Double Coverage with Machine-Learned Advice

Alexander Lindermayr* Nicole Megow* Bertrand Simon†

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 trustiness 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 heavily 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.

*Department of Mathematics and Computer Science, University of Bremen, Germany. *{linderal,nmegow}@uni-bremen.de*
†IN2P3 Computing Center, CNRS, Villeurbanne, France. *bertrand.simon@cc.in2p3.fr*

1 Introduction

The k-server problem is one of the most fundamental online optimization problems. Manasse et al. [35, 36] 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 *μ-competitive* if for every instance I there is some constant c 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. Note that for the k-server problem, the constant c may depend only on the initial server positions and k but not on the instance I.

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 [15], considered in this paper, the uniform metric space (paging problem) [42] and tree metrics [16]. For the k-server problem on the line, Chrobak et al. [15] devised the DoubleCoverage algorithm and prove 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.

In the past decades, artificial intelligence and machine learning (ML) advanced rapidly and nowadays ML methods can be expected to predict often—but not always—uncertain data with good accuracy. This lack of provable guarantees for ML predictions and the application-depending urgent need for trustable performance guarantees for algorithms lead to the area of *learning-augmented online algorithms*. This recently emerging and very active field 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 which quality a prediction has, but we can use it in the analysis to measure an algorithm's performance depending on η. If a learning-augmented algorithm is $\mu(\eta)$-competitive for some function μ, we say that the algorithm is *α-consistent* if $\alpha = \mu(0)$ and *β-robust* if $\mu(\eta) \leq \beta$ for any prediction with prediction error η [40].

Very recently, Antoniadis et al. [3] 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 unconstrained *memory* which can be a serious drawback in practice when there are numerous servers and the length of the input is unknown.

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 [26]. A downside of this restriction is that deterministic memoryless algorithms cannot be competitive, and there is no difference between the type of information gathered by DoubleCoverage and unconstrained information encoding. This difference has been acknowledged by informally considering DoubleCoverage as memoryless [24], 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 [3,34,41,44], weighted caching [22], ski rental [8,19,40,43,45], TCP acknowledgement [7, 8], bin packing [2], scheduling [6, 28, 39, 40, 45], secretary problems [5], linear search [1], sorting [33] and online covering problems [7]. Learning-augmented algorithms have proven to be successful also in other areas, e.g., to speed up search queries [27], in revenue optimization [37], to compute low rank approximations [21], frequency estimation [20] and bloom filters [38].

Purohit et al. [40] introduce the use of a parameter $\lambda \in [0, 1]$ to express the consistency-robustness tradeoff of learning-augmented algorithms. It can be interpreted as an algorithm's indicator of trust in the given predictions: smaller λ 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 [2, 5, 7, 8, 40, 43, 45].

As mentioned, Antoniadis et al. [3] 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 (unlimited) memory and obtains a competitive ratio of $\min\{9 + 36\eta/\textsc{Opt}, 9k\}$ in the error model that we define below.

The k-server problem has been studied also in the context of reinforcement learning (RL), originating at [23] and including hierarchical RL learning [29] as well as deep RL learning [32].

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 [25] with a competitive ratio of $2k - 1$. For several special metric spaces there are even tighter bounds known for this algorithm [9, 18, 46]. Recently, Lee [30] proposed a $(\log k)^{O(1)}$-competitive algorithm for any metric space by allowing randomization. For memoryless randomized algorithms, a lower bound of k on the competitive ratio is known [24] and some recent efforts focus on a more general variant [13].

Our contribution We design learning-augmented memory-constrained online algorithms for the k-server problem on the line. To describe our results in more detail, 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 $t = 1, \dots, n$, 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 may assume w.l.o.g. $s_1 \leq \ldots \leq s_k$, since 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 $r_1, \dots, r_n$, we define a *prediction* for I as a sequence of indices $p_1, \dots, p_n$ from the set $\{1, \dots, k\}$. If $s_1, \dots, s_k$ 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. We denote its cost by FtP(I). Now let be $C'_1, \dots, C'_n$ the schedule of FtP and $C_1, \dots, C_n$ the schedule of a fixed optimal solution for some instance I. We define the *error* η_t *of* p_t as the distance between the configurations C'_t and C_t. The *total prediction error* is equal to $\eta = \sum_{t=1}^{n} \eta_t = \sum_{t=1}^{n} d(C_t, C'_t)$. We call a prediction *perfect* if $\eta = 0$.

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*

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

Let η 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*

$$\min\left\{\alpha(k)\left(1 + \frac{2\eta}{\textsc{Opt}}\right), \beta(k)\right\}.$$

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

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 [15], which is k-robust. An additional challenge is to preserve the memory-constrained property. We achieve this, by generalizing the classical DoubleCoverage [15] 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 [15]. 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.

While our result is tailored to the k-server problem, the learning-augmented framework by Antoniadis et al. [3] 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 cost of the simulated individual algorithms, LambdaDC is a simple algorithm that does not require any memory. Further, LambdaDC has a better performance for $k < 20$ and an appropriate parameter λ (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 which 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 [4].

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 Lemma 22) 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).

Finally, we demonstrate the power of our approach in empirical experiments on real-world data. We show that for a reasonable choice of λ our method outperforms the classical online algorithm DoubleCoverage as well as the algorithm in [3] for nearly all prediction errors.

2 Algorithm and Roadmap for the Analysis

The Algorithm LambdaDC We generalize the classical DoubleCoverage [15] in an intuitive way. Our algorithm LambdaDC includes the information about predicted servers as well as our trust into this advice using the following idea. 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 λ 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.

Potential Function Analysis The analysis of our algorithm builds on the powerful *potential function method*, as does the analysis of the classical DoubleCoverage [15].

Our potential analysis follows the well-known *interleaving moves* technique [10]. 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 Φ 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 Φ by at most $\mu \cdot \Delta\mathcal{B}_t(I)$, whereas

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

we can use a simple 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, μ is equal to the competitive ratio of $\mathcal{A}$ since Φ_0 only depends on the initial configuration.

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 bound the cost of FtP by the cost of the fixed optimal solution Opt (fixed with respect to the definition of η) and the prediction error η. 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 [15]. It is composed of a *matching part* Ψ, summing the distances between the server positions of an algorithm and the reference algorithm (Opt, FtP) and a *spreadness part* Θ, 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 proven bounds on consistency and robustness.

Lemma 3. *The consistency of LambdaDC is at least $\alpha(k)$.*

Proof. Consider k servers initially at positions $0, 1, -1, 2, -2, \ldots$ and the request sequence of length $k+1$ at positions $0.5, 0, 1, -1, 2, -2, \ldots$. 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. □

Lemma 4. *The robustness of LambdaDC is at least $\beta(k)$.*

Proof. Consider k servers initially at positions $\beta(i) = \sum_{i=0}^{k-1} \lambda^{-i}$, for $i \in \{1, \ldots, k\}$, and the request sequence of length $k+1$ at positions $0, \beta(1), \beta(2), \ldots, \beta(k)$. 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 and give proofs for Theorems 1 and 2 while maintaining the same structure as in the 2-server case. Finally, we illustrate and discuss the results of computational experiments in Section 6.

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 $\text{Alg}(I)$, and the cost for serving a request r_t by $\Delta\text{Alg}_t(I)$. If t is clear from the context then we omit the index.

Theorem 5. *For any parameter $\lambda \in [0,1]$, LAMBDADC has a competitive ratio of at most*

$$\min\left\{(1+\lambda)\left(1+\frac{2\eta}{\textsc{Opt}}\right), 1+\frac{1}{\lambda}\right\}.$$

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

We follow the three-step approach outlined in the previous section. Lemmata 6 to 8 below imply Theorem 5. We firstly compare the algorithm's performance to FTP.

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* $\textsc{Alg}(I) \leq (1+\lambda) \cdot \textit{FtP}(I) + c$.

Proof. Let I be an arbitrary instance and let servers start at positions s_1^0 and s_2^0. If $\lambda = 0$, LAMBDADC equals FTP, hence $\textsc{Alg}(I) = \textsc{FtP}(I)$. 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

$$\Phi \;=\; \underbrace{\frac{1+\lambda}{\lambda}\left(|s_1 - x'_1| + |s_2 - x'_2|\right)}_{\Psi \text{ (matching part)}} \;+\; \underbrace{|s_1 - s_2|}_{\Theta \text{ (spreadness part)}}.$$

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 Φ 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 λ. We distinguish whether the request is between the algorithm's servers or not, and prove in each case that Φ 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 Ψ by $(1+\lambda)/\lambda$ while it increases Θ 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 Ψ by $(1+\lambda)/\lambda$, while s_2 increases Ψ by at most $(1+\lambda)/\lambda \cdot \lambda$. Also, Θ 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 $\textsc{Alg}(I) \leq (1+\lambda)\textsc{FtP}(I) + \lambda|s_1^0 - s_2^0|$. □

As a second step, we give a bound on the cost of FTP w.r.t. an optimal solution.

Lemma 7. *For any instance I, $\textit{FtP}(I) \leq \textsc{Opt}(I) + 2\eta$.*

Proof. For any request r_t, let $C_{t-1} = (x'_1, x'_2)$ and $C_t = (x_1, x_2)$ be the configurations of the fixed optimal solution used in the definition of η before and after serving r_t. We define similarly $C^P_{t-1} = (s'_1, s'_2)$

and $C_t^P = (s_1, s_2)$ for the configurations of FtP. Let ΔOpt_t and ΔFtP_t be the corresponding cost for serving r_t. By the triangle inequality, we have that

$$\begin{aligned}
\Delta\text{FtP}_t &= |s_1' - s_1| + |s_2' - s_2| \\
&\leq |x_1' - x_1| + |s_1' - x_1'| + |s_1 - x_1| + |x_2' - x_2| + |s_2' - x_2'| + |s_2 - x_2| \\
&= d(C_{t-1}, C_t) + \eta_{t-1} + \eta_t \\
&= \Delta\text{Opt}_t + \eta_{t-1} + \eta_t,
\end{aligned}$$

and we can finish the proof by summing over all requests. □

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

Lemma 8. *For any instance I and $\lambda \in (0, 1]$, there is some $c \geq 0$ that only depends on the initial configuration such that* $\text{Alg}(I) \leq (1 + 1/\lambda) \cdot \text{Opt}(I) + c$.

The proof of this claim is similar to the proof of Lemma 6 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 Φ increases by at most $(1 + \lambda) \cdot \Delta\text{Opt}$. 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 Φ decreases by at least $\lambda \cdot \Delta\text{Alg}$.

(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 Ψ by $1 + \lambda$ while it increases Θ by 1. Thus,

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

(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 Ψ by at most $1 + \lambda$, while s_2 decreases Ψ only by $(1 + \lambda)\lambda$. Also, Θ decreases by $1 + \lambda$. Put together,

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

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 9. *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, $\mathcal{A}(I) \leq (1+\lambda)\cdot \textsc{Opt}(I) + \nu$ if $\eta = 0$, where ν depends on the initial configuration. Let a, b and c be consecutive points on the line at position -1, 0 and $L \geq 1 + 1/\lambda$, and (a, b) the algorithm's initial configuration.

Consider the instance I^∞ 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^∞ 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 $\mathcal{A}(I^\infty) \leq (1+\lambda)\textsc{Opt}$, 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 $\mathcal{A}(I^\infty) \geq \mathcal{A}(I)$. 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 $\mathcal{A}(I) \geq 2n_L + 2L$. The consistency of $\mathcal{A}$ leads to $2n_L + 2L \leq (1+\lambda)2L + \nu$, which means $n_L \leq \lambda L + \nu$.

We now construct another instance I^ω by concatenating ω 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 $\mathcal{A}(I^\omega) \geq \omega \cdot (2n_L + 2L)$. Another solution for instance I^ω 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, $\textsc{Opt}(I^\omega) \leq L + \omega \cdot 2(n_L + 1)$. 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

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

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

4 The General Case with k Servers: Upper Bound

To prove Theorem 1, we follow again the three-step approach. We first show that the cost of LambdaDC is close to the cost of FtP, then we bound the cost of FtP by the cost of an optimal solution Opt, which implies an error-dependent competitive ratio, and finally, we give a robustness bound. The key contribution 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.

4.1 Error-dependent competitive ratio

In the first step of the analysis, we compare the performance of LambdaDC and FtP. Let I be an arbitrary instance. Note that $\lambda = 0$ implies $\textsc{Alg}(I) = \textsc{FtP}(I)$, so we now assume that $\lambda \in (0,1]$. We define a new potential function Φ as follows. Let $s_1, \ldots, s_k$ be the servers of LambdaDC and let $x'_1, \ldots, x'_k$ be the servers of FtP. For $1 \leq i < j \leq k$ and $\ell = \min\{j-i, k-(j-i)\} - 1$ we define $\delta_{ij} = \lambda^\ell$. 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 Ψ comes from the targeted competitive ratio. Then, in Θ, 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

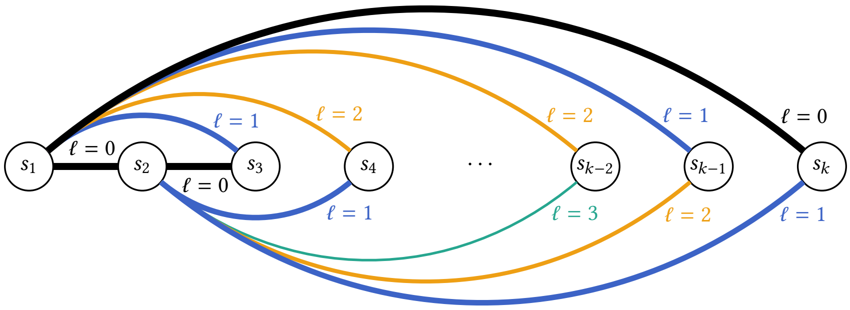


Figure 1: Visualization of all incident δ_{ij}-weights of the servers s_1 and s_2. The thickness (resp. color) of an arc indicates the influence of the corresponding distance in Φ.

distance of λ. Hence, correcting the position of this neighbor means that the next server moves by a distance λ^2. Therefore, this geometric decrease in the consequences of a movement also appears in the expression of Θ. The symmetric increase when $j-i$ grows is more difficult to explain intuitively, but is required to compensate the modifications of Ψ. The coefficients of Θ are illustrated in Figure 1.

Before The proof of the following technical observation is deferred to Appendix A.

Observation 10. *For every $k > 3$, we have $\frac{\alpha(k)}{\lambda} = \alpha(k-2) + \frac{1}{\lambda} + 1$.*

The analysis of LambdaDC requires evaluating the evolution of Φ after each request. The following lemma characterizes how a move of LambdaDC influences Θ.

Lemma 11. *Let $i \le \lfloor k/2 \rfloor$. If s_i moves from p to $p+x$, Θ changes by*

$$(-x) \cdot \left(1 + \alpha(k-2) - 2\sum_{j=0}^{i-2} \lambda^j\right).$$

Proof. Assume w.l.o.g. that the position of s_i decreases by one, that is, the server moves one unit to the left. Consider servers s_j, s'_j such that $j' + \ell = i = j - \ell$ for some $1 \le \ell \le i-1$. Since $\delta_{ij'} = \delta_{ij}$ we observe that the changes to the terms $\delta_{ij}|s_i - s_j|$ and $\delta_{ij'}|s_i - s'_j|$ of Θ cancel out. Hence, as $i \le \lfloor k/2 \rfloor$, the change of Θ due to the move of s_i is equal to $\sum_{j=2i}^{k} \delta_{ij}$. We now prove the statement depending on the parity of k.

(i) If k is odd, $k - 2i + 1$ is even. By definition,

$$\sum_{j=2i}^{k} \delta_{ij} = \sum_{j=2i}^{k} \lambda^{\min\{j-i, k-(j-i)\}-1} = 2\sum_{j=2i}^{(k-1)/2+i} \lambda^{j-i-1} = 2\sum_{j=i-1}^{(k-1)/2-1} \lambda^j$$
$$= 2 + 2\sum_{j=1}^{(k-3)/2} \lambda^j - 2\sum_{j=0}^{i-2} \lambda^j = 1 + \alpha(k-2) - 2\sum_{j=0}^{i-2} \lambda^j.$$

(ii) If k is even, $k - 2i + 1$ is odd, and there is a single term where the minimum in the definition of δ_{ij} in achieved by both conditions. Hence,

$$\sum_{j=2i}^{k} \lambda^{\min\{j-i, k-(j-i)\}-1} = 2\sum_{j=2i}^{k/2-1+i} \lambda^{j-i-1} + \lambda^{k/2-1} = 2\sum_{j=i-1}^{k/2-2} \lambda^j + \lambda^{k/2-1}$$
$$= 2 + 2\sum_{j=1}^{k/2-2} \lambda^j + \lambda^{k/2-1} - 2\sum_{j=0}^{i-2} \lambda^j = 1 + \alpha(k-2) - 2\sum_{j=0}^{i-2} \lambda^j. \qquad \square$$

Now, we are ready to analyze the performance of LambdaDC with respect to FtP.

Lemma 12. *For every instance I and $\lambda \in [0,1]$, there is some $c > 0$ that only depends on the initial configuration such that* $\text{Alg}(I) \leq \alpha(k) \cdot \text{FtP}(I) + c$.

Proof. Suppose that the next request appears. First FtP moves some server x'_i towards the request, and the distance to s_i increases by at most ΔFtP. Since this move only affects Ψ, $\Delta\Phi \leq \alpha(k)/\lambda \cdot \Delta\text{FtP}$. Second LambdaDC moves. We distinguish whether the request is between two servers or not, and assert for both cases $\Delta\Phi \leq -1/\lambda \cdot \Delta\text{Alg}$.

(a) Let the request be located w.l.o.g on the left of s_1. Thus, s_1 moves towards it and charges cost ΔAlg. The fact that some server x'_j must already be on r_t implies with Lemma 11 that

$$\Delta\Phi \leq -\frac{\alpha(k)}{\lambda}\Delta\text{Alg} + \left(1 + \alpha(k-2) - 2\sum_{j=0}^{1-2}\lambda^j\right)\Delta\text{Alg}.$$

Rearranging and using Observation 10 gives the claimed bound, that is

$$\left(-\alpha(k-2) - \frac{1}{\lambda} - 1 + 1 + \alpha(k-2)\right)\Delta\text{Alg} = -\frac{1}{\lambda}\Delta\text{Alg}.$$

(b) Let the request be between s_i and s_{i+1}. Assume w.l.o.g. that FtP serves it with x'_j and $j \leq i$. For ease of exposition, we assume that s_i travels distance 1 and s_{i+1} distance λ. Hence, $\Delta\text{Alg} = 1 + \lambda$. Since $j \leq i$, we know that x'_i must be located on the right of x'_j. Hence, the distance between s_i and x'_i decreases by 1, but the distance between s_{i+1} and x'_{i+1} increases by at most λ. Thus, $\Delta\Psi \leq \alpha(k)/\lambda \cdot (\lambda - 1)$. The change of Θ is clearly bounded from above by the case where s_i moves distance λ and s_{i+1} moves distance 1 for $i + 1 \leq \lfloor k/2 \rfloor$. Combining Lemma 11 for both servers gives

$$\begin{aligned}\Delta\Theta &= 1 + \alpha(k-2) - 2\sum_{j=0}^{i-1}\lambda^j - \lambda\left(1 + \alpha(k-2) - 2\sum_{j=0}^{i-2}\lambda^j\right)\\ &= -1 - \lambda + \alpha(k-2) - \lambda\alpha(k-2).\end{aligned}$$

Using this and Observation 10, we can bound the increase of the potential by

$$\begin{aligned}\Delta\Phi &\leq \frac{\alpha(k)}{\lambda}(\lambda - 1) - 1 - \lambda + \alpha(k-2) - \lambda\alpha(k-2)\\ &= -\frac{\alpha(k)}{\lambda} + \alpha(k-2) = -1 - \frac{1}{\lambda}\\ &= -\frac{1}{\lambda}\Delta\text{Alg}.\end{aligned}$$

□

Secondly, we analyze the performance of FtP with respect to an optimal solution for a fixed error. This is a straight-forward generalization of the 2-server case, and we deferred details to Appendix A.

Lemma 13. *For any instance I, $\text{FtP}(I) \leq \text{Opt}(I) + 2\eta$.*

4.2 The Robustness Bound

Proving a general upper bound on the competitive ratio of LambdaDC, 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 Φ. We generalize the function used for the consistency bound even further by refining the weights, in particular, adding server-dependent weights to the term Ψ 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 $s_1, \ldots, s_k$ be the servers of LambdaDC and let $x_1, \ldots, x_k$ 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

$$\delta_{ij} = \zeta_\ell = \frac{\lambda^\ell + \lambda^{k-2-\ell}}{1+\lambda^{k-2}}.$$

The intuition of the weights in the spreadness part Θ is the same as in the consistency potential function above. However, the new weights ω_i in the matching part Ψ (defined below) require the more complex weights δ_{ij} compared to the simpler λ^ℓ 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

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

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

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

This finishes the definition of the potential function Φ. To prove a robustness guarantee for LambdaDC, we show bounds on the change of Φ when the algorithms (LambdaDC and Opt) move their servers. To that end, several preliminary results will become handy. We first observe that the values $d_1, \ldots, d_k$ correlate with the change of Θ when LambdaDC moves a server.

Observation 14. *Let $i \le \lfloor k/2 \rfloor$. If server s_i moves from p to $p+x$, Θ changes by $(-x) \cdot d_i$.*

Proof. Assume w.l.o.g. that the position of s_i decreases by one, that is, the server moves one unit to the left. Consider servers s_j, s'_j such that $j' + \ell = i = j - \ell$ for some $1 \le \ell \le i-1$. Since $\delta_{ij'} = \delta_{ij}$ we observe that the changes to the terms $\delta_{ij}|s_i - s_j|$ and $\delta_{ij'}|s_i - s'_j|$ of Θ cancel out. Hence, as $i \le \lfloor k/2 \rfloor$, it suffices to consider the distances of s_i to servers s_j with $j \ge 2i$. Therefore,

$$\Delta\Theta = \sum_{j=2i}^{k} \delta_{ij} = \begin{cases} \sum_{\ell=i-1}^{k/2-2} 2\zeta_\ell + \zeta_{k/2-1} & \text{if } k \text{ is even, and} \\ \sum_{\ell=i-1}^{(k-3)/2} 2\zeta_\ell & \text{if } k \text{ is odd.} \end{cases}$$

The definition of ζ_ℓ implies that this is indeed equal to d_i. □

Next, we give several algebraic transformations of γ. Their proofs are deferred to Appendix A.

Lemma 15. *The following statements are true:*

(i) $\gamma = 2\lambda^{k-1}/(1+\lambda^{k-2})$.

(ii) For all $1 \le i \le \lfloor k/2 \rfloor$, *it holds* $(1+\lambda)\gamma = \lambda^{i+1} d_i - \lambda^i d_{i+1}$.

(iii) If k *is even, it holds* $\gamma = \lambda d_1 + (1+\lambda)\sum_{j=2}^{k/2}(-1)^{j-1} d_j$.

These preliminary results enable us to prove two more involved observations about the weights chosen for our potential function. The proofs are deferred to Appendix A. The first observation is important for all cases where a request appears between two servers. Recall the definition of Φ. If s_i moves with speed λ and s_{i+1} with speed 1, the changes to Ψ (increase or decrease) are scaled by $\beta\gamma\lambda\omega_i$ regarding s_i and $\beta\gamma\omega_{i+1}$ regarding s_{i+1}. If i is even, we can easily use the definition of ω, since the denominators cancel. However, if i is odd, we use the following alternative representation of the ω-weights.

Observation 16. *For* $2 \le i \le \lceil k/2 \rceil$, ω_i *is equal to*

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

The second observation is an upper and a lower bound of the ω-weights regardless of the corresponding server. The lower bound is necessary to show that $\Phi \ge 0$, while we use the upper bound to give an easy upper bound on the increase of the potential when the optimal solution moves, independently of its chosen server.

Observation 17. *The values* $\omega_1, \ldots, \omega_k$ *are at least* 0 *and at most* 1.

Before proving formally our robustness bound by exhaustively reviewing all possible moves and bounding the corresponding changes of Φ, we give some intuition.

We choose the scaling parameter γ such that the decrease of Φ exactly matches the required lower bound for the case where the request is outside 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 Φ decreases enough. The definition of the ω values ensures that a wrong prediction gives a tight bound on the decrease of Φ for LambdaDC's move, while a correct prediction still guarantees a loose bound.

Lemma 18. *For any instance* I *and* $\lambda \in (0,1]$, *there is some* $c \ge 0$ *that only depends on the initial configuration such that* $\textsc{Alg}(I) \le \beta(k) \cdot \textsc{Opt}(I) + c$.

Proof. Note that Observation 17 implies $\Phi \ge 0$. Suppose that the next request arrives. First the optimal solution increases due to Observation 17 the potential by at most $\beta\gamma\Delta\textsc{Opt}$ while LambdaDC remains in its previous configuration. Second LambdaDC moves. In the remaining proof we demonstrate that the potential decreases by at least $\gamma\Delta\textsc{Alg}$, which proves the Lemma. We look at the following set of exhaustive cases that occur when LambdaDC makes its move. Assume by scaling that in each case the fast server moves distance 1.

(a) Let the request w.l.o.g. be on the left of s_1. Hence, $\Delta\textsc{Alg} = 1$, and Θ increases by d_1 due to Observation 14. Since x_1 cannot be on the right side of the request, the potential changes by

$$\Delta\Phi = -\beta\gamma + d_1 = -(d_1 - \gamma) + d_1 = -\gamma\Delta\textsc{Alg}.$$

The remaining cases tackle the situations where the request is located between the two servers s_i and s_{i+1}. Without loss of generality we only look at those cases where $i \leq \lfloor k/2 \rfloor$, since the others hold by the symmetry of the line and by the symmetry of Φ.

(b) Let $1 \leq i \leq \lceil k/2 \rceil - 1$ and suppose that s_i is predicted while the optimal solution serves the request with x_j for some $j > i$. Note that $\Delta\textsc{Alg} = 1 + \lambda$. The change of Φ is at most

$$\Delta\Phi \leq \beta\gamma(\omega_i - \lambda\omega_{i+1}) - d_i + \lambda d_{i+1}.$$

By using the definition of ω_i if i is odd and Observation 16 if i is even, this is equal to

$$d_i - \lambda d_{i+1} - (1+\lambda)\gamma - d_i + \lambda d_{i+1} = -\gamma(1+\lambda) = -\gamma\Delta\textsc{Alg}.$$

(c) Let $1 \leq i \leq \lceil k/2 \rceil - 1$ and suppose that s_{i+1} is predicted while the optimal solution serves the request with x_j for some $j \leq i$. The change of Φ is at most

$$\Delta\Phi \leq \beta\gamma(\omega_{i+1} - \lambda\omega_i) - \lambda d_i + d_{i+1}.$$

By using the definition of ω_i if i is even and Observation 16 if i is odd, this is equal to

$$\lambda d_i - d_{i+1} - (1+\lambda)\gamma - \lambda d_i + d_{i+1} = -\gamma(1+\lambda) = -\gamma\Delta\textsc{Alg}.$$

(d) Let $1 \leq i \leq \lceil k/2 \rceil - 1$ and suppose that s_i is predicted while the optimal solution serves the request with x_j for some $j \leq i$. The change of Φ is at most

$$\Delta\Phi \leq \beta\gamma(\lambda\omega_{i+1} - \omega_i) - d_i + \lambda d_{i+1}.$$

By using the definition of ω_i if i is odd and Observation 16 if i is even, this is equal to

$$\begin{aligned}
&- d_i + \lambda d_{i+1} + (1+\lambda)\gamma - d_i + \lambda d_{i+1} \\
&= -\gamma(1+\lambda) + 2(\lambda d_{i+1} - d_i + (1+\lambda)\gamma) \\
&= -\gamma(1+\lambda) + 2(\lambda d_{i+1} + \lambda^2 d_i - \lambda^2 d_i - d_i + (1+\lambda)\gamma).
\end{aligned}$$

Applying Lemma 15(ii) yields

$$\begin{aligned}
&- \gamma(1+\lambda) + 2\left((\lambda^2 - 1)d_i - (1+\lambda)\frac{\gamma}{\lambda^{i-1}} + (1+\lambda)\gamma\right) \\
&\leq -\gamma(1+\lambda) = -\gamma\Delta\textsc{Alg}.
\end{aligned}$$

(e) Let $1 \leq i \leq \lceil k/2 \rceil - 1$ and suppose that s_{i+1} is predicted while the optimal solution serves the request with x_j for some $j > i$. The change of Φ is at most

$$\Delta\Phi \leq \beta\gamma(\lambda\omega_i - \omega_{i+1}) - \lambda d_i + d_{i+1}.$$

By using the definition of ω_i if i is even and Observation 16 if i is odd, we can conclude

$$\begin{aligned}
&- \lambda d_i + d_{i+1} + (1+\lambda)\gamma - \lambda d_i + d_{i+1} \\
&= -(1+\lambda)\gamma + 2((1+\lambda)\gamma - \lambda d_i + d_{i+1}).
\end{aligned}$$

Using Lemma 15(ii) gives

$$\begin{aligned}
&- (1+\lambda)\gamma + 2\left((1+\lambda)\gamma - (1+\lambda)\frac{\gamma}{\lambda^i}\right) \\
&\leq -\gamma(1+\lambda) = -\gamma\Delta\textsc{Alg}.
\end{aligned}$$

If k is even, there are two additional cases which occur when the request is located between the two middle servers $s_{k/2}$ and $s_{k/2+1}$. Note that these cases cannot be covered by the previous ones, since the ω-weights of the servers on both sides of the request are equal.

(f) Let the request be between $s_{k/2}$ and $s_{k/2+1}$, and suppose that $s_{k/2}$ is predicted while the optimal solution serves r with x_j for some $j > k/2$. The change of Φ is at most

$$\Delta\Phi \leq \beta\gamma(\omega_{k/2} - \lambda\omega_{k/2}) - \lambda d_{k/2} - d_{k/2}. \tag{1}$$

For the rest of this case, we distinguish two cases according to the parity of $k/2$, and show that $\Delta\Phi \leq -\gamma\Delta\textsc{Alg}$.

If $k/2$ is even, (1) is by Observation 16 and the definition of $\omega_{k/2}$ equal to

$$2\lambda\sum_{j=1}^{k/4} d_{2j-1} - 2\sum_{j=1}^{k/4-1} d_{2j} - d_{k/2} - \gamma$$
$$- \left(2\lambda\sum_{j=1}^{k/4-1} d_{2j} - 2\sum_{j=1}^{k/4-1} d_{2j+1} + \lambda d_{k/2} + (2+\lambda)\gamma\right) - \lambda d_{k/2} - d_{k/2}.$$

Noting that $2\lambda\sum_{j=1}^{k/4} d_{2j-1} = 2\lambda d_1 + 2\lambda\sum_{j=1}^{k/4-1} d_{2j+1}$ gives

$$-(3+\lambda)\gamma + 2\left(\lambda d_1 + (1+\lambda)\sum_{j=2}^{k/2}(-1)^{j-1}d_j\right).$$

We can conclude that this is equal to $-\gamma(1+\lambda)$ by Lemma 15(iii).

Similarly, *if $k/2$ is odd*, (1) is by Observation 16 and the definition of $\omega_{k/2}$ equal to

$$2\lambda\sum_{j=1}^{(k/2-1)/2} d_{2j} - 2\sum_{j=1}^{(k/2-3)/2} d_{2j+1} - d_{k/2} + \gamma$$
$$- \left(2\lambda\sum_{j=1}^{(k/2-1)/2} d_{2j-1} - 2\sum_{j=1}^{(k/2-1)/2} d_{2j} + \lambda d_{k/2} + \lambda\gamma\right) - \lambda d_{k/2} - d_{k/2}.$$

Noting that $2\lambda\sum_{j=1}^{(k/2-1)/2} d_{2j-1} = 2\lambda d_1 + 2\lambda\sum_{j=1}^{(k/2-3)/2} d_{2j+1}$ yields

$$\gamma - \lambda\gamma + 2\left(-\lambda d_1 + (1+\lambda)\sum_{j=2}^{k/2}(-1)^{j}d_j\right) = \gamma - \lambda\gamma - 2\left(\lambda d_1 + (1+\lambda)\sum_{j=2}^{k/2}(-1)^{j-1}d_j\right).$$

This is equal to $-\gamma(1+\lambda)$ by Lemma 15(iii).

(g) Let the request be between $s_{k/2}$ and $s_{k/2+1}$, and suppose that $s_{k/2}$ is predicted while the optimal solution serves r with x_j for some $j \leq k/2$. The change of Φ is at most

$$\Delta\Phi \leq \beta\gamma(\lambda\omega_{k/2} - \omega_{k/2}) - \lambda d_{k/2} - d_{k/2},$$

which is bounded from above by the previous case. Hence, $\Delta\Phi \leq -\gamma\Delta\textsc{Alg}$. □

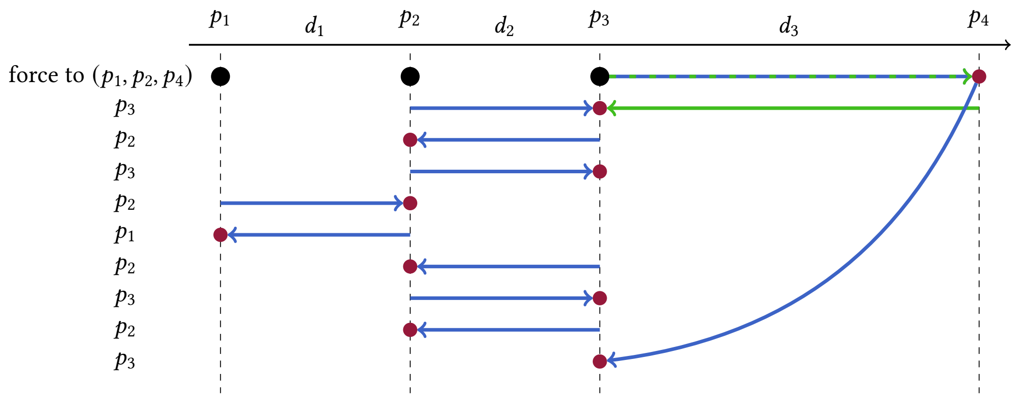


Figure 2: Instance I for $k = 3$. The prediction is drawn green. The blue moves indicate an exemplary schedule of an algorithm.

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 request 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. In the following, we make this intuition precise, sketch our worst-case construction and finally give a formal proof of Theorem 2.

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. 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 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 ε, 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 first sketch the main idea and present details afterwards. 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 this happens. 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 λ 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. It is then possible to 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.

We now present a formal and precise proof following the above ideas. Let $\lambda \in (0, 1]$. Recall that $\rho(k) = \sum_{i=0}^{k-1} \lambda^i$. Let $\mathcal{A}$ be a $\rho(i)$-consistent locally-consistent and memory-constrained deterministic online algorithm for the i-server problem on the line, for all $i \leq k$. The objective is to show that $\mathcal{A}$ is then at least $\beta(k)$-robust, with $\beta(k) = \sum_{i=0}^{k-1} \lambda^{-i}$.

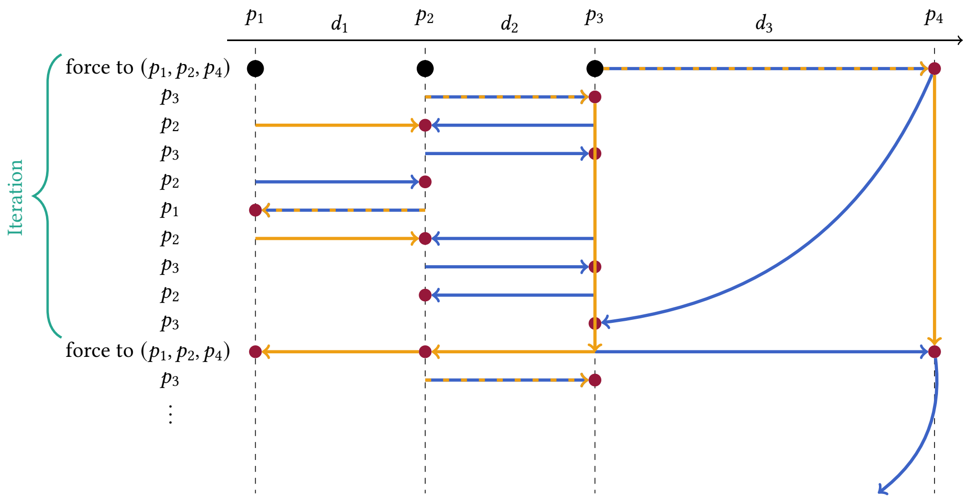


Figure 3: Instance I^ω for $k = 3$. The alternative (better) solution is drawn orange. The prediction and the exemplary moves of the algorithm are the same as in instance I for each iteration.

Let $p_1 \leq \ldots \leq p_{k+1}$ be points on the line with inter-distances $d_1, \ldots, d_k$, where for $1 \leq i \leq k-1$, $d_i = 1$, and $d_k > 1$ is arbitrarily large. We also define an arbitrarily small constant $\varepsilon > 0$ and say that a server *covers* a point p_i if it is at most a distance ε away from it. We refer to smaller positions on the line as *left*. Let $P := \{p_1, \ldots, p_k\}$. In the following we inductively construct an instance. In their initial configuration, *i.e.* at time $t = 0$, the k servers, $s_1, \ldots, s_k$, are located at $p_1, \ldots, p_k$. We assume that servers never overpass each other to simplify the notations. Then, we *force* the servers to $p_1, \ldots, p_{k-1}, p_{k+1}$ (see the memory-constrained definition). The instance terminates when $\mathcal{A}$ places s_k to cover p_k. At any time $t > 0$, the next requested point $r(t)$ is the *leftmost* point (*i.e.* the point with the smallest index) which is *not covered* by any server of $\mathcal{A}$. If p_1 is not covered and s_1 is on the left of $p_2 - \varepsilon$ then $r(t)$ is the second leftmost uncovered server. If p_1 is not covered and s_1 covers p_2 but did not serve it since leaving p_1 then $r(t)$ is p_2 and $r(t + 1)$ (next in time) is p_1. At any time $t > 0$, we denote the instance composed of $r_1, \ldots, r_t$ by I_t.

At every point in time, we give $\mathcal{A}$ the prediction that suggests serving a request at some point p_i with the server s_i. An exception is the first request, where s_k is predicted (note that the first request is always located at p_{k+1}). We now show that this construction rule is well-defined.

Lemma 19. *The construction ends after a finite number of requests.*

Proof. For the sake of contradiction, assume that the construction does not end after a finite number of requests. Hence, every request r except the first one must be in the set P, and by construction, no server covered r in the previous configuration. Thus, the server that serves r must have been moved with some cost at least ε, which implies that $\mathcal{A}$ has unbounded cost.

Now consider *any* infinite instance I^P which starts with a request at p_{k+1} followed by requests contained in P. An optimal solution for I^P is to serve the first request with s_k and then to move it immediately back to the set P, such that every point in P contains a server. Hence, the total cost of an optimal solution is constant. Therefore the consistency of $\mathcal{A}$ would be infinite, as the prediction given to $\mathcal{A}$ corresponds to the optimal solution, which is a contradiction. $\square$

Due to Lemma 19, we assume for the rest of this section that the construction ends after n steps, and we define $I = I_n$.

We first focus on the cost that $\mathcal{A}$ charges for I. Let D_i be the distance traveled by the server s_i in $\mathcal{A}$. Using the locally-consistent definition, we show the following relation between D_i's:

Lemma 20. *For all $i \leq k$, for ε small enough, we have $D_1 \leq \lambda^{i-1} D_i + O_k(\varepsilon D_i + d_1)$, where the notation $O_k(\cdot)$ treats k as a constant.*

Proof. Let $i \in \{2, 3, \dots, k\}$ and assume by induction that the relation is true for all $j < i$. Note that it is trivial for $i = 1$.

We denote by $\mathcal{A}_i(I)$ the cost of $\mathcal{A}$ restricted to the i leftmost servers. Consider the i leftmost servers and we apply the locally-consistent property of $\mathcal{A}$ on these servers. Let I' be the corresponding instance on i servers, where requests not served by $\{s_1, \dots, s_i\}$ are replaced by requests to the new position of s_i.

Consider the algorithm FtP serving I' following the initial predictions as in the locally-consistent definition. There are two types of requests: a point p_ℓ for $\ell < i$ is served at no cost by s_ℓ, and any other request is served by s_i. The objective is to upper bound $\text{FtP}(I')$ by D_i plus negligible terms. Consider all requests different from p_i served by s_i in FtP, and let r_1 and r_2 be two *consecutive* requests in this set (there can be other requests not belonging to this set between r_1 and r_2). These requests are based on requests of I outside of $\{p_1 \dots p_i\}$, which means that each of these points (except p_1) is covered by a server of $\mathcal{A}$ before the request, and that s_i also went to r_1 and r_2 in $\mathcal{A}$, at the time at which they are requested in I'. A technical difficulty here is that s_i does not need to be *exactly* at p_i before these requests: it can be within a distance of ε. There are several cases to analyze.

- If p_i is not requested between r_1 and r_2, then FtP pays the shortest path between r_1 and r_2, so at most how much s_i travels in $\mathcal{A}$.
- If s_i goes on p_i between r_1 and r_2 in $\mathcal{A}$, then FtP also pays at most how much s_i travels in $\mathcal{A}$.
- If p_i is requested between r_1 and r_2 and s_i does not go on p_i in $\mathcal{A}$, we focus on the subinstance I^* starting from the request r_1 and ending just before r_2 is requested. Let $\text{FtP}(I^*)$, $\mathcal{A}_i(I^*)$ and D^*_ℓ be the restrictions of $\text{FtP}(I)$, $\mathcal{A}_i(I)$ and D_ℓ to I^*. Note that $\text{FtP}(I^*) \leq D^*_i + \varepsilon$ as FtP moves s_i to r_1 then back to p_i whereas $\mathcal{A}$ needs only to move s_i to r_1 and then near p_i. The objective is now to show that this additive term ε is negligible compared to $\mathcal{A}_i(I^*)$, for which we need a further case distinction.
  - If r_1 is at least a distance $\sqrt{\varepsilon}$ away from p_i, then $\text{FtP}(I^*)$ moves s_i by a distance which is close to D^*_i. Specifically, we have $D^*_i \geq 2\sqrt{\varepsilon} - 2\varepsilon \geq \sqrt{\varepsilon}$ for ε small enough, and the relation $\text{FtP}(I^*) \leq D^*_i + \varepsilon$ implies $\text{FtP}(I^*) \leq (1 + \sqrt{\varepsilon}) D^*_i$.
  - If r_1 is at most a distance $\sqrt{\varepsilon}$ away from p_i, we get $\text{FtP}(I^*) \leq 2\sqrt{\varepsilon} + \varepsilon$ and we distinguish two cases which are slightly different if $i = 2$ or $i > 2$.
    * If $i > 2$ then the cost of $\mathcal{A}_i$ on I^* is at least $A_i(I^*) \geq D^*_{i-1} > d_{i-1} - \varepsilon = 1 - \varepsilon$ as p_i must have been served by s_{i-1} (previously located near p_{i-1}) if it was not served by s_i. We therefore obtain $\text{FtP}(I^*) \leq 3\sqrt{\varepsilon} \cdot D^*_{i-1}$.
    * If $i = 2$, the difference is that s_1 may be initially located anywhere between p_1 and p_2. s_1 serves $p_i = p_2$ when it is requested (as this case assumes s_2 does not serve p_2 in I^*), and then must serve p_1 by the definition of the instance I. Therefore, the cost of $\mathcal{A}_i$ on I^* is at least $A_i(I^*) \geq D^*_1 \geq d_1 = 1$. We thus obtain $\text{FtP}(I^*) \leq 3\sqrt{\varepsilon} \cdot D^*_1$.

Summing over all subinstances, we obtain the following inequality:

$$\text{FtP}(I') \leq (1 + \sqrt{\varepsilon}) D_i + 3\sqrt{\varepsilon} \cdot \sum_{\ell=1}^{i-1} D_\ell \leq D_i + 3\sqrt{\varepsilon} \cdot \mathcal{A}_i(I).$$

As the initial and final configurations are identical up to a distance of d_1 for s_1 and ε for other servers, the locally-consistent property for I' yields

$$\mathcal{A}_i(I) \leq \rho(i) D_i + 3\sqrt{\varepsilon}\rho(i) \cdot \mathcal{A}_i(I) + O(\varepsilon k^2 + d_1 k).$$

For ε small enough, we have $3\sqrt{\varepsilon}\rho(i) < 1/2$, which implies that $\mathcal{A}_i(I) \leq 2\rho(i)D_i + O(\varepsilon k^2 + d_1 k)$. Using this new bound on $\mathcal{A}_i(I)$ on the right-hand side of the above inequality leads to the following:

$$\mathcal{A}_i(I) \leq \rho(i)D_i + O(\varepsilon k^2 + d_1 k + \sqrt{\varepsilon}\rho(i)^2 D_i)$$
$$\sum_{\ell=1}^{i} D_\ell \leq D_i + (\rho(i) - 1)D_i + O(\varepsilon k^2 + d_1 k + \sqrt{\varepsilon}\rho(i)^2 D_i)$$
$$\sum_{\ell=1}^{i-1} D_\ell \leq (\rho(i) - 1)D_i + O(\varepsilon k^2 + d_1 k + \sqrt{\varepsilon}\rho(i)^2 D_i)$$

We now use the induction hypothesis to lower bound D_ℓ by $\lambda^{1-\ell}D_1 + O_k(\varepsilon D_i + d_1)$ and replace $\rho(i)$ by its expression, before dividing all sides by $\sum_{\ell=0}^{i-2} \lambda^{-\ell}$. We use the notation $O_k(\cdot)$ to avoid detailing the irrelevant dependencies on k, note that $\rho(i)$ depends only on λ and k so does not appear inside the notation $O_k(\cdot)$.

$$\sum_{\ell=1}^{i-1} \frac{1}{\lambda^{\ell-1}} D_1 \leq \sum_{\ell=1}^{i-1} \lambda^\ell D_i + O_k(\varepsilon D_i + d_1)$$
$$D_1 \leq \lambda^{i-1} D_i + O_k(\varepsilon D_i + d_1). \qquad \square$$

We build the instance I^ω repeating the instance I ω times, starting directly by the force to $p_1, \dots, p_{k-1}, p_{k+1}$. The predictions for each iteration correspond to the predictions defined in instance I. We now bound the optimal cost for this instance.

Lemma 21. $\textsc{Opt}(I^\omega) \leq 2d_k + \omega \cdot (D_1 + 2\sum_{i=2}^{k-1} d_i)$.

Proof. Consider the following schedule for I^ω: at each iteration, move $k-1$ servers to $p_2, \dots, p_{k+1}$ and alternate between p_1 and p_2 with s_1. We now analyze how many alternations we need to do in each iteration. By definition of the instance, p_1 is only requested if s_1 has served p_2 since it last left p_1. Therefore, the distance traveled by s_1 equals D_1. At the end of the iteration, we move back the $k-2$ middle servers, giving the target cost. $\square$

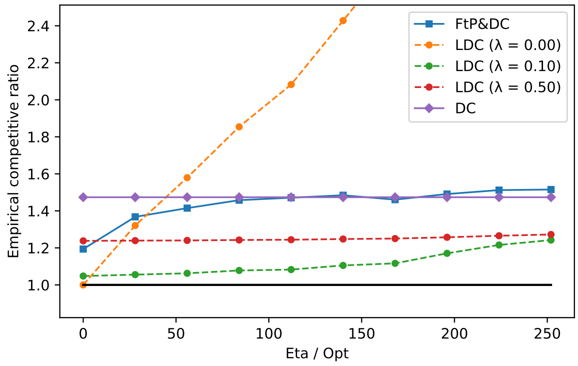
Proof of Theorem 2. As $\mathcal{A}$ is memory-constrained, its behavior on each iteration of I is identical, s_k is at p_k initially, then the k servers are forced to the points $p_1, \dots, p_{k-1}, p_{k+1}$ before continuing the requests. Therefore $\mathcal{A}$ must pay at least d_k to serve the first force operation, and then must make the same decisions in all iterations.

Using Lemma 21, the competitive ratio of $\mathcal{A}$ for instance I^ω is therefore at least

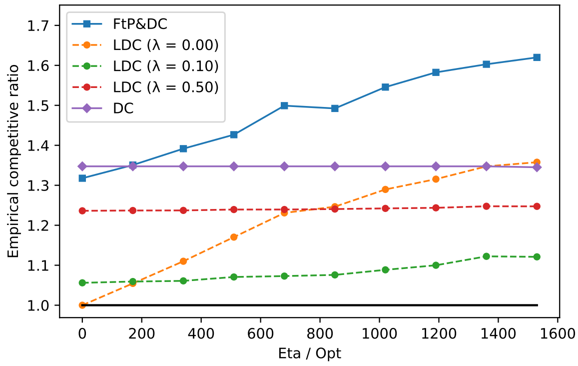
$$\frac{\mathcal{A}(I^\omega)}{\textsc{Opt}(I^\omega)} \geq \frac{\omega \cdot \sum_{i=1}^{k} D_i}{2d_k + \omega \cdot (D_1 + 2\sum_{i=2}^{k-1} d_i)}$$
$$\xrightarrow{\omega\to\infty} \frac{\sum_{i=1}^{k} D_i}{D_1 + 2\sum_{i=2}^{k-1} d_i}.$$

Consider d_k arbitrarily large (but still small compared to ω). If D_1 is bounded by a constant, then the competitive ratio is unbounded, so $\mathcal{A}$ is not robust. Otherwise, the terms d_i become negligible compared to D_1, and we show that the limit of the competitive ratio is lower bounded by the desired robustness expression, using Lemma 20 (which implies that d_1 is also negligible compared to any D_i):

$$\frac{\mathcal{A}(I^\omega)}{\textsc{Opt}(I^\omega)} \xrightarrow{d_k\to\infty} \frac{\sum_{i=1}^{k} D_i}{D_1} \geq \sum_{i=0}^{k-1} \frac{1}{\lambda^i + O_k(\varepsilon + \frac{d_1}{D_{i+1}})} \xrightarrow{\varepsilon\to 0} \sum_{i=0}^{k-1} \lambda^{-i}. \qquad \square$$



(a) Results for $k = 2$, $p = 10$, $b = 28$.



(b) Results for $k = 10$, $p = 10$, $b = 170$.

Figure 4: Means over empirical competitive ratios of all instance-prediction combinations per bin.

Simple algebraic manipulations (given in the appendix) show that the consistency of LambdaDC is best possible up to a factor of 2.

Lemma 22. *For every $\lambda \in [0, 1]$, $\alpha(k) < 2\rho(k)$ and, thus, the consistency of LambdaDC is best possible up to a factor of 2.*

6 Experiments

We supplement our theoretical results by empirically comparing our learning-augmented algorithm LambdaDC with the classical online algorithm ignoring predictions DoubleCoverage [15] and the previously proposed prediction-based algorithm FtP&DC [3] on real world data[1]. We generate instances with 1000 requests in the interval $[0, 4000]$ based on the BrightKite-Dataset [14], which is composed of sequences of scaled coordinates where users check into social media apps. This dataset was used previously in this field to evaluate and compare learning-augmented algorithms for cashing problems [3, 34]. We extract sequences with a length of 1000 checkins, normalize the scaling of latitudes, and use these values as the positions of the requests on the line. All servers start at the same initial random position. We generate 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. Additionally, we use an optimal solution of the instance as the perfect prediction. Our procedure carefully samples predictions for given parameters p and b. We set $p = 10$ and b to the largest value such that for at least 40 instances we find these evenly distributed predictions. Other instances are discarded.

We simulate the algorithms on all instances and predictions and compute the empirical competitive ratios. The results are displayed in Fig. 4. They show well that, for a reasonable choice of λ ($0.1 \leq \lambda \leq 0.5$) LambdaDC outperforms both DoubleCoverage and FtP&DC throughout almost all generated relative prediction errors.

7 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 [15] in an intuitive

[1]The source code is hosted on GitHub: https://github.com/Mountlex/kserver.

way and admits a (nearly) tight error-dependent competitive analysis, based on new potential functions, and outperforms other methods from the literature.

Clearly, it would be interesting to see whether our results generalize to more general metric spaces than the line. In fact, our upper bounds for the 2-server problem can be extended to tree metrics, see [31], 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 focussed on memory-constrained algorithms, leaving open a more precise quantification of the power of memory. Also, the recent success on randomized k-server algorithms [11, 30] raises the question whether and how randomized algorithms can benefit from (ML) predictions.

Another direction is to consider generalizations of the k-server problem, such as the k-taxi problem [17], where a server can get relocated after serving a request. We developed a learning-augmented algorithm for the 2-taxi problem in general metrics [31] by building upon a deterministic algorithm proposed in [17]. Interestingly, it was recently shown that DoubleCoverage is a best possible deterministic algorithm for the k-taxi problem on HSTs [12]. Possibly, our algorithm LambdaDC performs well when being adapted to this setting.

References

[1] Spyros Angelopoulos. Online search with a hint. In *ITCS*, volume 185 of *LIPIcs*, pages 51:1–51:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[2] 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.

[3] Antonios Antoniadis, Christian Coester, Marek Eliás, 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.

[4] Antonios Antoniadis, Carsten Fischer, and Andreas Tönnis. A collection of lower bounds for online matching on the line. In *Latin American Symposium on Theoretical Informatics*, pages 52–65. Springer, 2018.

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

[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 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.

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

[11] 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.

[12] Niv Buchbinder, Christian Coester, and Joseph Naor. Online k-taxi via double coverage and time-reverse primal-dual. *CoRR*, abs/2012.02226, 2020.

[13] 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.

[14] 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.

[15] 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.

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

[17] Christian Coester and Elias Koutsoupias. The online k-taxi problem. In *STOC*, pages 1136–1147. ACM, 2019.

[18] Christian Coester and Elias Koutsoupias. Towards the k-server conjecture: A unifying potential, pushing the frontier to the circle. *CoRR*, abs/2102.10474, 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] Chen-Yu Hsu, Piotr Indyk, Dina Katabi, and Ali Vakilian. Learning-based frequency estimation algorithms. In *ICLR*, 2019.

[21] Piotr Indyk, Ali Vakilian, and Yang Yuan. Learning-based low-rank approximations. In *NeurIPS*, pages 7400–7410, 2019.

[22] Zhihao Jiang, Debmalya Panigrahi, and Kevin Sun. Online algorithms for weighted paging with predictions. In *ICALP*, volume 168 of *LIPIcs*, pages 69:1–69:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

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

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

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

[26] Elias Koutsoupias and David Scot Taylor. The CNN problem and other k-server variants. *Theoretical Computer Science*, 324(2):347 – 359, 2004. Online Algorithms: In Memoriam, Steve Seiden.

[27] 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.

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

[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] James R. Lee. Fusible HSTs and the randomized k-server conjecture. In *FOCS*, pages 438–449. IEEE Computer Society, 2018.

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

[32] 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.

[33] 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.

[34] 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.

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

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

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

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

[46] 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 Proofs for Section 4

Observation 10. *For every $k > 3$, we have $\frac{\alpha(k)}{\lambda} = \alpha(k-2) + \frac{1}{\lambda} + 1$.*

Proof. We prove this statement depending on the parity of k. If k is odd, $k-2$ is also odd. By definition of $\alpha(k)$,

$$1 + \frac{1}{\lambda} + \alpha(k-2) = \frac{1}{\lambda} + 2 + 2\sum_{i=1}^{(k-3)/2} \lambda^i = \frac{1}{\lambda} + 2\sum_{i=1}^{(k-1)/2} \lambda^{i-1} = \frac{\alpha(k)}{\lambda}.$$

If k is even, $k-2$ is also even, and we conclude

$$\begin{aligned} 1 + \frac{1}{\lambda} + \alpha(k-2) &= \frac{1}{\lambda} + 2 + 2\sum_{i=1}^{(k-2)/2-1} \lambda^i + \lambda^{(k-2)/2} \\ &= \frac{1}{\lambda} + 2\sum_{i=1}^{k/2-1} \lambda^{i-1} + \lambda^{(k-2)/2} \\ &= \frac{\alpha(k)}{\lambda}. \end{aligned}$$

□

Lemma 13. *For any instance I, $\textsc{FtP}(I) \leq \textsc{Opt}(I) + 2\eta$.*

Proof. Let r_t be any request and let $C_{t-1} = (x'_1, \ldots, x'_k)$ and $C_t = (x_1, \ldots, x_k)$ be the configurations of the fixed optimal solution before and after it serves r_t. For FtP we define similarly $C^P_{t-1} = (s'_1, \ldots, s'_k)$ and $C^P_t = (s_1, \ldots, s_k)$. Assume that the optimal solution serves r_t with x'_i and FtP with s'_j. Since we can assume that the optimal solution and FtP only move one server per request by postponing unnecessary moves, we can compute the distance between two configurations by looking at the distance these servers move, *i.e.*,

$$\begin{aligned} \Delta\textsc{FtP}_t &= d(C^P_{t-1}, C^P_t) = |s'_j - s_j| \text{ and} \\ \Delta\textsc{Opt}_t &= d(C_{t-1}, C_t) = |x'_i - x_i|. \end{aligned}$$

The definitions of the errors η_{t-1} and η_t give lower bounds concerning the moved servers, that is

$$\begin{aligned} \eta_{t-1} &= \sum_{\ell=1}^{k} |x'_\ell - s'_\ell| \geq |x'_j - s'_j| + |x'_i - s'_i| \text{ and} \\ \eta_t &= \sum_{\ell=1}^{k} |x_\ell - s_\ell| \geq |x_j - s_j| + |x_i - s_i|. \end{aligned}$$

Using the triangle inequality and symmetry of $|\cdot|$, and noting that $s_i = s'_i$ and $x_j = x'_j$, yields

$$\begin{aligned} \Delta\textsc{FtP}_t &= d(C^P_{t-1}, C^P_t) \\ &= |s'_j - s_j| + |s'_i - s_i| \\ &\leq |s'_j - x'_j| + |x'_j - x_j| + |x_j - s_j| + |s'_i - x'_i| + |x'_i - x_i| + |x_i - s_i| \\ &= |x'_j - x_j| + |x'_i - x_i| + |x'_j - s'_j| + |x'_i - s'_i| + |x_j - s_j| + |x_i - s_i| \\ &\leq d(C_{t-1}, C_t) + \eta_{t-1} + \eta_t \\ &= \Delta\textsc{Opt}_t + \eta_{t-1} + \eta_t. \end{aligned}$$

Summing over all requests gives the asserted bound. □

Lemma 15. *The following statements are true:*

(i) $\gamma = 2\lambda^{k-1}/(1+\lambda^{k-2})$.

(ii) For all $1 \leq i \leq \lfloor k/2 \rfloor$, *it holds* $(1+\lambda)\gamma = \lambda^{i+1}d_i - \lambda^i d_{i+1}$.

(iii) If k *is even, it holds* $\gamma = \lambda d_1 + (1+\lambda)\sum_{j=2}^{k/2}(-1)^{j-1}d_j$.

Proof. (i) Since

$$\lambda^{k-1}(\beta-1) = \lambda^{k-1}\sum_{\ell=1}^{k-1}\lambda^{-\ell} = \sum_{\ell=0}^{k-2}\lambda^{\ell},$$

we conclude by the definition of γ and d_1 that

$$\gamma = \frac{d_1}{\beta-1} = \frac{2}{(1+\lambda^{k-2})(\beta-1)}\sum_{\ell=0}^{k-2}\lambda^{\ell} = \frac{2}{1+\lambda^{k-2}}\lambda^{k-1}.$$

(ii) Simplifying the right-hand side gives

$$\begin{aligned}
\lambda^{i+1}d_i - \lambda^i d_{i+1} &= \frac{2\lambda^i}{1+\lambda^{k-2}}\left(\lambda\sum_{\ell=i-1}^{k-1-i}\lambda^{\ell} - \sum_{\ell=i}^{k-2-i}\lambda^{\ell}\right) = \frac{2\lambda^i}{1+\lambda^{k-2}}\left(\sum_{\ell=i}^{k-i}\lambda^{\ell} - \sum_{\ell=i}^{k-2-i}\lambda^{\ell}\right) \\
&= \frac{2\lambda^i}{1+\lambda^{k-2}}\left(\lambda^{k-i}+\lambda^{k-i-1}\right) = 2\left(\frac{\lambda^k+\lambda^{k-1}}{1+\lambda^{k-2}}\right) \\
&= (1+\lambda)\left(\frac{2\lambda^{k-1}}{1+\lambda^{k-2}}\right).
\end{aligned}$$

Then, Lemma 15(i) concludes the proof.

(iii) Assume that k is even. The right-hand side is equal to

$$(-1)^{k/2-1}\lambda d_{k/2} + \sum_{j=1}^{k/2-1}(-1)^{j-1}(\lambda d_j - d_{j+1}).$$

By Lemma 15(ii),

$$(-1)^{k/2-1}\lambda d_{k/2} + (1+\lambda)\sum_{j=1}^{k/2-1}(-1)^{j-1}\frac{\gamma}{\lambda^j},$$

which is equal to

$$\frac{1}{\lambda^{k/2-1}}\left((-1)^{k/2-1}\lambda^{k/2}d_{k/2} + \sum_{j=0}^{k/2-2}(-1)^{k/2-j}(\lambda^j+\lambda^{j+1})\gamma\right).$$

We proceed by applying a telescoping sum argument. Since $k/2-(k/2-2)=2$, the last term of the sum $\lambda^{k/2-1}\gamma$ is positive. Similarly, the first term $\lambda^0\gamma$ has the same sign as $(-1)^{k/2-0} = -(-1)^{k/2-1}$. The remaining terms of the sum cancel out. Thus, it remains

$$\frac{1}{\lambda^{k/2-1}}\left(\lambda^{k/2-1}\gamma + (-1)^{k/2-1}(\lambda^{k/2}d_{k/2}-\gamma)\right).$$

By definition, $d_{k/2} = 2\lambda^{k/2-1}/(1+\lambda^{k-2})$. Hence, $\lambda^{k/2}d_{k/2}$ is equal to γ by Lemma 15(i). We conclude that the expression is indeed equal to γ.

□

Observation 16. *For* $2 \le i \le \lceil k/2 \rceil$, ω_i *is equal to*

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

Proof. We first note that for every $1 \le j \le \lfloor k/2 \rfloor - 1$, applying Lemma 15(ii) with j and $j+1$ yields

$$\lambda d_j - d_{j+1} = \frac{1+\lambda}{\lambda^j}\gamma = \lambda^2 d_{j+1} - \lambda d_{j+2}. \tag{2}$$

We now prove the statement separately for all even and all odd values of $2 \le i \le \lceil k/2 \rceil$ by induction.

As induction base for the even case, we first prove the claim for $i = 2$. Indeed,

$$\omega_2 = \frac{\lambda d_2 + (2+\lambda)\gamma}{\beta\gamma\lambda} = \frac{d_2 + (2+\lambda)\gamma/\lambda}{\beta\gamma} = \frac{2\lambda d_1 - d_2 - \gamma}{\beta\gamma}.$$

Note that the last equality derives from Lemma 15(ii). Now assume that $i > 2$ is even. The induction hypothesis for $i - 2$ yields in this case

$$\beta\gamma\lambda \cdot \omega_{i-2} = 2\lambda^2 \sum_{j=1}^{i/2-1} d_{2j-1} - 2\lambda \sum_{j=1}^{i/2-2} d_{2j} - \lambda d_{i-2} - \lambda\gamma. \tag{3}$$

We want to prove that $\beta\gamma\lambda \cdot \omega_i$ is equal to

$$2\lambda^2 \sum_{j=1}^{i/2} d_{2j-1} - 2\lambda \sum_{j=1}^{i/2-1} d_{2j} - \lambda d_i - \lambda\gamma,$$

which can be rearranged to

$$2\lambda^2 \sum_{j=1}^{i/2-1} d_{2j-1} - 2\lambda \sum_{j=1}^{i/2-2} d_{2j} - \lambda d_{i-2} - \lambda\gamma - \lambda d_i - \lambda d_{i-2} + 2\lambda^2 d_{i-1}.$$

Replacing the right side of (3) in the above expression yields

$$\beta\gamma\lambda \cdot \omega_{i-2} - \lambda d_i - \lambda d_{i-2} + 2\lambda^2 d_{i-1}.$$

Since (2) gives $2(1+\lambda^2)d_{i-1} = 2\lambda(d_{i-2} + d_i)$, and by the definition of ω_{i-2}, this can be rewritten to

$$\begin{aligned} &2\lambda \sum_{j=1}^{i/2-2} d_{2j} - 2\sum_{j=1}^{i/2-2} d_{2j+1} + \lambda d_{i-2} + (2+\lambda)\gamma - 2d_{i-1} + \lambda d_{i-2} + \lambda d_i \\ &= 2\lambda \sum_{j=1}^{i/2-1} d_{2j} - 2\sum_{j=1}^{i/2-1} d_{2j+1} + (2+\lambda)\gamma + \lambda d_i, \end{aligned}$$

which is indeed equal to $\beta\gamma\lambda \cdot \omega_i$ by definition.

As induction base for the odd case, we start by proving the claim for $i = 3$, that is

$$\omega_3 = \frac{2\lambda d_2 - d_3 + \gamma}{\beta\gamma} = \frac{2\lambda^2 d_2 - \lambda d_3 + \lambda\gamma}{\beta\gamma\lambda} = \frac{2\lambda d_1 - 2d_2 + \lambda d_3 + \lambda\gamma}{\beta\gamma\lambda}.$$

In the last equality we used that $2(1+\lambda^2)d_2 = 2\lambda(d_1+d_3)$ by (2). Now assume that $i > 3$ is odd. By induction hypothesis for $i-2$,

$$\beta\gamma\lambda \cdot \omega_{i-2} = 2\lambda \sum_{j=1}^{(i-1)/2-1} d_{2j-1} - 2 \sum_{j=1}^{(i-1)/2-1} d_{2j} + \lambda d_{i-2} + \lambda\gamma. \tag{4}$$

Consider the claimed expression for $\beta\gamma\lambda \cdot \omega_i$, that is

$$2\lambda \sum_{j=1}^{(i-1)/2} d_{2j-1} - 2 \sum_{j=1}^{(i-1)/2} d_{2j} + \lambda d_i + \lambda\gamma,$$

which we can rearrange to

$$2\lambda \sum_{j=1}^{(i-1)/2-1} d_{2j-1} - 2 \sum_{j=1}^{(i-1)/2-1} d_{2j} + \lambda d_{i-2} + \lambda\gamma + \lambda d_{i-2} + \lambda d_i - 2d_{i-1}.$$

Replacing the right side of (4) in the above expression gives

$$\beta\gamma\lambda \cdot \omega_{i-2} + \lambda d_{i-2} + \lambda d_i - 2d_{i-1}.$$

Noting that (2) gives $2(1+\lambda^2)d_{i-1} = 2\lambda(d_{i-2}+d_i)$ yields together with the definition of ω_{i-2} the equivalent expression

$$\begin{aligned}
&2\lambda^2 \sum_{j=1}^{(i-1)/2-1} d_{2j} - 2\lambda \sum_{j=1}^{(i-3)/2-1} d_{2j+1} - \lambda d_{i-2} + \lambda\gamma + 2\lambda^2 d_{i-1} - \lambda d_{i-2} - \lambda d_i \\
&= 2\lambda^2 \sum_{j=1}^{(i-1)/2} d_{2j} - 2\lambda \sum_{j=1}^{(i-3)/2} d_{2j+1} - \lambda d_i + \lambda\gamma.
\end{aligned}$$

Since this is by definition equal to $\beta\gamma\lambda \cdot \omega_i$, we can also conclude this case. ◻

Observation 17. *The values $\omega_1, \ldots, \omega_k$ are at least 0 and at most 1.*

Proof. By definition, $\omega_1 = \omega_k = 1$. We now show this property for ω_i depending on whether $2 \le i \le k-1$ is even or odd.

Assume that i is even. By definition, the numerator of ω_i is equal to

$$2\lambda \sum_{j=1}^{i/2-1} d_{2j} - 2 \sum_{j=1}^{i/2-1} d_{2j+1} + \lambda d_i + (2+\lambda)\gamma.$$

Using the definition of d_i and Lemma 15(i) gives

$$\begin{aligned}
&\frac{2}{1+\lambda^{k-2}} \left(\lambda \sum_{j=1}^{i/2-1} \sum_{\ell=2j-1}^{k-1-2j} 2\lambda^\ell - \sum_{j=1}^{i/2-1} \sum_{\ell=2j}^{k-2-2j} 2\lambda^\ell + \lambda \sum_{\ell=i-1}^{k-1-i} \lambda^\ell \right) + (2+\lambda)\gamma \\
&= \frac{2}{1+\lambda^{k-2}} \left(\sum_{j=1}^{i/2-1} 2(\lambda^{k-2j} + \lambda^{k-1-2j}) + \sum_{\ell=i}^{k-i} \lambda^\ell \right) + (2+\lambda)\gamma \\
&= \frac{2}{1+\lambda^{k-2}} \left(\sum_{\ell=k-i+1}^{k-2} 2\lambda^\ell + \sum_{\ell=i}^{k-i} \lambda^\ell \right) + (2+\lambda)\gamma \\
&= \frac{2}{1+\lambda^{k-2}} \left(\sum_{\ell=k-i+1}^{k-1} 2\lambda^\ell + \sum_{\ell=i}^{k-i} \lambda^\ell \right) + \lambda\gamma
\end{aligned}$$

Since $\beta\gamma\lambda = \lambda\gamma + \lambda d_1 \geq 0$, we conclude that $\omega_i \geq 0$. Further, using the fact that $\sum_{\ell=k-i+1}^{k-1} \lambda^\ell \leq \sum_{\ell=1}^{i-1} \lambda^\ell$ yields

$$\frac{2}{1+\lambda^{k-2}} \left(\sum_{\ell=k-i+1}^{k-1} 2\lambda^\ell + \sum_{\ell=i}^{k-i} \lambda^\ell \right) + \lambda\gamma \leq \frac{2}{1+\lambda^{k-2}} \sum_{\ell=1}^{k-1} \lambda^\ell + \lambda\gamma = \lambda d_1 + \lambda\gamma,$$

and we conclude that $\omega_i \leq 1$.

Assume that i is odd. By definition, the numerator of ω_i is equal to

$$2\lambda \sum_{j=1}^{(i-1)/2} d_{2j} - 2 \sum_{j=1}^{(i-3)/2} d_{2j+1} - d_i + \gamma.$$

Using definitions gives

$$\begin{aligned}
&\frac{2}{1+\lambda^{k-2}} \left(\lambda \sum_{j=1}^{(i-1)/2} \sum_{\ell=2j-1}^{k-1-2j} 2\lambda^\ell - \sum_{j=1}^{(i-3)/2} \sum_{\ell=2j}^{k-2-2j} 2\lambda^\ell - \sum_{\ell=i-1}^{k-1-i} \lambda^\ell \right) + \gamma \\
&= \frac{2}{1+\lambda^{k-2}} \left(\sum_{j=1}^{(i-1)/2} \sum_{\ell=2j}^{k-2j} 2\lambda^\ell - \sum_{j=1}^{(i-3)/2} \sum_{\ell=2j}^{k-2-2j} 2\lambda^\ell - \sum_{\ell=i-1}^{k-1-i} \lambda^\ell \right) + \gamma \\
&= \frac{2}{1+\lambda^{k-2}} \left(\sum_{j=1}^{(i-1)/2} 2(\lambda^{k-2j} + \lambda^{k-1-2j}) + \sum_{\ell=i-1}^{k-1-i} 2\lambda^\ell - \sum_{\ell=i-1}^{k-1-i} \lambda^\ell \right) + \gamma \\
&= \frac{2}{1+\lambda^{k-2}} \left(\sum_{\ell=k-i}^{k-2} 2\lambda^\ell + \sum_{\ell=i-1}^{k-1-i} \lambda^\ell \right) + \gamma.
\end{aligned}$$

Since $\beta\gamma = \gamma + d_1 \geq 0$, we conclude that $\omega_i \geq 0$. Further, using the fact that $\sum_{\ell=k-i}^{k-2} \lambda^\ell \leq \sum_{\ell=0}^{i-2} \lambda^\ell$ yields

$$\frac{2}{1+\lambda^{k-2}} \left(\sum_{\ell=k-i}^{k-2} 2\lambda^\ell + \sum_{\ell=i-1}^{k-1-i} \lambda^\ell \right) + \gamma \leq \frac{2}{1+\lambda^{k-2}} \sum_{\ell=0}^{k-2} \lambda^\ell + \gamma = d_1 + \gamma,$$

and we conclude that $\omega_i \leq 1$. □

B Proofs for Section 5

Lemma 22. *For every $\lambda \in [0,1]$, $\alpha(k) < 2\rho(k)$ and, thus, the consistency of* LambdaDC *is best possible up to a factor of* 2.

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

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

□