 2 of the *optimal* consistency-robustness tradeoff (among locally-consistent and memory-constrained algorithms). For $k = 2$, LambdaDC achieves the optimal tradeoff (among memory-constrained algorithms).

We demonstrate the power of our approach in empirical experiments on real-world data. We show that for a reasonable choice of $\lambda$ our method outperforms the classical online algorithm DoubleCoverage as well as the algorithm in [2] for nearly all prediction errors.

Finally, we address the learnability of our predictions, even though this is not the focus of our work. We show that a *static* prediction sequence is PAC-learnable. We show a bound on the sample complexity that is polynomial in the number of requests, $n$, and the number of servers, $k$, and we give a learning algorithm with a polynomial running time in $n, k$ and the number of samples.

## 2 Algorithm and Roadmap for the Analysis

### The Algorithm LambdaDC

We generalize the classical DoubleCoverage [16] by including the information about predicted servers as well as our trust into this advice, in an intuitive way. If a request $r_t$ appears between two servers, the one closer to the predicted server $p_t$ moves by a greater distance towards the request – as if it traveled at a higher speed.

Formally, we define LambdaDC for a given $\lambda \in [0, 1]$ as follows. If $r_t < s_1$ or $r_t > s_k$, then LambdaDC only moves the closest server. Otherwise, we have $s_i < r_t < s_{i+1}$. If $p_t \leq i$, then LambdaDC moves $s_i$ with speed 1 and $s_{i+1}$ with speed $\lambda$ towards $r_t$ until one server reaches the request. If $p_t \geq i+1$, the speeds of $s_i$ and $s_{i+1}$ are swapped. Hence, LambdaDC equals FtP (with shortcuts) for $\lambda = 0$, and DoubleCoverage for $\lambda = 1$. Using nonintegral values for $\lambda$ gives an algorithm that interpolates between both.

### Potential Function Analysis

The analysis of our algorithm builds on the powerful *potential function method*, as does the analysis of the classical DoubleCoverage [16].

Our potential analysis follows the well-known *interleaving moves* technique [11]. To compare two algorithms $\mathcal{A}$ and $\mathcal{B}$ in terms of competitiveness, we simulate both in parallel on some instance $I$. Then, we employ a potential function $\Phi$ which maps at every time $t$ the state of both algorithms (i.e. the algorithms current configurations) to a value $\Phi_t \geq 0$, the potential at time $t$. We define $\Delta\Phi_t = \Phi_t - \Phi_{t-1}$. Let $\Delta\mathcal{B}_t(I)$ resp. $\Delta\mathcal{A}_t(I)$ denote the cost $\mathcal{A}$ resp. $\mathcal{B}$ charges for serving the request at time $t$ and let $\mu > 0$. For every request $r_t$, we assume that first $\mathcal{B}$ serves the request, and second $\mathcal{A}$. If

**(i)** the move of $\mathcal{B}$ increases $\Phi$ by at most $\mu \cdot \Delta\mathcal{B}_t(I)$, whereas

**(ii)** the move of $\mathcal{A}$ decreases $\Phi$ by at least $\Delta\mathcal{A}_t(I)$,

we can use a telescoping sum argument to conclude $\mathcal{A}(I) \leq \mu \cdot \mathcal{B}(I) + \Phi_0$. Note that if $\mathcal{B}$ is the optimal algorithm, $\mu$ is equal to the competitive ratio of $\mathcal{A}$ since $\Phi_0$ only depends on $C_0$.

To show an error-dependent competitive ratio in the learning-augmented setting, we follow three steps. We show first that the cost of LambdaDC is close to the cost of FtP, that is $\text{Alg}(I) \leq \alpha(k) \cdot \text{FtP}(I) + c$ for some $c > 0$ and for every instance $I$. Note that this corresponds to the consistency case as FtP is the optimal algorithm if $\eta = 0$. Second we plug in the definition of our prediction error $\eta$ to bound the cost of FtP by the cost of the fixed optimal solution (fixed with respect to the definition of $\eta$) and $\eta$. Combining both results yields the first part of the competitive ratio of Theorem 1. Lastly we prove a robustness bound, i.e. a general bound independent of the prediction, on the cost of LambdaDC with respect to Opt. All additive constants in the competitive ratios only depend on the initial configuration of the servers, being zero if all servers start at the same position.

The potential functions we use to analyze LambdaDC are inspired by the potential function in the classical analysis of DoubleCoverage [16]. It is composed of a *matching part* $\Psi$, summing the distances between the server positions of an algorithm and the reference algorithm (Opt, FtP) and a *spreadness part* $\Theta$, summing the distances between an algorithms server positions. To incorporate the more sophisticated server moves at different speeds, we introduce multiplicative coefficients to both parts. The main technical contribution lies in identifying the proper weights and performing the much more involved analysis.

### Lower Bounds for LambdaDC

Our analysis is tight w.r.t the given bounds on consistency and robustness.

▶ **Lemma 3.** *LambdaDC is at least $\alpha(k)$-consistent and $\beta(k)$-robust.*

**Proof.** We give two separate instances for consistency and robustness.

**(i)** Consider $k$ servers initially at positions $0, 1, -1, 2, -2, \dots$ and the request sequence of length $k+1$ at positions $0.5, 0, 1, -1, 2, -2, \dots$. There is a solution of cost 1 that only moves the server that is initially at 0.
LambdaDC serves the first request by moving the optimal server from 0 to 0.5 and additionally the one from 1 to $1-\lambda/2$. With the second request, the first server is moved back to 0, having moved a total distance of 1, and the server from $-1$ moves to $-1+\lambda/2$. For the third request, the server from original position 1 returns to this position, etc. Each server moves back to its initial position $i$ after moving a total distance of $\lambda^{|i|}$. Repeating this example gives the lower bound on the consistency.

**(ii)** Consider $k$ servers initially at positions $\beta(i) = \sum_{i=0}^{k-1} \lambda^{-i}$, for $i \in \{1, \dots, k\}$, and the request sequence of length $k+1$ at positions $0$, $\beta(1)$, $\beta(2)$, $\dots$, $\beta(k)$. There is a solution of cost 2 that only moves the server that is initially at 1. Consider predictions corresponding always to the rightmost server at the highest position.
LambdaDC serves the second request by moving both servers from 0 and $\beta(2)$ to $\beta(1)$ as the closest server moves by a distance of 1 and the furthest server, which is predicted, moves by a distance of $1/\lambda$. Similarly, for each request except the last one, both servers neighboring the request end up serving the request simultaneously. So the $i$-th server moves by a total distance of $2/\lambda^{i-1}$. Repeating this example gives the lower bound on the robustness. ◀

### Organization of the Paper

For ease of exposition, we first consider the setting of 2 servers in Section 3. Then, we extend the techniques to the general setting in Sections 4 and 5 while maintaining the same structure as for $k = 2$. We illustrate and discuss the results of computational experiments in Section 6, and, finally, talk about PAC learnability of our predictions in Section 7.

## 3 Full Analysis for Two Servers

### 3.1 Error-dependent Competitive Ratio of LambdaDC

We show the theoretical guarantees of LambdaDC claimed in Theorem 1 restricted to two servers. We denote the cost of LambdaDC for some instance $I$ by $\textsc{Alg}(I)$, and the cost for serving a request $r_t$ by $\Delta\textsc{Alg}_t(I)$. If $t$ is clear from the context then we omit the index.

▶ **Theorem 4.** *For any parameter $\lambda \in [0,1]$, LambdaDC has a competitive ratio of at most*

$$\min\left\{(1+\lambda)\left(1+\frac{\eta}{\textsc{Opt}}\right), 1+\frac{1}{\lambda}\right\}.$$

*Thus, it is $(1+\lambda)$-consistent and $(1+1/\lambda)$-robust.*

We follow the three-step approach outlined in the previous section. The definition of $\eta$ immediately gives for any instance $I$ and prediction with error $\eta$ that $\textsc{FtP}(I) = \textsc{Opt}(I) + \eta$. With Lemmas 5 and 6 this implies Theorem 4. We firstly compare the algorithm to $\textsc{FtP}$.

▶ **Lemma 5.** *For any instance $I$ and $\lambda \in [0,1]$, there is some $c \geq 0$ that only depends on the initial configuration such that* $\textsc{Alg}(I) \leq (1+\lambda) \cdot \textsc{FtP}(I) + c$.

**Proof.** Let $I$ be an arbitrary instance and let servers start at positions $s_1^0$ and $s_2^0$. If $\lambda = 0$, LambdaDC only shortcuts FtP's moves, hence $\textsc{Alg}(I) \leq \textsc{FtP}(I)$. Now assume that $\lambda > 0$. Let $s_1, s_2$ be LambdaDC's servers and $x_1', x_2'$ be FtP's servers. We simulate $I$ in parallel for both algorithms. At every time $t$, we map the configurations of both algorithms to a non-negative value using the potential function

$$\Phi \quad = \quad \underbrace{\frac{1+\lambda}{\lambda}\left(|s_1 - x_1'| + |s_2 - x_2'|\right)}_{\Psi \text{ (matching part)}} \quad + \quad \underbrace{|s_1 - s_2|}_{\Theta \text{ (spreadness part)}}.$$

Suppose that a new request arrives. First, FtP serves the request. Assume that $x_1'$ moves and charges cost $\Delta\textsc{FtP}$. Since LambdaDC remains in its previous configuration, $|x_1' - s_1|$ increases by at most $\Delta\textsc{FtP}$, and $\Phi$ increases by at most $(1+\lambda)/\lambda \cdot \Delta\textsc{FtP}$. Second, LambdaDC moves. Assume by scaling the instance that the algorithm serves the request after exactly one time unit, i.e., the fast server moves distance 1 and the slow server distance $\lambda$. We distinguish whether the request is between the algorithm's servers or not, and prove in each case that $\Phi$ decreases by at least $1/\lambda \cdot \Delta\textsc{Alg}$.

**(a)** Suppose the request is not between the servers $s_1$ and $s_2$; say, it is left of $s_1$. Then LambdaDC moves only $s_1$ and $\Delta\textsc{Alg} = 1$. Either $x_1'$ or $x_2'$ covers the request, hence moving $s_1$ decreases $\Psi$ by $(1+\lambda)/\lambda$ while it increases $\Theta$ by 1. Thus,

$$\Delta\Phi \leq -\frac{1+\lambda}{\lambda} + 1 = -\frac{1}{\lambda} = -\frac{1}{\lambda} \cdot \Delta\textsc{Alg}.$$

**(b)** Suppose the request is between $s_1$ and $s_2$, and suppose that $s_1$ is predicted. LambdaDC moves both servers and $\Delta\textsc{Alg} = 1 + \lambda$. This means that $x_1'$ already covers the request. Thus, moving $s_1$ towards the request decreases $\Psi$ by $(1+\lambda)/\lambda$, while $s_2$ increases $\Psi$ by at most $(1+\lambda)/\lambda \cdot \lambda$. Also, $\Theta$ decreases by $1+\lambda$. We can conclude that

$$\Delta\Phi \leq \frac{1+\lambda}{\lambda}(-1+\lambda) - (1+\lambda) = -\frac{1}{\lambda}(1+\lambda) = -\frac{1}{\lambda} \cdot \Delta\textsc{Alg}.$$

Summing over all rounds, we obtain $\textsc{Alg}(I) \leq (1+\lambda)\textsc{FtP}(I) + \lambda|s_1^0 - s_2^0|$. ◀

Finally, we give a robustness guarantee for LambdaDC's performance independently of the prediction quality.

▶ **Lemma 6.** *For any instance $I$ and $\lambda \in (0,1]$, there is some $c \geq 0$ that only depends on the initial configuration such that* $\textsc{Alg}(I) \leq (1 + 1/\lambda) \cdot \textsc{Opt}(I) + c$.

The proof of this claim is similar to the proof of Lemma 5 with the crucial difference that the reference algorithm is unknown. Hence, the multiplicative factor is larger but relative to the optimal solution and, thus, independent of the prediction error.

**Proof.** Let $I$ be an arbitrary instance and let $\lambda \in (0,1]$. Let $s_1, s_2$ be LambdaDC's servers and $x_1, x_2$ the servers of an optimal algorithm. We define

$$\Phi = \underbrace{(1+\lambda)\left(|s_1 - x_1| + |s_2 - x_2|\right)}_{\Psi} + \underbrace{|s_1 - s_2|}_{\Theta}.$$

Upon arrival of a request, first the optimal algorithm moves and $\Phi$ increases by at most $(1+\lambda) \cdot \Delta\textsc{Opt}$. Second LambdaDC moves and, by scaling the instance, we assume that the request is served after exactly one time unit. We distinguish whether the request is between the algorithm's servers or not, and show that in each case $\Phi$ decreases by at least $\lambda \cdot \Delta\textsc{Alg}$.

**(a)** Let the request be not between the servers, say on the left of $s_1$. Either $x_1$ or $x_2$ covers the request, hence moving $s_1$ decreases $\Psi$ by $1+\lambda$ while it increases $\Theta$ by 1. Thus,

$$\Delta\Phi \le -(1+\lambda)+1 = -\lambda = -\lambda \cdot \Delta\textsc{Alg}.$$

**(b)** Let the request be between $s_1$ and $s_2$, and suppose that $s_1$ is predicted. The request is covered by $x_1$ or $x_2$. In the worst case ($x_2$ covers the request), moving $s_1$ towards the request increases $\Psi$ by at most $1+\lambda$, while $s_2$ decreases $\Psi$ only by $(1+\lambda)\lambda$. Also, $\Theta$ decreases by $1+\lambda$. Put together,

$$\Delta\Phi \le (1+\lambda)(1-\lambda) - (1+\lambda) = -\lambda(1+\lambda) = -\lambda \cdot \Delta\textsc{Alg}.$$ ◀

## 3.2 Optimality of LambdaDC: the Consistency-Robustness Tradeoff

We now show that LambdaDC is optimal for two servers, in the sense that no memory-constrained algorithm can achieve a better robustness-consistency tradeoff. As we target memory-constrained algorithms, at any time, we can use *force* requests, cf., Section 1, to enforce the algorithm to place its servers at prescribed locations.

▶ **Theorem 7.** *Let $\mathcal{A}$ be a learning-augmented memory-constrained algorithm for the 2-server problem on the line and let $\lambda \in (0,1]$. If $\mathcal{A}$ is $(1+\lambda)$-consistent, it is at least $(1+1/\lambda)$-robust.*

**Proof.** Let $\lambda \in (0,1]$ and $\mathcal{A}$ be a $(1+\lambda)$-consistent, memory-constrained algorithm for the 2-server problem on the line. This means for every instance $I$, $\mathcal{A}(I) \le (1+\lambda)\cdot\textsc{Opt}(I)+\nu$ if $\eta=0$, where $\nu$ depends on the initial configuration. Let $a, b$ and $c$ be consecutive points on the line at position $-1$, 0 and $L \ge 1+1/\lambda$, and $(a,b)$ the algorithm's initial configuration.

Consider the instance $I^\infty$ which is composed of a force to $(a,c)$, followed by arbitrarily many alternating requests at $b$ and $a$. Clearly, an optimal solution for instance $I^\infty$ is to move the right server to $c$ and then immediately back to $b$ with a total cost of $2L$.

Assume that $\mathcal{A}$ gets this optimal solution as prediction. $\mathcal{A}$ moves one server to $c$ for the first request. Since the consistency implies that $\mathcal{A}(I^\infty) \le (1+\lambda)\textsc{Opt}$, at some point in time $\mathcal{A}$ has to move the right server to $b$. Denote the instance which ends at this point in time by $I$. Note that $\mathcal{A}(I^\infty) \ge \mathcal{A}(I)$. Let $n_L$ denote the number of times in instance $I$ where the left server moves from $a$ to $b$ and back to $a$ (cost of 2). Since the right server pays at least $L$ for moving from $c$ to $b$, we conclude $\mathcal{A}(I) \ge 2n_L + 2L$. The consistency of $\mathcal{A}$ leads to $2n_L + 2L \le (1+\lambda)2L + \nu$, which means $n_L \le \lambda L + \nu/2$.

We now construct another instance $I^\omega$ by concatenating $\omega$ copies of instance $I$, each starting by the force to $(a,c)$. We call such a copy an *iteration*, and in each iteration we use the same predictions as in instance $I$. $\mathcal{A}$ has to pay at least $L$ for the force, as the right server was previously on $b$, and then $\mathcal{A}$ follows the same behavior as in $I$ in each iteration. So $\mathcal{A}(I^\omega) \ge \omega\cdot(2n_L+2L)$. Another solution for instance $I^\omega$ is to move the right server to $c$ in the beginning with cost $L$ and leave it there, while the left server alternates between $a$ and $b$. Hence, $\textsc{Opt}(I^\omega) \le L + \omega\cdot 2(n_L+1)$. Indeed, $b$ is requested $n_L+1$ times per iteration: $n_L$ where $\mathcal{A}$ uses the left server and one where it uses the right server. The ratio is then

$$\frac{\mathcal{A}(I^\omega)}{\textsc{Opt}(I^\omega)} \ge \frac{\omega\cdot(2n_L+2L)}{L+\omega\cdot 2(n_L+1)} \xrightarrow{\omega\to\infty} \frac{2n_L+2L}{2(n_L+1)} = 1 + \frac{L-1}{n_L+1} \ge 1 + \frac{L-1}{\lambda L + \frac{\nu}{2} + 1} \xrightarrow{L\to\infty} 1 + \frac{1}{\lambda},$$

which implies that $\mathcal{A}$ is at least $(1+1/\lambda)$-robust. ◀

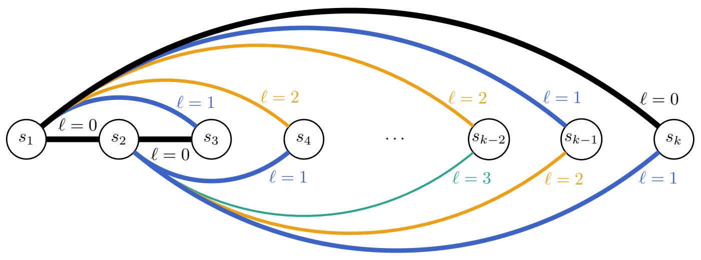


**Figure 1** Visualization of all incident $\delta_{ij}$-weights of the servers $s_1$ and $s_2$. The thickness (resp. color) of an arc indicates the influence of the corresponding distance in $\Phi$.

## 4 The General Case with $k$ Servers: Upper Bound

We present two lemmas which imply Theorem 1. The novelty lies in designing appropriate potential functions that capture the server movements at different speeds. This takes substantially more technical care than in the 2-server case but builds on the same ideas.

In the first step of the analysis, we compare the performance of LambdaDC and FtP.

▶ **Lemma 8.** *For every instance $I$ and $\lambda \in [0, 1]$, there is some $c > 0$ that only depends on the initial configuration such that* $\text{Alg}(I) \leq \alpha(k) \cdot \text{FtP}(I) + c$.

Let $I$ be an arbitrary instance. Note that $\lambda = 0$ implies $\text{Alg}(I) \leq \text{FtP}(I)$ as LambdaDC can only shortcut FtP's moves. So, we now assume that $\lambda \in (0, 1]$. We define a new potential function $\Phi$ as follows. Let $s_1, \dots, s_k$ be the servers of LambdaDC and let $x'_1, \dots, x'_k$ be the servers of FtP. For $1 \leq i < j \leq k$ and $\ell = \min\{j - i, k - (j - i)\} - 1$ we define $\delta_{ij} = \lambda^\ell$, see Figure 1. Then,

$$\Phi = \underbrace{\frac{\alpha(k)}{\lambda} \cdot \sum_{i=1}^{k} |s_i - x'_i|}_{\Psi} + \underbrace{\sum_{i<j} \delta_{ij} |s_i - s_j|}_{\Theta}.$$

Intuitively, the leading coefficient of $\Psi$ comes from the targeted competitive ratio. Then, in $\Theta$, the coefficient in front of each term depends on the number of interleaving servers. Following the idea of Lemma 3, when LambdaDC moves a server by a distance of 1 as in Opt, its neighbor moves by a distance of $\lambda$. Hence, correcting the position of this neighbor means that the next server moves by a distance $\lambda^2$. Therefore, this geometric decrease in the consequences of a movement also appears in the expression of $\Theta$. The symmetric increase when $j - i$ grows is more difficult to explain intuitively, but is required to compensate the modifications of $\Psi$. The coefficients of $\Theta$ are illustrated in Figure 1.

We carefully analyze in the full version of this paper how the potential changes when FtP and LambdaDC move servers. Further, we give a robustness guarantee for LambdaDC for any error.

▶ **Lemma 9.** *For any instance $I$ and $\lambda \in (0, 1]$, there is some $c \geq 0$ that only depends on the initial configuration such that* $\text{Alg}(I) \leq \beta(k) \cdot \text{Opt}(I) + c$.

Proving the general upper bound on the competitive ratio, independent of the prediction error, is much more intricate than in the two-server case and than the consistency proof. Again, our key ingredient is a carefully chosen potential function $\Phi$. We generalize the function used for the consistency bound even further by refining the weights, in particular, adding server-dependent weights to the term $\Psi$ measuring the distance between the positions of the algorithm's servers and the optimal servers.

Let $\lambda \in (0,1]$. Fix $k$, let $\beta = \beta(k) = \sum_{i=0}^{k-1} \lambda^{-i}$, and let $s_1, \dots, s_k$ be the servers of LambdaDC and let $x_1, \dots, x_k$ be the servers of an optimal solution. The potential function is

$$\Phi = \underbrace{\beta\gamma \left( \sum_{i=1}^{k} \omega_i |s_i - x_i| \right)}_{\Psi} + \underbrace{\sum_{i<j} \delta_{ij} |s_i - s_j|}_{\Theta} .$$

We specify the weights in this function as follows. For a pair of servers $s_i, s_j$ with $1 \le i < j \le k$, let $\ell = \min\{j-i, k-(j-i)\} - 1$ and $\delta_{ij} = (\lambda^{\ell} + \lambda^{k-2-\ell})/(1+\lambda^{k-2})$.

The intuition of the weights in the spreadness part $\Theta$ is the same as in the consistency potential function above. However, the new weights $\omega_i$ in the matching part $\Psi$ (defined below) require the more complex weights $\delta_{ij}$ compared to the simpler $\lambda^{\ell}$ weights.

Further, we define $d_{\lceil k/2 \rceil} = 0$ if $k$ is odd and for all $1 \le i \le \lfloor k/2 \rfloor$ let

$$d_i = d_{k+1-i} = \frac{2}{1+\lambda^{k-2}} \sum_{\ell=i-1}^{k-1-i} \lambda^{\ell}.$$

We demonstrate in the full version of this paper that these values correspond to the change of $\Theta$ when a server of LambdaDC moves. Let $\gamma = d_1/(\beta - 1)$, $\omega_1 = \omega_k = 1$ and for $2 \le i \le \lceil k/2 \rceil$ we define the server-individual weights

$$\omega_i = \omega_{k+1-i} = \begin{cases} \dfrac{2\lambda \sum_{j=1}^{i/2-1} d_{2j} - 2\sum_{j=1}^{i/2-1} d_{2j+1} + \lambda d_i + (2+\lambda)\gamma}{\beta\gamma\lambda} & \text{if } i \text{ is even, and} \\ \dfrac{2\lambda \sum_{j=1}^{(i-1)/2} d_{2j} - 2\sum_{j=1}^{(i-3)/2} d_{2j+1} - d_i + \gamma}{\beta\gamma} & \text{if } i \text{ is odd.} \end{cases}$$

In the full version of this paper we prove Lemma 9 by exhaustively reviewing all possible moves and bounding the corresponding change of $\Phi$. Establishing a constant upper bound of the $\omega$-weights yields a general upper bound on the increase of $\Phi$ independently of the choice of the optimal solution's server. We further choose the scaling parameter $\gamma$ such that the decrease of $\Phi$ exactly matches the required lower bound for the case where the request is outside of the convex hull of LambdaDC's servers. The remaining cases are split among the possible locations where a request can appear between two servers of LambdaDC, and we show in each case that $\Phi$ decreases enough. Intuitively, the $\omega$ values are defined such that a wrong prediction gives a tight bound on the decrease of $\Phi$ for LambdaDC's move, while a correct prediction still guarantees a loose bound.

## 5 The Consistency-Robustness Tradeoff

In this section we give a bound on the consistency-robustness tradeoff, as stated in Theorem 2. Our bound holds for memory-constrained algorithms that satisfy a certain locality property, which includes LambdaDC. Informally, we require that a $k$-server algorithm with a certain consistency $\mu(k)$ shall have a consistency $\mu(k')$ on a sub-instance that it serves with $k' < k$ servers. The rationale is to prevent the mere presence of additional unused workers to allow the algorithm to perform poorly on a subinstance served by few servers, as $\mu(k') < \mu(k)$. Hence, such algorithms are expected to present a better performance on a modified instance where some extreme servers are removed and side-effects due to their presence are simulated. In the following, we make this intuition precise and sketch our worst-case construction.

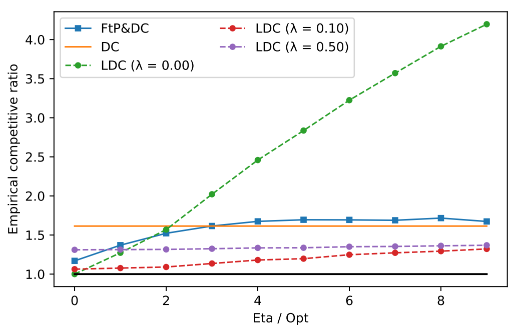
Given an algorithm $\mathcal{A}$ which is $\mu(k)$-consistent for the $k$-server problem, we define the notion of **locally-consistent**. Given an instance of the $k$-server problem served by algorithm $\mathcal{A}$, consider any subset $S'$ of $k'$ consecutive servers. We construct an instance $I'$ of the $k'$-server problem based on $I$ and $S'$: If a request of $I$ is predicted to be served by a server in $S'$ then this request is replicated in $I'$. Otherwise, $I'$ requests the position of the closest server among $S'$ after $\mathcal{A}$ served this request in $I$ (in order to take into account side-effects due to additional servers in the original instance). Let $\text{FtP}(I')$ be the cost of solving $I'$ following the original predictions of $I'$ (using the closest server among $S'$ if a server outside of $S'$ was initially predicted). An algorithm is *locally-consistent* if its total cost on $I$ restricted to the servers in $S'$ is at most $\mu(k') \cdot \text{FtP}(I') + c$, where $c$ can be upper bounded based only on the initial configuration. We further require that if the initial and final configurations differ by a total distance of $\varepsilon$, then $c = O(k'\varepsilon)$. Note that LambdaDC is locally-consistent as its behavior in $I$ restricted to the servers in $S'$ is equal to its behavior in $I'$ with $k'$ servers.

The proof of Theorem 2 generalizes ideas from the 2-server case (Section 3.2) in a highly non-trivial way. We only sketch the main idea and refer to the full version of this paper for details. Let $\mathcal{A}$ be a memory-constrained and locally-consistent deterministic algorithm. We construct an instance that starts with $k$ equidistant servers. First, a point far on the right is requested. Then the initial server locations are requested following specific rules until the rightmost server comes back. Predictions correspond to the server initially at the point requested. The consistency of $\mathcal{A}$ limits the possible cost paid before the rightmost server comes back. The locally-consistent definition allows, with technical care, to link the distance traveled by two neighboring servers: the left one travels a total distance at most $\lambda$ times the right one (plus negligible terms). An offline solution can afford to initially shift all servers to the right, and then move only the leftmost server, which $\mathcal{A}$ could not move much. We then repeat this instance, and use the memory-constrained and deterministic characteristics of $\mathcal{A}$ to eliminate constant costs and show the desired robustness lower bound, again with technical care.

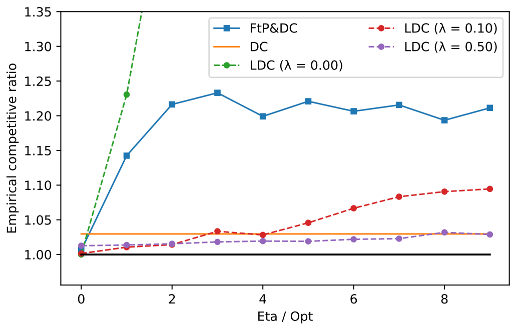
## 6 Experiments

We supplement our theoretical results by empirically comparing our learning-augmented algorithm LambdaDC with the classical online algorithm ignoring predictions DoubleCoverage [16] and the previously proposed prediction-based algorithm FtP&DC [2] on real world data. We implemented FtP&DC with the hyperparameter $\gamma$ equal to 1. The instances are based on the BrightKite-Dataset [15], which is composed of sequences of coordinates of app check-ins. This dataset was used previously to evaluate and compare learning-augmented algorithms for caching problems [2, 33]. We extract sequences of 1000 checkins, normalize the latitudes to the interval $[0, 4000]$, and use these values as the positions of the requests on the line. All servers start at the same initial random position.

We generate synthetic predictions in a semi-random fashion. Fix two parameters $p$, the number of *bins*, and $b$, the *bin size*, and an instance. Our goal is to generate evenly distributed predictions, i.e., in each bin $i \in \{1, \dots, p\}$ there are at least five predictions with relative error between $(i-1)b$ and $ib$. For that we iteratively sample many predictions with an increasing number of wrong choices with respect to the optimal solution. While this procedure does not find all predictions, especially these with the largest relative error, it gives a good tradeoff between running time and range of prediction error. Additionally, we use an optimal solution of the instance as the perfect prediction. We set $p = 10$ and $b$ as high as we find for at least 40 instances of such predictions. Other instances are discarded.

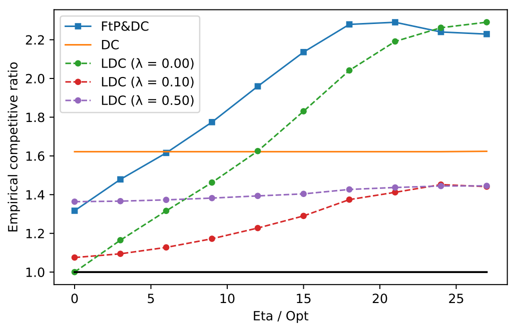


**(a)** non-lazy.

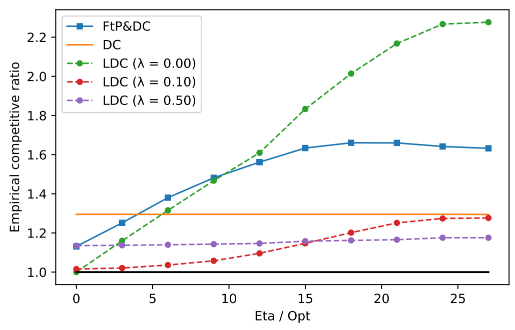


**(b)** lazy.

**Figure 2** Results for $k = 2$ and $b = 1$.



**(a)** non-lazy.



**(b)** lazy.

**Figure 3** Results for $k = 50$ and $b = 3$.

We implement all algorithms in lazy and non-lazy variants. The mean empirical competitive ratios for $k = 2$ and $k = 50$ are displayed in Figures 2 and 3. They show well that, for a reasonable choice of $\lambda$ ($0.1 \leq \lambda \leq 0.5$), LambdaDC outperforms both DoubleCoverage and FtP&DC for almost all generated prediction errors. This is true even if laziness is allowed. We suspect that FtP&DC only makes few expensive resets in our instances, while LambdaDC benefits from many cheap improvements of a lazy execution.

## 7 PAC Learnability of Predictions

While our results show the applicability of untrusted predictions, it is a natural question whether such predictions are actually learnable.

In Appendix A, we show that for our model a *static* prediction sequence is PAC learnable in an agnostic sense using empirical risk minimization. That is, given an unknown distribution over request sequences which we can sample, we can find a prediction that is close to the best possible prediction for this distribution in terms of prediction error using a bounded number of samples.

▶ **Theorem 10.** *For any $\epsilon, \delta \in (0, 1)$, a known initial configuration $C_0$ and any distribution $\mathcal{D}$ over the sequences of $n$ requests of known extent, there exists an algorithm which, given an i.i.d. sample of $\mathcal{D}$ of size $m \in \mathcal{O}\left(\frac{1}{\epsilon^2} \cdot (n \log k - \log \delta)\eta_{\max}^2\right)$, returns a prediction $\tau_p \in \mathcal{H}$ in polynomial time depending on $k$, $n$ and $m$, such that with probability of at least $(1 - \delta)$ it holds $\mathbb{E}_{\sigma\sim\mathcal{D}}[\eta_\sigma(\tau_p)] \leq \mathbb{E}_{\sigma\sim\mathcal{D}}[\eta_\sigma(\tau^*)] + \epsilon$, where $\tau^* = \arg\min_{\tau\in\mathcal{H}} \mathbb{E}_{\sigma\sim\mathcal{D}}[\eta_\sigma(\tau)]$.*

We remark that a pre-computed static prediction does not include information about the partially revealed input. Thus, this is a rather weak prediction and may not help LambdaDC much. The existence of an *adaptive* prediction policy which can be efficiently learned remains an open question. Such a policy would provide much more valuable information to our learning-augmented online algorithm.

## 8 Conclusion

We show the power of (untrusted) predictions in designing online algorithms for the $k$-server problem on the line. Our algorithm generalizes the classical DoubleCoverage algorithm [16] in an intuitive way and admits a (nearly) tight error-dependent competitive analysis, based on new potential functions, and outperforms other methods from the literature. While we can show PAC learnability for static predictions, we leave open whether possibly more powerful adaptive prediction models are learnable.

Clearly, it would be interesting to see whether our results generalize to more general metric spaces than the line. In fact, in a full version we show that our upper bounds for the 2-server problem can be extended to tree metrics and we expect that an extension to $k$ servers is possible. However, for more general metrics our current approach seems not to generalize well. Further, we focused on memory-constrained algorithms, leaving open a more precise quantification of the power of memory. Finally, the recent success on randomized k-server algorithms [12] raises the question whether and how randomized algorithms can benefit from (ML) predictions.

## References

1 Spyros Angelopoulos, Christoph Dürr, Shendan Jin, Shahin Kamali, and Marc P. Renault. Online computation with untrusted advice. In *ITCS*, volume 151 of *LIPIcs*, pages 52:1–52:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

2 Antonios Antoniadis, Christian Coester, Marek Eliáš, Adam Polak, and Bertrand Simon. Online metric algorithms with untrusted predictions. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pages 345–355. PMLR, 2020.

3 Antonios Antoniadis, Carsten Fischer, and Andreas Tönnis. A collection of lower bounds for online matching on the line. In *LATIN*, volume 10807 of *Lecture Notes in Computer Science*, pages 52–65. Springer, 2018.

4 Antonios Antoniadis, Themis Gouleakis, Pieter Kleer, and Pavel Kolev. Secretary and online matching problems with machine learned advice. In *NeurIPS*, 2020.

5 Yossi Azar, Stefano Leonardi, and Noam Touitou. Flow time scheduling with uncertain processing time. In *STOC*, pages 1070–1080. ACM, 2021.

6 Étienne Bamas, Andreas Maggiori, Lars Rohwedder, and Ola Svensson. Learning augmented energy minimization via speed scaling. In *NeurIPS*, 2020.

7 Étienne Bamas, Andreas Maggiori, and Ola Svensson. The primal-dual method for learning augmented algorithms. In *NeurIPS*, 2020.

8 Soumya Banerjee. Improving online rent-or-buy algorithms with sequential decision making and ML predictions. In *NeurIPS*, 2020.

9 Yair Bartal, Béla Bollobás, and Manor Mendel. Ramsey-type theorems for metric spaces with applications to online problems. *J. Comput. Syst. Sci.*, 72(5):890–921, 2006.

10 Yair Bartal and Elias Koutsoupias. On the competitive ratio of the work function algorithm for the k-server problem. *Theor. Comput. Sci.*, 324(2-3):337–345, 2004.

11 Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis.* Cambridge University Press, 1998.

12 Sébastien Bubeck, Michael B. Cohen, Yin Tat Lee, James R. Lee, and Aleksander Madry. k-server via multiscale entropic regularization. In *STOC*, pages 3–16. ACM, 2018.

13 Niv Buchbinder, Christian Coester, and Joseph (Seffi) Naor. Online k-taxi via double coverage and time-reverse primal-dual. In *IPCO*, volume 12707 of *Lecture Notes in Computer Science*, pages 15–29. Springer, 2021.

14 Ashish Chiplunkar and Sundar Vishwanathan. Randomized memoryless algorithms for the weighted and the generalized $k$-server problems. *ACM Trans. Algorithms*, 16(1):14:1–14:28, 2020.

15 Eunjoon Cho, Seth A. Myers, and Jure Leskovec. Friendship and mobility: user movement in location-based social networks. In *KDD*, pages 1082–1090. ACM, 2011.

16 Marek Chrobak, Howard J. Karloff, T. H. Payne, and Sundar Vishwanathan. New results on server problems. *SIAM J. Discret. Math.*, 4(2):172–181, 1991.

17 Marek Chrobak and Lawrence L. Larmore. An optimal on-line algorithm for k-servers on trees. *SIAM J. Comput.*, 20(1):144–148, 1991.

18 Paul Dütting, Silvio Lattanzi, Renato Paes Leme, and Sergei Vassilvitskii. Secretaries with advice. In *EC*, pages 409–429. ACM, 2021.

19 Sreenivas Gollapudi and Debmalya Panigrahi. Online algorithms for rent-or-buy with expert advice. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 2319–2327. PMLR, 2019.

20 Sungjin Im, Ravi Kumar, Mahshid Montazer Qaem, and Manish Purohit. Non-clairvoyant scheduling with predictions. In *SPAA*, pages 285–294. ACM, 2021.

21 Manoel Leandro L Junior, AD Doria Neto, and Jorge D Melo. The k-server problem: a reinforcement learning approach. In *IJCNN*, 2005.

22 Elias Koutsoupias. The k-server problem. *Comput. Sci. Rev.*, 3(2):105–118, 2009.

23 Elias Koutsoupias and Christos H. Papadimitriou. On the k-server conjecture. *J. ACM*, 42(5):971–983, 1995.

24 Elias Koutsoupias and David Scot Taylor. The CNN problem and other k-server variants. *Theor. Comput. Sci.*, 324(2-3):347–359, 2004.

25 Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *SIGMOD Conference*, pages 489–504. ACM, 2018.

26 Ravi Kumar, Manish Purohit, Aaron Schild, Zoya Svitkina, and Erik Vee. Semi-online bipartite matching. In *ITCS*, volume 124 of *LIPIcs*, pages 50:1–50:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

27 Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In *SODA*, pages 1859–1877. SIAM, 2020.

28 Thomas Lavastida, Benjamin Moseley, R. Ravi, and Chenyang Xu. Learnable and instance-robust predictions for online matching, flows and load balancing. In *ESA*, volume 204 of *LIPIcs*, pages 59:1–59:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

29 M. Leandro Costa, C. A. Araujo Padilha, J. Dantas Melo, and A. Duarte Doria Neto. Hierarchical reinforcement learning and parallel computing applied to the k-server problem. *IEEE Latin America Transactions*, 14(10):4351–4357, 2016.

30 Alexander Lindermayr. Learning-augmented online algorithms for the 2-server problem on the line and generalizations. Master's thesis, University of Bremen, Germany, 2020.

31 Ramon Augusto Sousa Lins, Adrião Duarte Dória Neto, and Jorge Dantas de Melo. Deep reinforcement learning applied to the $k$-server problem. *Expert Syst. Appl.*, 135:212–218, 2019.

32 Pinyan Lu, Xuandi Ren, Enze Sun, and Yubo Zhang. Generalized sorting with predictions. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 111–117. SIAM, 2021.

33 Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 3302–3311. PMLR, 2018.

34 Mohammad Mahdian, Hamid Nazerzadeh, and Amin Saberi. Allocating online advertisement space with unreliable estimates. In *EC*, pages 288–294. ACM, 2007.

35 Mark S. Manasse, Lyle A. McGeoch, and Daniel Dominic Sleator. Competitive algorithms for on-line problems. In *STOC*, pages 322–333. ACM, 1988.

36 Mark S. Manasse, Lyle A. McGeoch, and Daniel Dominic Sleator. Competitive algorithms for server problems. *J. Algorithms*, 11(2):208–230, 1990.

37 Andres Muñoz Medina and Sergei Vassilvitskii. Revenue optimization with approximate bid predictions. In *NIPS*, pages 1858–1866, 2017.

38 Michael Mitzenmacher. A model for learned bloom filters and optimizing by sandwiching. In *NeurIPS*, pages 462–471, 2018.

39 Michael Mitzenmacher. Scheduling with predictions and the price of misprediction. In *ITCS*, volume 151 of *LIPIcs*, pages 14:1–14:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

40 Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ML predictions. In *NeurIPS*, pages 9684–9693, 2018.

41 Dhruv Rohatgi. Near-optimal bounds for online caching with machine learned advice. In *SODA*, pages 1834–1845. SIAM, 2020.

42 Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning - From Theory to Algorithms*. Cambridge University Press, 2014.

43 Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.

44 Shufan Wang, Jian Li, and Shiqiang Wang. Online algorithms for multi-shop ski rental with machine learned advice. In *NeurIPS*, 2020.

45 Alexander Wei. Better and simpler learning-augmented online caching. In *APPROX*, volume 176 of *LIPIcs*, pages 60:1–60:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

46 Alexander Wei and Fred Zhang. Optimal robustness-consistency trade-offs for learning-augmented online algorithms. In *NeurIPS*, 2020.

47 Wenming Zhang and Yongxi Cheng. A new upper bound on the work function algorithm for the k-server problem. *J. Comb. Optim.*, 39(2):509–518, 2020.

## A PAC Learnability of Predictions

We show that our predictions are PAC learnable in an agnostic sense with a sample complexity polynomial in the number of requests and we give an efficient learning algorithm. Let $\mathcal{D}$ be an unknown distribution of sequences of $n$ requests represented by points in the interval $[0, 1]$. Here we assume a bounded line as a metric (scaled to $[0, 1]$), which is a restriction but natural in most applications. Further, we assume that we can sample i.i.d. sequences from $\mathcal{D}$.

Let $\mathcal{H} = \{1, \dots, k\}^n$ denote a hypothesis class containing all possible static predictions, i.e., the set of all $k$-server solutions for request sequences of length $n$. Let $C_0$ be a known initial configuration. The prediction error for a prediction $\tau \in \mathcal{H}$ on a request sequence $\sigma$ is defined as $\eta_\sigma(\tau) = \textsc{Ftp}(\sigma, \tau) - \textsc{Opt}(\sigma)$, where $\textsc{Ftp}(\sigma, \tau)$ is the total cost of following the prediction $\tau$ on the sequence $\sigma$ starting in $C_0$, and $\textsc{Opt}(\sigma)$ is the cost of an optimal solution on $\sigma$ starting in $C_0$. Then, $\eta_\sigma(\tau) \le \eta_{\max} \le n$ for all possible sequences $\sigma$ and for all $\tau \in \mathcal{H}$.

We argue that we can use a classical *empirical risk minimization (ERM)* learning method, see, e.g., [42]. The ERM method uses a training set $S = \{\sigma_1, \dots, \sigma_m\}$ of i.i.d. samples from $\mathcal{D}$. Then, it determines a prediction $\tau_p \in \mathcal{H}$ that minimizes the *empirical error* $\eta_S(\tau) = \frac{1}{m}\sum_{j=1}^{m} \eta_{\sigma_j}(\tau)$. Since our hypothesis class is finite and the error function bounded, classical results imply that our predictions are PAC learnable in an agnostic sense with a polynomial sample complexity. Further, we show that the problem of finding the prediction minimizing the empirical error within the training set can be reduced to an offline $k$-server problem on a modified request sequence $\widetilde{\sigma}$ of length $n$, where the distance between the $\ell$th and $i$th request in $\widetilde{\sigma}$ is given by $\frac{1}{m}\sum_{j=1}^{m} d(\sigma_j(\ell), \sigma_j(i))$. This problem can be solved efficiently [16].

▶ **Theorem 10.** *For any $\epsilon, \delta \in (0,1)$, a known initial configuration $C_0$ and any distribution $\mathcal{D}$ over the sequences of $n$ requests of known extent, there exists an algorithm which, given an i.i.d. sample of $\mathcal{D}$ of size $m \in \mathcal{O}\left(\frac{1}{\epsilon^2} \cdot (n \log k - \log \delta)\eta_{\max}^2\right)$, returns a prediction $\tau_p \in \mathcal{H}$ in polynomial time depending on $k$, $n$ and $m$, such that with probability of at least $(1-\delta)$ it holds $\mathbb{E}_{\sigma\sim\mathcal{D}}[\eta_\sigma(\tau_p)] \leq \mathbb{E}_{\sigma\sim\mathcal{D}}[\eta_\sigma(\tau^*)] + \epsilon$, where $\tau^* = \arg\min_{\tau\in\mathcal{H}} \mathbb{E}_{\sigma\sim\mathcal{D}}[\eta_\sigma(\tau)]$.*

**Proof.** Since the hypothesis class $\mathcal{H}$ is finite with $|\mathcal{H}| = k^n$, and our non-negative error function is bounded by $\eta_{\max}$, classical results, see e.g. [42], imply that $\mathcal{H}$ is agnostically PAC-learnable using the ERM algorithm with a sample complexity of

$$m \leq \left\lceil \frac{2\log(2|\mathcal{H}|/\delta)\eta_{\max}^2}{\epsilon^2} \right\rceil \in \mathcal{O}\left(\frac{(n\log k - \log\delta)\eta_{\max}^2}{\epsilon^2}\right).$$

That is, given a sample of size at least $m$, the ERM algorithm outputs with a probability of at least $(1-\delta)$ a prediction $\tau_p$ such that $\mathbb{E}_{\sigma\sim\mathcal{D}}[\eta_\sigma(\tau_p)] \leq \mathbb{E}_{\sigma\sim\mathcal{D}}[\eta_\sigma(\tau^*)] + \epsilon$ holds, where $\tau^* = \arg\min_{\tau\in\mathcal{H}} \mathbb{E}_{\sigma\sim\mathcal{D}}[\eta_\sigma(\tau)]$.

It remains to describe an efficient implementation of the ERM algorithm for our setting. Let $S = \{\sigma_1, \ldots, \sigma_m\}$ be a sample drawn i.i.d. from $\mathcal{D}$. We assume that this can be done in polynomial time in $m$. For a sequence $\sigma_j \in S$ let $\sigma_j(i)$ be the position of the $i$th request in $\sigma_j$. We further define for $1 \leq \ell \leq i \leq n$ and $1 \leq k' \leq k$ the distance functions $\delta_j(\ell, i) = |\sigma_j(\ell) - \sigma_j(i)|$ and $\gamma_j(k', i) = |C_0(k') - \sigma_j(i)|$. The empirical error of a prediction $\tau$ is in our setting defined as

$$\eta_S(\tau) = \frac{1}{m}\sum_{j=1}^{m} \eta_{\sigma_j}(\tau) = \frac{1}{m}\sum_{j=1}^{m} \mathrm{FtP}(\sigma_j, \tau) - \mathrm{Opt}(\sigma_j).$$

The ERM algorithm outputs the prediction $\tau_p \in \mathcal{H}$ that minimizes $\eta_S(\tau)$ as a function over $\mathcal{H}$. Since iterating over all predictions in $\mathcal{H}$ takes exponential time, we compute $\tau_p$ differently. To do so, we first observe that $\frac{1}{m}\sum_{j=1}^{m} \mathrm{Opt}(\sigma_j)$ is independent of $\tau$, thus minimizing $\eta_S(\tau)$ can be reduced to minimizing

$$\begin{aligned}\frac{1}{m}\sum_{j=1}^{m} \mathrm{FtP}(\sigma_j, \tau) &= \frac{1}{m}\sum_{j=1}^{m}\sum_{k'=1}^{k}\sum_{i=1}^{n} \xi_{k',i}^{\tau} \cdot \gamma_j(k', i) + \sum_{\ell=1}^{i} \chi_{k',i,\ell}^{\tau} \cdot \delta_j(\ell, i) \\ &= \sum_{k'=1}^{k}\sum_{i=1}^{n} \xi_{k',i}^{\tau} \cdot \left(\frac{1}{m}\sum_{j=1}^{m} \gamma_j(k', i)\right) + \sum_{\ell=1}^{i} \chi_{k',i,\ell}^{\tau} \cdot \frac{1}{m}\sum_{j=1}^{m} \delta_j(\ell, i), \qquad (1)\end{aligned}$$

where $\chi_{k',i,\ell}^{\tau} \in \{0,1\}$ indicates (i.e. is equal to 1) that server $k'$ serves the $i$th request of $\sigma_j$ directly after the $\ell$th request of $\sigma_j$ in $\tau$ and $\xi_{k',i}^{\tau} \in \{0,1\}$ indicates that the $i$th request of $\sigma_j$ is the first one that server $k'$ serves in $\tau$.

We now demonstrate that we can efficiently compute a prediction $\tau \in \mathcal{H}$ that minimizes (1). Indeed, observe that (1) is equal to the total cost of the solution $\tau$ for the $k$-server instance that starts in $C_0$ and serves a sequence $\widetilde{\sigma}$ of length $n$, where the distance between the $\ell$th and $i$th request in $\widetilde{\sigma}$ is given by $\delta'(\ell, i) = \frac{1}{m}\sum_{j=1}^{m} \delta_j(\ell, i)$ and the distance between the $i$th request in $\widetilde{\sigma}$ and the initial position of server $k'$ is given by $\gamma'(k', i) = \frac{1}{m}\sum_{j=1}^{m} \gamma_j(k', i)$. But this means that any optimal solution $\widetilde{\tau}$ for this instance also minimizes (1). Clearly, $\widetilde{\tau} \in \mathcal{H}$, and an optimal solution for a k-server instance with known distance functions can be computed in $\mathcal{O}(kn^2)$ time using a min-cost flow algorithm [16]. ◀

## B Bound between Consistencies

▶ **Lemma 11.** *For every $\lambda \in [0,1]$, $\alpha(k) < 2\rho(k)$.*

**Proof.** First note that for $\lambda = 1$, $\alpha(k) = k = \rho(k)$. Now suppose that $\lambda < 1$. Applying the formula for the finite geometric series gives

$$\rho(k) = \frac{1-\lambda^k}{1-\lambda}.$$

We now prove the result based on the parity of $k$. Assume that $k$ is even. Recall that

$$\alpha(k) = 1 + 2\sum_{i=1}^{k/2-1} \lambda^i + \lambda^{k/2} = 1 + 2\frac{\lambda - \lambda^{k/2}}{1-\lambda} + \lambda^{k/2}$$

and, thus,

$$\frac{\alpha(k)}{\rho(k)} = \frac{(1+\lambda^{k/2})(1-\lambda) + 2(\lambda - \lambda^{k/2})}{1-\lambda^k} = \frac{1+\lambda-\lambda^{k/2}-\lambda^{k/2+1}}{1-\lambda^k} < 2.$$

Assume that $k$ is odd, then

$$\alpha(k) = 1 + 2\sum_{i=1}^{(k-1)/2} \lambda^i = 1 + 2\frac{\lambda - \lambda^{(k+1)/2}}{1-\lambda},$$

and we conclude that

$$\frac{\alpha(k)}{\rho(k)} = \frac{1-\lambda+2(\lambda-\lambda^{(k+1)/2})}{1-\lambda^k} = \frac{1+\lambda-2\lambda^{(k+1)/2}}{1-\lambda^k} < 2.$$ ◀