NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

BSc THESIS

Online Learning-Augmented Algorithms

Konstantinos T. Lakis

Supervisors: **Themistoklis Gouleakis,** Postdoctoral Researcher, MPI
Panagiotis Stamatopoulos, Assistant Professor, UoA

ATHENS

June 2022

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Άμεσοι Αλγόριθμοι με Προβλέψεις

Κωνσταντίνος Θ. Λάκης

Επιβλέποντες: **Θεμιστοκλής Γουλεάκης,** Μεταδιδακτορικός Ερευνητής, MPI
Παναγιώτης Σταματόπουλος, Επίκουρος Καθηγητής, ΕΚΠΑ

ΑΘΗΝΑ

Ιούνιος 2022

BSc THESIS

Online Learning-Augmented Algorithms

Konstantinos T. Lakis
S.N.: 1115201700069

SUPERVISORS: **Themistoklis Gouleakis,** Postdoctoral Researcher, MPI
Panagiotis Stamatopoulos, Assistant Professor, UoA

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Άμεσοι Αλγόριθμοι με Προβλέψεις

Κωνσταντίνος Θ. Λάκης
Α.Μ.: 1115201700069

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Θεμιστοκλής Γουλεάκης,** Μεταδιδακτορικός Ερευνητής, MPI
Παναγιώτης Σταματόπουλος, Επίκουρος Καθηγητής, ΕΚΠΑ

ABSTRACT

In this thesis we study the emerging field of Online Learning-Augmented Algorithms. These are Online Algorithms which in addition to the regular input also receive predictions about their input beforehand. First, a few online problems and their analysis are given to introduce the premise. Then, we give a presentation of some important and representative results in the area of Learning-Augmented Algorithms, with the goal of explaining the techniques used in their design. Finally, we give some original results regarding the Online Travelling Salesman Problem on the Line under such a setting.

In the classical TSP on the Line, there is a stream of requests released over time along the real line. The goal is to minimize the makespan of the algorithm. We distinguish between the *open* variant and the *closed* one, in which we additionally require the algorithm to return to the origin after serving all requests. The state of the art is a 1.64-competitive algorithm and a 2.04-competitive algorithm for the closed and open variants, respectively [15]. In both cases, a tight lower bound is known [8, 15].

In both variants, our primary prediction model involves predicted *positions* of the requests. We introduce algorithms that (i) obtain a tight 1.5 competitive ratio for the closed variant and a 1.66 competitive ratio for the open variant in the case of perfect predictions, (ii) are robust against unbounded prediction error, and (iii) are smooth, i.e., their performance degrades gracefully as the prediction error increases.

Moreover, we further investigate the learning-augmented setting in the *open* variant by additionally considering a prediction for the last request served by the optimal offline algorithm. Our algorithm for this enhanced setting obtains a 1.33 competitive ratio with perfect predictions while also being smooth and robust, beating the lower bound of 1.44 we show for our original prediction setting for the open variant. Also, we provide a lower bound of 1.25 for this enhanced setting.

SUBJECT AREA: Online Learning-Augmented Algorithms

KEYWORDS: Online TSP on the Line, Consistency, Smoothness, Robustness, Competitive Ratio

ΠΕΡΙΛΗΨΗ

Σε αυτή την πτυχιακή εργασία μελετάμε το πρόσφατα εκκολαπτόμενο πεδίο των Άμεσων Αλγορίθμων με Προβλέψεις (Online Learning-Augmented Algorithms). Αυτοί είναι άμεσοι αλγόριθμοι οι οποίοι λαμβάνουν και προβλέψεις για την είσοδό τους, προτού αυτή τους εμφανιστεί. Αρχικά, εξετάζουμε μερικά κλασσικά Προβλήματα με Άμεση Ανταπόκριση (Online Problems) και την ανάλυσή τους. Έπειτα, παρουσιάζουμε μερικά από τα πιό σημαντικά και αντιπροσωπευτικά πρόσφατα αποτελέσματα στον χώρο των Άμεσων Αλγορίθμων με Προβλέψεις, με σκοπό να περιγράψουμε τις τεχνικές που χρησιμοποιούνται στην σχετική βιβλιογραφία. Τέλος, δίνουμε μερικά νέα θεωρητικά αλλά και πειραματικά αποτελέσματα όσον αφορά το Πρόβλημα του Πλανώδιου Πωλητή στον Άξονα με Άμεση Ανταπόκριση (Online TSP on the Line) με ύπαρξη προβλέψεων.

Στην κλασσική έκδοση αυτού του προβλήματος, μία σειρά από αιτήματα (requests) εμφανίζονται με το πέρασμα του χρόνου πάνω στον άξονα των πραγματικών αριθμών (real line). Ο στόχος είναι να ελαχιστοποιηθεί η χρονοκαθυστέρηση (makespan), δηλαδή ο χρόνος που χρειάζεται ο αλγόριθμος για να ικανοποιήσει όλα τα αιτήματα. Υπάρχει η *ανοιχτή* (open) έκδοση του προβλήματος και η *κλειστή* (closed), στην οποία επίσης απαιτούμε από τον αλγόριθμο να επιστρέψει στο αρχικό σημείο (origin). Οι καλύτεροι αλγόριθμοι που υπάρχουν είναι 1.64- και 2.04-ανταγωνιστικοί (competitive) αντίστοιχα [15]. Και στις δύο εκδόσεις, υπάρχει εφαπτόμενο κάτω φράγμα (lower bound) [8, 15].

Το κύριο μοντέλο προβλέψεων (prediction model) που χρησιμοποιούμε περιέχει προβλέψεις για τις *θέσεις* (positions) των αιτημάτων. Δίνουμε αλγόριθμους οι οποίοι (i) έχουν την ιδιότητα της συνέπειας (consistency), δηλαδή είναι 1.5- και 1.66-ανταγωνιστικοί με τέλειες προβλέψεις για την κλειστή και ανοιχτή έκδοση αντίστοιχα, (ii) είναι εύρωστοι (robust) ενάντια σε οσοδήποτε λανθασμένες προβλέψεις, και (iii) είναι ομαλοί (smooth), δηλαδή η απόδοση τους χειροτερεύει ελεγχόμενα ανάλογα με την ποιότητα των προβλέψεων.

Επιπλέον, μελετάμε βαθύτερα την ανοιχτή έκδοση, επαυξάνοντας το μοντέλο προβλέψεων μας με μία πρόβλεψη για το ποιό είναι το αίτημα το οποίο θα ικανοποιούσε τελευταίο ένας βέλτιστος Αλγόριθμος με Καθυστερημένη Ανταπόκριση (Offline Algorithm). Ο αλγόριθμος που δίνουμε για αυτή την περίπτωση είναι 1.33-συνεπής (consistent) χωρίς να χάνει την ομαλότητα (smoothness) και την ευρωστία (robustness) του, καταρρίπτοντας το κάτω φράγμα του 1.44 που αποδεικνύουμε για την μη-επαυξημένη περίπτωση. Τέλος, δείχνουμε ένα κάτω φράγμα του 1.25 για την επαυξημένη περίπτωση.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Άμεσοι Αλγόριθμοι με Προβλέψεις

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Το πρόβλημα του πλανώδιου πωλητή στον άξονα, Συνέπεια, Ομαλότητα, Ευρωστία, Λόγος Ανταγωνιστικότητας

CONTENTS

LIST OF FIGURES

LIST OF TABLES

PREFACE

The main part of this thesis was carried out during a research internship at the Max Planck Institute for Informatics, where I was fortunate to have Dr. Themistoklis Gouleakis as my advisor. We also collaborated with Golnoosh Shahkarami, a Ph.D. student at MPI. I sincerely thank them both for their useful guidance throughout this process.

Additionally, I would like to thank Prof. Panagiotis Stamatopoulos, who has been a great teacher throughout my undergraduate studies in various courses and is the most adept user of the Socratic method I have encountered. I owe a substantial part of my motivation to do research to his patience in giving me essential feedback to countless exploratory questions I frequently posed to him.

1. ONLINE ALGORITHMS

1.1 What is an Online Problem and why do we care?

Many problems of significance in today's world concern themselves with the optimization of some sort of choice. Choosing which car to buy, where to go for vacation and even deciding on a life partner all essentially boil down to the maximization of an (implicit) profit function or equivalently the minimization of an (implicit) cost function. Such problems are called optimization problems. In many cases, we need to make decisions regarding these problems without having all the pieces of information necessary. For example, we must decide whether to buy an electric or a gas-fueled car without knowing that the price of gas will double in the following years. These types of problems are reffered to as online, precisely because the algorithm that solves them must act in response to information arriving partially and over time.

It turns out that many important real-world problems can be modeled by online problems, making the study of these problems worthwhile even from a practical viewpoint. In order to get a better feel for this, in the next section we present some of these problems along with their applications. In some of these problems, we also describe their corresponding offline version, in which the algorithm can see the entire input in advance.

1.2 Examples of Online Problems

In this section, we give some examples of online problems which we will later analyze in more detail. We will examine the Paging Problem and Online TSP on the Line.

1.2.1 The Paging Problem

Undoubtedly one of the most well-studied online problems is the Paging Problem. This problem concerns itself with the management of a computer's memory. We are supplied with two types of memory. The cache is small but fast and the main memory is large but slow. The cache is split into k cells. The main memory consists of N pages. Each cell of the cache can contain at most one page of the main memory at any time.

We are presented with some page requests over time. If the page requested is contained in a cell of the cache, we say we have a cache hit. In the opposite case, we have a cache miss. When this happens, we must bring the requested page into the cache. This may require the eviction of a page that is currently in one of the cache's cells. The requirement in this problem is to decide which page to evict when there is need to, with the goal of minimizing the number of cache misses.

An obvious application of this problem is the management of actual memory systems inside computers. As the amount of data is becoming increasingly vast, memory latency is becoming the bottleneck in many computational problems. An efficient paging algorithm can limit the number of disk accesses in large databases, which greatly improves the turnaround time for requests.

In the offline version of this problem, we get to know the entire page request sequence beforehand. In such a case, the optimal strategy involves evicting the page that will be required latest from now on. This strategy is known as Belady's rule [13]. Obviously, this cannot directly be mapped to a general real world scenario, since we would need to be able to predict the future. However, some programs have predictable memory access patterns that enable us to foresee their page requests. In other cases, we can get a pretty good estimate of the memory access patterns of a program after we have run it multiple times.

This realization has led to the development of algorithms that take advantage of this ”predictability”. This strategy does work well in practice, but theoretically speaking is not guaranteed to. It is precisely this ”paradox” that led to the development of beyond-worst case analysis, one branch of which is the field of learning-augmented algorithms.

1.2.2 The Online Travelling Salesman Problem on the Line

A less known problem, and the one to which we present original contributions, is the Online TSP on the Line. In this problem, we are presented with requests along the real number line. These requests are each released at a specific point in time. Only after a request is released are we informed of its existence and are able to serve it. We are tasked with managing an agent that is initially on the origin. We can move the agent with up to unit speed to either direction at any point in time. The agent can serve a request by simply travelling to it after it has been released. There is no notion of serve time - the agent can leave the request immediately after reaching it.

There are two versions of the goal in this problem. In the *closed* version, the goal is to minimize the time it takes the agent to serve all the requests *and* return to the origin. In the *open* version, the agent need not return to the origin. In this thesis, we concern ourselves with the *closed* version.

An application of this problem is the management of robots operating on warehouses. Suppose we have deployed a robot in a row of storage shelves inside a warehouse of a company like Amazon. The robot is tasked with retrieving items from the storage shelves, which are requested by customers. Of course, it only gets to know the details of a request after it is given by the customer. Also, we can't know when the final request is released, since a customer can always make another order. Our (or equivalently, the robot's) goal is to bring all the required items to a specific pick-up point (the ”origin”) so they can be shipped. It is obvious that solving the theoretical version of the problem is equivalent to managing the robot efficiently. An implicit assumption made here is that the robot will always be able to carry as many items as necessary.

The offline version of this problem is still useful from a practical viewpoint. As you may have guessed, the difference is that we know at $t = 0$ when and where each request will occur. A real world scenario where this could be used is the delivery of parcels across a street. We know which parcels should be delivered and where. The release time of a request corresponds to a time after which the person who expects the parcel will be available to receive it.

1.3 The evaluation of Online Algorithms

1.3.1 Competitive analysis

In this subsection, we will give an overview of the framework first presented by Sleator and Tarjan at [44] that is used to evaluate the performance of online algorithms. We compare the performance of online algorithms based on what an algorithm that knows all the input in advance can achieve. We refer to such an algorithm by OPT. We use $OPT(I)$ to refer to the cost of the solution provided by OPT for input I. To refer to the algorithm we are examining, we simply swap OPT with ALG in these definitions. For the formal (mathematical) part, we will always refer to minimization problems and cost functions. This is not a restriction, since any maximization problem can be transformed to a minimization problem. We will however discuss situations where the goal may either be to minimize or maximize the cost.

With this in mind, we are interested in the competitive ratio achieved by ALG.

Definition 1 (Competitive ratio). *We say that an online algorithm ALG for a minimization problem P is c-competitive if for every input I there exists a constant $\alpha \geq 0$ such that*

$$ALG(I) \leq c \cdot OPT(I) + \alpha$$

When ALG is c-competitive, we also say that ALG achieves a competitive ratio of c.

When $\alpha = 0$, we say that ALG is *strictly c-competitive*. In competitive analysis we essentially try to determine this constant c.

A different and helpful way to look at competitive analysis is to consider the problem as a game between an online player (whose actions are governed by ALG) and an adversary ADV that aims to maximize ALG's competitive ratio. ADV is given ALG's description, thus is able to simulate it. Therefore, ADV can design an entire input instance in advance. Then, since ADV knows the entire input before being asked to solve it, it can do so optimally. It is only a matter of finding which input I maximizes $\frac{ALG(I)}{ADV(I)} = \frac{ALG(I)}{OPT(I)}$, which is equivalent to the definition given earlier.

We will make use of this viewpoint mainly when proving lower bounds on the competitiveness of some algorithms. In these cases, we will describe the actions of ADV in such a way that it generates an input I_d that is guaranteed to lead to a competitive ratio of at least d. We will say that we describe a d-attack when doing so.

1.3.2 Analysis of the Paging Problem

Before presenting the main results achieved in the study of this problem, we first provide more rigorous notation and problem definition.

Problem Definition

The general idea of the problem has been discussed earlier. Recall that we have N pages in the main memory and k cells in the cache. To formalize things, we give the notation we will use to describe the problem. We will use p_i to refer to page number i in the main memory, where $0 \leq i < N$. Also, we will denote the j-th cell of the cache by c_j, where $0 \leq j < k$. We say that $s(c_j) = p_i$ if the page p_i is currently stored in the cell c_j. If no page is stored in c_j, we have $s(c_j) = \varnothing$.

A series of pages $p_{r_0}, p_{r_1}, \cdots, p_{r_m}$ is requested. If at the time p_{r_x} is requested we have $s(c_j) = p_{r_x}$ for some j, we say we have a cache hit. Else, we have a cache miss. When we have a cache miss, we are obliged to evict one page from a cell c_e and load the requested page on it. This gives a new function s' where $s'(c_e) = p_{r_x}$, while s' is identical to s for all other cells.

The objective here is to minimize the number of cache misses by specifying an eviction strategy. We may now proceed with presenting the most interesting results.

A k-attack

Before we see how we can tackle the problem, let's take a look at the difficulties that may arise for an algorithm. In more detail, we will see that no algorithm can be better than k-competitive by providing a k-attack.

Let's assume that we have $k+1$ pages and pages p_1 through p_k are initially in the cache. The adversary ADV makes T requests in the following manner. Simply, ADV asks for the one page that is currently not in the cache of ALG, i.e. the page p_r that satisfies $\nexists\, j \; s.t. \; s(c_j) = p_r$. One such page always exists, by virtue of the pigeonhole principle and the limitation imposed upon the cells of containing at most one page. Thus, ALG will have to suffer T cache misses.

On the other hand, we shall see that OPT can get away with $\frac{T}{k}$ cache misses. OPT always evicts the page to be requested the furthest in the future. We claim that the *delay* of the request of this page is at least k. Indeed, suppose that for all k pages, their next request (after the cache miss we are considering) is at most $k-1$ requests away. Thus, k pages must be requested in the following $k-1$ requests, a contradiction due to the pigeonhole principle. We now see that for every cache miss, OPT is "safe" for at least the next $k-1$ requests. Thus, OPT incurs at most $\frac{T}{k}$ cache misses.

Combining these two observations, we can see that no ALG can be better than k-competitive.

Some Algorithms for the Paging Problem

In this subsection we will refer to a few algorithms which are examined in a more detailed manner in [17].

LRU (Least Recently Used) is arguably the most popular and practically used algorithm for this problem. As the name suggest, this algorithm chooses to evict the page which was requested the furthest back in the past (or equivalently, least recently). It turns out to be k-competitive, because it belongs to a family of algorithms (marking algorithms) which will later see that is comprised only of k-competitive algorithms.

FIFO (First In First Out) is the next one in popularity. Again as you may have imagined, it evicts the page which was added to the cache the earliest. Equivalently, it evicts the page that has stayed in the cache the longest. Similarly to LRU, FIFO belongs to the family of conservative algorithms, which also contains only k-competitive algorithms.

FWF (Flush When Full) is a naive algorithm that just flushes its entire cache when it is full.

In other words, when all cells contain a page, it immediately evicts all pages. With a very slight modification, this algorithm can be shown to belong to the making algorithms family, thus being k-competitive. This modification involves flushing the cache step by step when page faults occur, choosing one arbitrary marked page to evict each time.

We now proceed to describe the two families of algorithms we referred to. Their competitiveness is intuitive and the proof is only outlined. The interested reader may find this information in Chapter 3 of [17].

Marking algorithms

We fix a series $\sigma = p_{r_1}, p_{r_2}, \cdots, p_{r_m}$ of page requests. We split this series into phases as follows. The first phase begins with p_{r_1}. When the $(k+1)$-th distinct page is requested during a phase -counting the page that initiated this phase- a new phase begins. In other words, each phase contains requests that correspond to exactly k distinct pages, except perhaps for the last phase, which may have less than k distinct pages.

Now consider a marking scheme that works as follows. At the start of every phase, we unmark every page in the cache. When a page is first requested during a phase, we mark it. Then, an algorithm ALG belongs to the marking algorithms family if it never evicts a marked page.

We can easily see that any marking algorithm is indeed k-competitive, because it suffers at most k page faults during any single phase, while OPT incurs at least one page fault in each phase. The first argument can be seen by the fact that once we mark a page, we never evict it again. Thus, whenever we have a page fault, the requested page is marked and cannot be evicted until the next phase. So, each cell can correspond to at most one page fault during each phase, which proves the claim. Additionally, OPT must suffer a page fault for every phase, because it also only has k cells.

Conservative algorithms

The definition of a conservative algorithm is pretty straightforward. An algorithm ALG belongs to the conservative algorithms family if for every input sequence that requests at most k distinct pages, ALG incurs at most k page faults. These input sequences are similar to the phases of the marking algorithms.

The proof that all conservative algorithms are k-competitive is a slight modification of the one used for the marking algorithms.

1.3.3 Analysis of Online TSP on the Line

In order to present the most important results in the study of this problem, we first present a more rigorous description of the problem, given in [15] by Bjelde et al.

Problem Definition

We have an agent that can move along the real line with at most unit speed. We let $pos(t)$ denote the position of the agent at time $t \geq 0$. We also know that $pos(0) = 0$.

With this notation, the speed limitation of the agent can be expressed by the inequality $|pos(t) - pos(t')| \leq |t - t'|$ for all $t,\ t' \geq 0$. A series of requests $\sigma_1, ..., \sigma_n$ arrives over time with $\sigma_i = (a_i; r_i)$. Here, a_i denotes the position of the request and r_i is the release time. We will use $p^R = \max_i\{a_i\}$ to denote the rightmost point of the input and $p^L = \min_i\{a_i\}$ to refer to the leftmost point of the input. From here on, we will use the terms *positive* and *right* interchangeably when refering to points on the real line. The same holds for the terms *negative* and *left*.

Initial results

Earlier work by Ausiello et al in [8] has provided some initial results on the competitive ratio achievable in this problem. In more detail, they give a lower bound of ≈ 1.64 on the competitive ratio achievable by any algorithm. In our terminology, they give a 1.64-attack. Additionally, a $\frac{7}{4}$-competitive algorithm is presented. We will summarize the main points of these results here, since they are relevant to our contributions.

A first lower bound

Before giving the 1.64-attack, we describe a $\frac{3}{2}$-attack. At $t = 1$, ADV places a request on -1 if $pos(1) \geq 0$. Else, it places a request on 1. It is easy to see that ADV can finish by $t = 2$. On the other hand, ALG cannot finish before $t = 3$. Therefore, the competitive ratio in this case is at least $\frac{3}{2}$. Hence, no algorithm can be better than $\frac{3}{2}$-competitive. Note that we will use a similar attack when giving our own lower bound for the case where predictions are used.

The best lower bound

Now we will describe the general idea of the 1.64-attack. We suppose that an algorithm ALG has a competitive ratio of $\rho < \frac{(9+\sqrt{17})}{8}$. Firstly, we argue that $-(2\rho - 3) \leq pos_{ALG}(1) \leq (2\rho - 3)$. To see this, let's suppose that $pos_{ALG}(1) > (2\rho - 3)$. In this case, ADV can just place a request on -1. Then, ALG would need more than $1 + (2\rho - 3) + 2 = 2\rho$ time units to solve the problem, while ADV could do it in exactly 2. Thus, the competitive ratio would be larger than ρ. The case where $pos_{ALG}(1) < -(2\rho - 3)$ similarly leads to a contradiction.

Thus, we may proceed under the assumption that indeed $-(2\rho - 3) \leq pos_{ALG}(1) \leq (2\rho - 3)$. ADV places two requests at 1 and -1, both at $t = 1$. At $t = 3$, ALG cannot have served both requests. Let's assume without loss of generality that it has not served -1. With similar arguments to those of the previous paragraph, we get that $pos_{ALG}(3)$ is *not* inside of the interval $[-(7 - 4\rho), (7 - 4\rho)]$. Indeed, if this is not true, ADV can place a request at 1 at $t = 3$. Then, ADV would finish in 4 time units, while ALG would require more than 4ρ. Because $\rho < \frac{(9+\sqrt{17})}{8}$, we can see that $[-(7-4\rho), (7-4\rho)]$ strictly contains $[-(2-3\rho), (2-3\rho)]$.

Now, we are left with only two possible cases.

1. At $t = 3$, ALG has not yet served 1.

2. At $t = 3$, ALG has served 1 and also $(7 - 4\rho) \leq pos_{ALG}(3) \leq 1$. We know that $pos_{ALG}(3)$ cannot be to the left of $-(7 - 4\rho)$, because $pos_{ALG}(1)$ was inside $[-(2 - 3\rho), (2 - 3\rho)]$. In more detail, 2 time units is just not enough to start within $[-(2 - 3\rho), (2 - 3\rho)]$, go serve the request at 1 and then reach $-(7 - 4\rho)$.

In both of these cases, we see that ALG is at a distance of at most $1 - (7 - 4\rho)$ to one extreme and has not yet served the other. Indeed, this property is sufficient for the rest of proof. The details are a bit technical and of not great intuitive value, thus are omitted. The interested reader can find them in Chapter 3 of [8]. The main takeaway is the strategy of using attacks to obtain properties about an algorithm's behavior, which then provide the basis for further attacks, until one has examined all the cases. We also made use of this strategy in our attempts at providing lower bounds for the learning-augmented setting.

The $\frac{7}{4}$-competitive algorithm (PQR)

As we mentioned, a $\frac{7}{4}$-competitive algorithm is given in [8], going by the name PQR (possibly queue requests). The algorithm is based on the idea of postponing some requests that are relatively close to the origin. This set of postponed requests is named Q. Any other unserved request belongs to the set of (not postponed) requests P. The algorithm optimally serves all the requests in P first. After that, it optimally serves all the requests in Q. We will not go into the details of the algorithm, but it is interesting to note the following. By contstruction, the requests in Q all lay on the same side of the origin at any time. Thus, the tour-planning of the algorithm is always simple in the sense that it moves to one of the two extremes straight after the other and then back to the origin.

With this observation, the essence of the algorithm boils down to choosing which extreme to go for first. Stripping this decision down even more, we only ever need to decide when to *switch* extremes. By that we mean that while we previously had planned to go right first and then left, the release of a new request compels us to instead go left first and then right. How PQR decides this is not so important, but it gives some insight on possible algorithmic ideas for the learning-augmented setting. In the following section we will pay our debt of explicitly presenting an algorithm for the problem.

Newer (and final) results

In their recent paper [15], Bjelde et al manage to match the lower bound given in [8] by Ausiello et al with the design of a ≈ 1.64-competitive algorithm. We give an overview of this algorithm in the following section.

The ≈ 1.64 algorithm

While the proof for its competitive ratio is rather complex and long, the algorithm is actually quite simple. It utilizes a waiting strategy, as opposed to zealously serving the requests like PQR. When the algorithm has decided to move, it chooses which extreme to serve first in a simple but insightful way. By examining the possible attacks, the authors have identified a criterion which governs the choice of the first extreme. Before presenting the algorithm, we will explain how this criterion was chosen. As a sidenote, we attempted to use both of these strategies when trying to devise an algorithm for the learning-augmented setting.

The extreme choice criterion

In order to describe the criterion, we look at a possible attack that can happen. From here on, let $L(t), R(t)$ refer to the positions of the leftmost and rightmost requests of the input respectively, at time t. Also, let $r^L(t)$ and $r^R(t)$ denote their release times. Then, $\sigma^R(t) = (R(t); r^R(t))$. We define $\sigma^L(t)$ in the obvious manner.

Suppose that ALG decides to serve $\sigma^L(t)$ before $\sigma^R(t)$. However, ADV might have chosen to serve $\sigma^R(t)$ first, possibly (we assume it does for this example) as early as possible, i.e. at $r^R(t)$. Let t_0 be the point in time when ALG reaches the origin after serving $\sigma^L(t)$. Also, let t' be the point in time when ADV reaches $L(t)$ after serving $\sigma^R(t)$. We assume for the sake of explaining the attack that $t' \leq t_0$.

With this in mind, let's think about what happens if ADV moves to the left after t' up until t_0. There exists the possibility that ADV will place a request at $p' = -|L(t)| - (t_0 - t') = -t_0 + r^R(t) + |R(t)|$, at time t_0.

Then, ADV can finish at time $t_0 + |p'| = t_0 + |L(t)| + (t_0 - t') = 2t_0 - r^R(t) - |R(t)|$. However, ALG still needs to serve the requests at $R(t)$ and p'. Thus, ALG cannot finish before $t_0 + 2|p'| + 2|R(t)| = 3t_0 - 2r^R(t)$. For ALG to be ρ-competitive, we must ensure that:

$$\frac{3t_0 - 2r^R(t)}{2t_0 - r^R(t) - |R(t)|} \leq \rho \Longleftrightarrow t_0 \geq \frac{\rho|R(t)| - (2-\rho)r^R(t)}{2\rho - 3}$$

If we think about what this tells us, we can see that it practically defines a threshold before which any ALG should *not* have served one extreme and returned to the origin, if it is to be ρ-competitive. At first glance, this seems counterintuitive. After all, we simply took care of a request quite early. If anything, this should mean that we are on the right track, right?

It turns out, as we saw, that this viewpoint is flawed. To see this more clearly, note that serving $L(t)$ in the previous example is of no significance, since it would be served later anyway on the way back from p'. If anything, making an effort to serve it only takes us away from the "safe" spot in the origin. The observation of this threshold's existence leads to the definition of:

$$SR(\sigma^R(t), t) = \frac{\rho|R(t)| - (2-\rho)r^R(t)}{2\rho - 3}$$

Similarly, we define $SR(\sigma^L(t), t)$ as the earliest an algorithm is allowed to have served $\sigma^R(t)$ and return to the origin. The algorithm takes care to never break these thresholds. Surprisingly, this simple idea is enough to guarantee that the algorithm is indeed ≈ 1.64-competitive.

This idea is encapsulated by the extreme choice criterion as follows. If by going to one extreme first (say $\sigma^R(t)$) we would return to the origin too early (i.e. before $SR(\sigma^L(t), t)$), we instead go to the other extreme ($\sigma^L(t)$) first. This is only one of the things we consider when choosing extremes, but is the most important decision the algorithm makes. All the other criteria are simple and obvious.

Without further ado, we present the algorithm's definition in the following section.

Definition of the algorithm

But first, we give some final definitions so the algorithm can make sense. We refer to the position of the extreme with the largest absolute value by $far(t)$. Obviously, $near(t)$ refers to the position of the other extreme.

Now consider two possible routes an optimal agent may follow if they only had to serve $\sigma^L(t)$ and $\sigma^R(t)$. They may immediately travel to $\sigma^L(t)$, wait for it to release and then go serve $\sigma^R(t)$ or do the exact same but with $\sigma^R(t)$ first. We call the fastest of these two routes $T_{greedy}(t)$. Note that $|T_{OPT}| \geq |T_{greedy}(t)|$. Additionally, $G_1(t)$ refers to the position of the first extreme served by $T_{greedy}(t)$. $G_2(t)$ is defined analogously.

Lastly, we define $R(\sigma, t)$ as the earliest point in time that ALG can move from $pos(t)$ to α and then back to the origin, where α is the position of the request σ.

We can finally give the definition of the algorithm. We simply provide a rule through which the algorithm updates its planned trajectory whenever a new (significant) request is released, since this is sufficient to fully describe the behavior of the algorithm.

```
Algorithm 1: UPDATE($t, pos(t), \sigma^L(t), \sigma^R(t)$) for the closed online TSP problem
Input  : Current time $t$
         Current position $pos(t)$
         Unserved extreme requests $\sigma^L(t)$ and $\sigma^R(t)$
Output : A closed TSP tour serving all remaining requests

$t_{wait} \leftarrow \rho|T_{greedy}(t)| - (|pos(t) - far(t)| + |far(t)| + 2|near(t)|)$;

// Safe route because we get to wait
if $t_{wait} \geq t$ then
    return $waituntil(t_{wait}) \oplus move(far(t)) \oplus move(near(t)) \oplus move(0)$;
end

// We get to follow the greedy tour
if $sign(pos(t)) = sign(G_1(t))$ or $R(G_2(t), t) < SR(G_1(t), r)$ then
    return $move(G_1(t)) \oplus move(G_2(t)) \oplus move(0)$;
end

// Opposite to greedy tour but we know $R(G_2(t), t) \geq SR(G_1(t), r)$
return $move(G_2(t)) \oplus move(G_1(t)) \oplus move(0)$;
```

Competitiveness of the algorithm

This algorithm achieves a competitive ratio of $\frac{(9+\sqrt{17})}{8} \approx 1.64$. The proof of to this is rather complex and long (it spans about 15 pages in the paper) and is therefore only outlined. The main idea is to focus on the last time the algorithm defines its trajectory. If it follows the safe tour, the competitiveness is obvious. When it gets to follow the greedy tour, competitiveness derives from the fact that we *closely* follow the greedy tour. The only case that is not easy to see is when we serve the extremes in the opposite order to that of the greedy tour. We only do that when we know that $R(G_2(t), t) \geq SR(G_1(t), t)$. This inequality turns out to be sufficient to prove competitiveness in this case also.

2. LEARNING-AUGMENTED ALGORITHMS

Properties of learning-augmented algorithms. As we have mentioned, the desiderata for learning-augmented algorithms are the properties of consistency, smoothness and robustness. In high level, the consistency of an algorithm refers to its provably superior performance under perfect predictions compared to the best algorithm which does not use predictions. On the opposite side, the robustness of an algorithm provides worst case guarantees even for arbitrarily bad predictions. Finally, the smoothness eases out the transition between the two. That is, the performance of a smooth algorithm gracefully deteriorates along with the quality of the predictions. We now formalize these properties. We say that an algorithm is:

1) α*-consistent*, if it is α*-competitive* when there is no error.

2) β*-robust*, if it is β*-competitive* regardless of prediction error.

3) γ*-smooth* for a continuous function $\gamma(err)$, if it is $\gamma(err)$*-competitive*, where err is the prediction error. Note that err could potentially be a tuple of error types.

In general, if c is the best competitive ratio achievable without predictions, it is desirable to have $\alpha < c$, $\beta \leq k \cdot c$ for some constant k and also the function γ should increase from α to β along with the error err. We note that c, α, β and the outputs of γ may be functions of the input and not constant.

Let us now start examining some problems in this area to better understand it.

2.1 Ski Rental

In this section we will start to examine the field of Online Learning-Augmented Algorithms by first focusing on a simple problem known as the Ski Rental Problem. This is arguably one of the most basic and representative rent/buy problems. In these problems, one has to either keep paying a small fee over and over or instead pay a large fee at any time and rid themselves of any fees down the line.

One application of the Ski Rental Problem is in caching, where the system decides if it will read a data block during each pass paying 1 bus cycle, or if it will pass over the block of data and spend many bus cycles to retrieve the data should it be needed later. This is referred to as snoopy caching and is examined in [27] by Karlin et al. Another application is in computer task scheduling, where the system can either serve a task (paying a lot in processing time) or keep it in ”waiting” (thus paying a little for each time unit in resources). This is examined by Seiden in [43]. Finally, the problem finds application in TCP acknowledgment [26], where we must acknowledge the reception of packets not much later than their reception. We can acknowledge multiple packets by sending one package (which is the large fee in this case). We can also delay acknowledgment of a packet in hopes that we will do so for many other packets with the same cost (delaying corresponds to the small fee in this case).

A great deal of generalizations exists for the Ski Rental Problem. Multiple rent options are considered in the Multislope Ski Rental Problem, which has been studied by Fujiwara et al. in [20]. Naturally, the case where multiple buy options exist has also been studied [1].

Alongside the addition of options, generalizations such as the Parking Permit Problem [34] exist. The difference in this problem is that purchases have time-limits regardless of whether they are used or not (much like a parking permit is valid for a certain amount of time). Finally, in [24], the authors consider the problem of deciding which edges of a graph to rent/buy so that sufficient flow between sources and sinks is attained. The simplest case where we have one source and one sink with an edge that connects them (that has rent cost 1 and buy cost b) corresponds to the Ski Rental Problem.

2.1.1 The problem

Imagine that you are planning to go skiing for an unknown number of days. One source of uncertainty could be the weather (you might want to go back home if it gets too cold).

The skiing resort you've chosen offers two options. You may either rent the equipment on a day-to-day basis or pay a set amount to use it for the whole season.

Depending on the number of days you will stay, one of the options will be better than the other. If we normalize the cost of renting for one day to be 1€ then we can assume that the season pass would cost b€. Suppose you would stay for T days in the end.

Then, it is obvious that it is worth it to get the season pass if $b < T$. However, we are intersted in the optimal strategy of an agent that does not know T beforehand. This strategy is called the break-even strategy. For the first $b - 1$ days, we rent the equipment. On day number b, we get the season pass.

It turns out that this strategy describes a (best possible) 2-competitive algorithm (shown in [28]). Indeed, if our algorithm does not get the season pass, then neither does the optimal agent, meaning we have the optimal cost. On the other hand, if we ski for at least b days, the cost of our algorithm is exactly $b - 1 + b = 2b - 1$, while the best strategy would be to get the season pass on the first day with total cost b.

2.1.2 Solving the problem with predictions

As you may have imagined, the only source of difficulty in this problem is that we have no idea about the value of T before deciding. However, you can imagine that an experienced skier can make an educated guess on it.

More abstractly, we could have a hint T' from an oracle that tries to predict the true value T. Then, we are able to design an algorithm that is better than 2-competitive when T' is sufficiently close to T, but not very bad when that difference is large.

This difference is captured in the error η of the predictions which is equal to $|T' - T|$. The smaller this value, the better the performance of an algorithm that uses this prediction is expected to be.

Kumar et al. presented such an algorithm in [40], which has the desired consistency, smoothness and robustness properties. Before giving the algorithm, let λ be a trust value in the range $(0, 1)$. The lower λ is, the more we trust the predictions. Thus, a low λ improves our performance with good predictions but worsens it with bad ones. The algorithm is similar to the break-even algorithm and also pretty simple:

In essence, the algorithm takes the initial break-even limit b and either multiplies it or divides it by λ based on if the predicted days are more than b or not.

If the predicted number of days is less than b, the algorithm gets the season pass earlier than when the break-even algorithm would. On the other hand, if T' is larger than b, it gets

Algorithm 2: Ski Rental with predictions

```
Input  : The prediction $T'$,
         The season pass cost $b$,
         The trust value $\lambda \in (0, 1)$
Output : The day on which we get the season pass

if $T' \geq b$ then
    return $\lceil \lambda b \rceil$;
end
else
    return $\lceil \frac{b}{\lambda} \rceil$;
end
```

the season pass later than the break-even algorithm. How much earlier/later the algorithm gets the season pass is governed by the value of λ.

It turns out that this is algorithm is $(1 + \lambda)$-robust and $(1 + \frac{1}{\lambda})$-consistent. This is derived from the more general result that the algorithm's competitive ratio is at most:

$$1 + min\left(\frac{1}{\lambda}, \lambda + \frac{\eta}{(1-\lambda)|OPT|}\right)$$

2.1.3 Consistency vs. Robustness

For the previous algorithm, a larger λ improves our robustness guarantee, while a smaller λ improves our consistency guarantee. Unfortunately, we can't have both at the same time with this algorithm. But could we possibly have an algorithm that exhibits a better consistency-robustenss relationship than this one? Or, more generally, what is the best such relationship we can hope to achieve? The answer is provided by Alexander Wei et al. in [49].

In more detail, the following theorem is proved in their paper:

Theorem 1. *If $\lambda \in (0, 1)$, any $(1 + \lambda)$-consistent algorithm for ski rental with predictions must be at least $(1 + \frac{1}{\lambda})$-robust.*

2.2 Secretary Problem

In this section we focus our attention to a different problem known as the Secretary Problem. Suggesting its level of abstraction, the same problem is also known as the Marriage Problem, the Sultan's Dowry Problem, the Fussy Suitor Problem, the Googol Game, and the Best Choice Problem. In short, the goal of the problem is to maximize the probability with which we pick the best out of n candidates assuming we can examine them one after another and have to irrevocably decide on rejecting or accepting a candidate immediately after seeing them (without knowing anything about the next candidates we would be presented with).

A possible real-world application of this problem is the (increasingly difficult) task of finding gas in low prices. When driving on the highway, one passes by a certain number of gas stations. Each one has a particular value, which is inversely proportional to the price offered. We may not return to a gas station after passing by it (rejection). Additionally, we stop searching when we've decided to fill up the tank of our car at a specific gas station (acceptance).

Many variations and generalizations of this problem are encountered in the literature. Instead of aiming for the best candidate, Rose et al. [42] consider the goal of trying to pick the candidate whose value is the median across all the values. More generally, the goal of picking the a^{th} best candidate is examined in [45]. An interesting special case of this problem, namely the one where we try to pick the *second*-best candidate is presented in [46] by Vanderbei. It is described as the "Postdoc Problem", suggesting that the best candidate will be accepted to Harvard and go there instead, diminishing our interest in picking them over the second-best candidate who is actually going to accept our offer. The optimal competitive ratio in this case is shown to depend on n and to approach $\frac{1}{4}$ as n tends to infinity, showing that this problem is actually harder than the basic version.

Another direction of tweaking the problem is in increasing the number of tries we have. In this generalization, we are allowed r choices, and we win if any of our choices is the best candidate. One can imagine that our probability of winning is higher in this case and scaling with r. Indeed, in [22], the authors show that with $r = 2$ choices, the optimal solution yields a competitive ratio of $e^{-1} + e^{-\frac{3}{2}}$. They also prove similar results for $r = 3, 4$.

Matsui et al. extend these results in [33] by proving that for any number of choices r, the probability of winning converges to $p_1 + p_2 + ... + p_r$, where each p_i is a limit that involves cutoff thresholds used in an optimal algorithm for the problem (one such threshold is $\frac{n}{e}$ used in the basic version of the problem).

Finally, the case where multiple different roles to fill exist is studied in [29] by Kesselheim et al. Here, each candidate has a different value for each possible role. When hiring a candidate, we also need to specify the role which they will fill. The goal now is to maximize the sum of values obtained across all roles. This is in fact identical to the problem of finding a maximum-weight matching in an edge-weighted bipartite graph where the n nodes of one side arrive in an online fashion. Generalizing the algorithm used in the basic version, a $\frac{1}{e}$-competitive algorithm has been designed for this problem.

2.2.1 The problem

Suppose you work in the HR department of a major company such as Google. On a given day, you are expected to fill an open position, let's say that of a software engineer. In total n interviews have been scheduled for you to assess the competency and "fit" of each candidate. However, you have to decide upon hiring or rejecting a candidate right after the interview, without seeing the following candidates before deciding. Your goal is to maximize the expected quality of the hired candidate.

Note that this is slightly different from the classical version of the problem, where instead we want to maximize the probability with which we pick the *best* candidate. However, the optimal solution is the same for both and in the following we focus on the variant which tries to maximize the expected value.

You would not want to hire a candidate too early, because it is quite possible that another, much better one would show up later. On the flip side, waiting too much on your decision may leave you with only a few options towards the end, possibly much worse than the previous candidates.

The (best possible) solution to this problem is to simply observe the first $\frac{n}{e}$ candidates and note the quality of the best candidate so far. Then, you simply hire the next candidate which beats the memorized candidate (or the final one if no such candidate appears). With this strategy, your hired candidate's expected quality is $\frac{1}{e}$ times that of the best possible candidate.

2.2.2 Formal definition

In the Secretary Problem we have a set $\{1, ..., n\}$ of secretaries, each one associated with a value $u_i \geq 0$. These secretaries are presented to us in a uniformly random order. When a secretary i arrives, we must irrevocably decide to reject or hire them. If we decide to hire them, we automatically reject all subsequent candidates. If we reject them, we just continue with the next secretary.

The goal here is to maximize the expected value of the secretary that we hire. The optimal solution to this is to observe the first $\frac{n}{e}$ secretaries and record the maximum value v_{max} amongst them. After that, we hire the first secretary whose value surpasses v_{max}. If no such secretary arrives up until the last one, we hire the last secretary, since we have to pick one. This strategy yields a $\frac{1}{e}$-approximation of the problem (compared to the offline version in which we can simply pick the best secretary).

2.2.3 Solving the problem with predictions

In a real world scenario, one would suppose that at least one of the applicants to Google will be quite proficient. Therefore, it is quite safe to predict that the maximum value of the candidates would be let's say at least about 8/10. It is obvious that such information would be helpful to the HR employee. In their shoes, you would hardly settle with a candidate whose value is less than 6/10 for example.

Precisely this prediction scheme is proposed by Antoniadis et al. in [6]. In more detail, the prediction consists of a value p^* that attempts to approximate $OPT = max_i\{u_i\}$. Of course, this does not give a suggestion regarding which candidate has this value.

They define the prediction error as:

$$\eta = |p^* - OPT|$$

Additionally, the authors define a parameter λ which attempts to approximate η. This parameter is used by the algorithm. When λ is close to p^*, the algorithm behaves in a similar way to the classical $\frac{1}{e}$-approximation. When λ approaches 0, the algorithm relies more heavily on the prediction. In a sense, λ encapsulates the ”trust” of the algorithm in the predictions, much like in the Ski Rental Problem.

Finally, a parameter $c \geq 1$ is specified which restricts the losses along with the gains of the algorithm. In general, the algorithm always guarantees $\frac{1}{ce}$-competitiveness. We note that in this problem we want to *maximize* the competitiveness value, unlike in the Ski Rental Problem. Setting c close to 1 makes us safe from bad predictions but also mitigates the positive effects of good predictions. As we increase c, our safety net is lowered but also the possible gains are more pronounced. In a sense, the choice of c is like a gamble. Low c means low-risk, low-reward and high c means high-risk, high-reward.

2.2.4 The Lambert *W*-function

Before continuing, we present the Lambert W-function, which is utilized in the algorithm. This function is simply the inverse relation of the function $f(w) = we^w$. For our purposes, we restrict w to the reals. In this case, the equation $ye^y = x$ for $-\frac{1}{e} \leq x < 0$ has two solutions denoted by $W_{-1}(x)$ and $W_0(x)$. We have $W_{-1}(x) \leq W_0(x)$ with the equality holding only if $x = -\frac{1}{e}$.

2.2.5 The algorithm

Algorithm 3: Online Secretary Problem with predictions

```
Input  : The prediction $p^*$,
         The confidence parameter $\lambda$,
         The "gamble" parameter $c$
Output : The hired secretary $a$
$u' \leftarrow 0$;

Phase I:
for $i = 1, ..., exp\{W_{-1}(-\frac{1}{ce})\} \cdot n$ do
    $u' \leftarrow max\{u', u_i\}$;
end

$t \leftarrow max\{u', p^* - \lambda\}$
Phase II:
for $i = exp\{W_{-1}(-\frac{1}{ce})\} \cdot n + 1, ..., exp\{W_0(-\frac{1}{ce})\} \cdot n$ do
    if $u_i > t$ then
        return $a_i$
    end
end

$t \leftarrow max\{u_j \ : \ j \in \{1, ..., exp\{W_0(-\frac{1}{ce})\} \cdot n\}\}$
Phase III:
for $i = exp\{W_0(-\frac{1}{ce})\} \cdot n + 1, ..., n$ do
    if $u_i > t$ then
        return $a_i$
    end
end
```

We can see that the algorithm is comprised of three phases. Phases I and III are pretty much the same as the ones in the classical algorithm. In Phase I, the algorithm simply observes the secretaries and keeps note of the maximum value so far. In Phase III, the algorithm chooses the first secretary to surpass the maximum value observed so far (including the secretaries examined in Phase II).

The real interest is in Phase II. During this phase, a secretary is chosen only if they have a larger value than all of the previous ones and are also at most λ units worse than the predicted best value. We can see that setting λ close to p^* essentially makes this Phase work in the same way as Phase III. A low λ on the other hand sets the bar which secretaries must pass close to the predicted best value.

2.2.6 Performance

The most important question one has to ask themselves when they've designed an algorithm is "How well does it perform?". Here, we merely present the authors' answer to this question, as the technical aspects are not of particular intuitive value (although quite digestible).

This answer is given in the form of two lower bounds for the competitiveness of the algorithm. The first bound holds when $\eta < \lambda$, i.e. we have not underestimated the error. In this case, the competitive ratio of the algorithm for given c, λ and η is at least:

$$max\left\{\frac{1}{ce}, f(c) \cdot max\left\{1 - \frac{\lambda + \eta}{OPT}, 0\right\}\right\}$$

The function $f(c)$ is given in terms of the Lambert W-function, particularly:

$$f(c) = exp\left\{W_0\left(-\frac{1}{ce}\right)\right\} - exp\left\{W_{-1}\left(-\frac{1}{ce}\right)\right\}$$

It should be noted that $f(c)$ vaguely corresponds to the duration of Phase II. Intuitively, the larger this value, the more likely it is for the best secretary to appear during Phase II. Probabilistic analysis (such as in [18] and [31]) then extends this high probability to a higher expected value. Hence, a high $f(c)$ leads to high competitiveness, under the assumption that $\lambda > \eta$.

In the other case, where $\eta \geq \lambda$, the lower bound we have is only $\frac{1}{ce}$.

2.3 Speed Scaling Problem

The final problem we will examine in this chapter is the Online Speed Scaling Problem. The problem we are considering is motivated as in the following scenario. We have a server (computer) that receives requests in an online manner. For each request some we have to perform some sort of computation and, as a measure of Quality-of-Service, we demand that each request is satisfied within some fixed amount of time. In order to satisfy all the requests in time the server may change its processor speed dynamically at any point in time. However, the energy consumption may be a super-linear function of the processing speed (in practice, we assume that the power consumption is s^α where s is the speed of the processor and $\alpha > 1$). Therefore, the problem of minimizing the energy consumption is not trivial. This problem can be examined in the online model where the server does not have any information regarding the future tasks. However, this assumption is unnecessarily restrictive because these requests tend to exhibit some patterns that could be predicted. Thus, a good algorithm should be able to utilize some given predictions about the input.

The real-world applications of this problem are obvious. As power management has become a primary concern in modern data centers, computing resources are being scaled dynamically to minimize energy consumption. Thus, an efficient solution to this problem could yields substantial amounts of energy conservation, benefiting both the environment and the economy.

The original speed scaling problem proposed by Yao et al. in [51] is well understood in both the offline and online setting. In its fully general version, a set of tasks each with different arrival times, workloads, and deadlines has to be carried out in time while the speed of the processor is dynamically adjusted so as to minimize the energy consumption. For the offline setting Yao et al. showed that the problem admits a polynomial time solution through a greedy algorithm. In the online setting, where the jobs are only revealed at their release times, Yao et al. gave two algorithms: (1) the AVERAGE RATE heuristic (AVR), for which they gave a bound of $2^{\alpha-1}\alpha^{\alpha}$ on the competitive ratio. This was later proved to be asymptotically tight by Bansal et al. [11]. (2) The OPTIMAL AVAILABLE heuristic (OA), which was proven to be α^{α}-competitive in [12]. In the same work, Bansal et al. gave a third online algorithm named BKP that exhibits a competitive ratio which is asymptotically equal to e^{α}. Even though these exponential in α competitive ratios might not seem very satisfactory, Bansal et al. also demonstrated that the exponential dependency cannot be lower than e^{α}. A number of versions of the problem have also been examined in the offline setting (precedence constraints, no preemption allowed, nested jobs and more described in a recent survey by Gerards et al. [21]) and also under a stochastic optimization viewpoint (see for example [2]). We should note that, while the problem is interesting in theory in the general case, i.e. when α is part of the input, in practice we usually focus our attention to small values such as 2 or 3 since they model relevant physical laws (described by Bansal et al. in [12]). Even though the BKP algorithm provides the best theoretical asymptotic guarantee, OA or AVR often give better solutions for small α and therefore remain practically relevant.

2.3.1 Solving the problem with predictions

First of all, we describe the problem definition given in [9]. The authors restrict the problem to the uniform case, i.e. the one where the difference between deadlines and release times is the same for every request. This is the Quality-of-Service interval we discussed.

Problem Definition. An instance of the problem is formally described as a triple (w, D, T) where $[0, T]$ is a finite time horizon, each time $i \in 0, ..., T-D$ jobs with a total workload $w_i \in Z_{\geq 0}$ arrive, which have to be completed by time $i + D$. To do so, we can adjust the speed $s_i(t)$ at which each workload w_i is processed for $t \in [i, i + D]$. Jobs may be processed in parallel. The overall speed of our processing unit at time t is the sum $s(t) = \sum_i s_i(t)$, which yields a power consumption of $s(t)^{\alpha}$, where $\alpha > 1$ is a problem specific constant. Since we want to finish each job on time, we require that the amount of work dedicated to job i in the interval $[i, i + D]$ should be w_i. In other words, $\int_i^{i+D} s_i(t)\,dt = w_i$. In the offline setting, the whole instance is known in advance, i.e., the vector of workloads w is entirely accessible. In the online problem, at time i, the algorithm is only aware of all workloads w_j with $j \leq i$, i.e., the jobs that were released before time i. As noted by Bansal et al. [12], in the offline setting the problem can be formulated concisely as the following mathematical program:

Definition 2 (Uniform Speed Scaling Problem). *On input (w, D, T) compute the optimal solution for*

$$min \int_0^T s(t)^{\alpha}\,dt \qquad s.t.\ \forall i \int_i^{i+D} s_i(t)\,dt = w_i, \quad \forall t \sum_i s_i(t) = s(t), \quad \forall i \forall t\ s_i(t) \geq 0.$$

The prediction model and error. Given the previous problem definition, we now present the predicition model used as well as the error measure that comes with it. The prediction consists of a vector w_{pred} of workloads that attempts to approximate the true vector of workloads w_{real}. Then, the error is defined as

$$err(w_{real}, w_{pred}) = ||w_{real} - w_{pred}||_{\alpha}^{\alpha} = \sum_{i} |w_{real}(i) - w_{pred}(i)|^{\alpha}.$$

The algorithm. Now we are ready to present the algorithm used. This algorithm is called LEARNING AUGMENTED SCHEDULING (LAS) and its pseudocode is provided below. But first, it is helpful to look at the theorem which is derived for LAS.

Theorem 2. *For any $\epsilon > 0$, the algorithm LAS has a competitive ratio of at most*

$$min\left\{(1+\epsilon)\, OPT + O\left(\frac{\alpha}{\epsilon}\right)^{\alpha} err,\ O\left(\frac{\alpha}{\epsilon}\right)^{\alpha} OPT\right\}.$$

```
Algorithm 4: LEARNING AUGMENTED SCHEDULING (LAS)
Input  : $T, D,$ and $w_{pred}$ initially and $w_{real}$ in an online fashion.
Output : A feasible schedule $(s_i)_{i=0}^{T-D}$
Let $\delta > 0$ with $\left(\frac{1+\delta}{1-\delta}\right)^{\alpha} = 1 + \epsilon$.

Compute optimal offline schedule for $(w_{pred}, T, (1-\delta)D)$ where the jobs $w_{pred}(i)$ are
  run at uniform speeds $c_i$ and disjoint intervals $[a_i, b_i]$ as described in [51].

on arrival of $w_{real}(i)$ do
    Let $s_i'(t) = \begin{cases} min\left\{\frac{w_{real}(i)}{b_i - a_i}, c_i\right\} & t \in [a_i, b_i], \\ 0, & otherwise. \end{cases}$
    Let $s_i''(t) = \begin{cases} \frac{1}{D} max\left\{0, w_{real}(i) - w_{pred}(i)\right\} & t \in [i, i+D], \\ 0, & otherwise. \end{cases}$
    Let $s_i(t) = \frac{1}{\delta D} \int_{t-\delta D}^{t} s_i'(r) + s_i''(r)\, dr$
end on
```

We will now distill the most important aspects of this algorithm. First of all, note that the algorithm uses a subroutine for the computation of an optimal offline solution based on the predictions. This is very common in learning-augmented algorithms. The same strategy was also used in the Secretary problem. However, the input to to this subroutine is slightly modified (the term D is changed to $(1-\delta)D$). This is so that we can achieve robustness. A convolution of two different schedules is used to achieve this.

One of the operands of the convolution is the part of the workload that was predicted ($s_i'(t)$) while the other takes care of the case where the actual workload is higher than predicted ($s_i''(t)$). Note that if the predictions are perfect, the terms $s_i''(t)$ are always equal to zero and therefore LAS follows exactly the optimal schedule computed in the first lines of the pseudocode. The competitive ratio (consistency in this case) is then only affected by the choice of δ, which also defines the robustness value. Thus, the δ term expresses our trust in the predictions.

Finally, the intution for the smoothness comes from the observation that the terms $s_i''(t)$ (which may possibly affect our competitive ratio) scale along with the values $w_{real}(i) - w_{pred}(i)$, which in turn are related to the error.

Extensions. The authors of [9] also provide a way to utilize an *evolving* prediction model, i.e. one that can update predictions for the requests that have not arrived yet. Additionally, they explain how their results can be generalized to the non-uniform case where the difference between deadlines and release times is not constant.

3. OUR CONTRIBUTIONS

3.1 Introduction

The Traveling Salesman Problem (TSP) is one of the most fundamental and widely studied problems in computer science, both in its offline version [50], where the input is known in advance, and the online version [8] where it arrives sequentially. In this work, we consider the online Traveling Salesman Problem (TSP) on the real line. This version of the problem arises in real-world scenarios such as one dimensional delivery/collection tasks. Such tasks include the operation of elevator systems, robotic screwing/welding, parcel collection from massive storage facilities and cargo collection along shorelines [7, 39]. Furthermore, one can think of other relevant practical settings such as the movement of emergency evacuation vehicles along a perilous highway, where there cannot be knowledge in advance regarding the time and location of persons requiring assistance. However, the great availability of data as well as the improved computer processing power and machine learning algorithms can make it possible for predictions to be made on these locations (e.g combining information from historical data, the weather forecast, etc). In a line of work that started a few years ago [32] and sparked a huge interest [3, 4, 23, 40, 41, 47, 48], it has been demonstrated that such prior knowledge about the input of an online algorithm has the potential to achieve improved performance (i.e competitive ratio) compared to known algorithms (or even lower bounds) that do not use (resp. assume the absence of) any kind of prediction. Therefore, it is natural to consider ways to utilize this information in this problem using a so-called learning-augmented approach.

The input to our online algorithm consists of a set of requests, each associated with a position on the real line as well as a release time. An algorithm for this problem faces the task of controlling an agent that starts at the origin and can move with at most unit speed. The agent may serve a request at any time after it is released. The algorithm's objective is to minimize the makespan, which is the total time spent by the agent before serving all requests. We have two different variants of the problem, depending on whether the agent is required to return to the origin after serving all the requests or not. This requirement exists in the *closed* variant, while it does not in the *open* variant. The makespan in the closed variant is the time it takes the agent to serve the requests *and* return to the origin.

We quantify the performance of an online algorithm by its *competitive ratio*, i.e., the maximum ratio of the algorithm's *cost* to that of an optimal *offline* algorithm OPT, over all possible inputs. We say that an algorithm with a competitive ratio of c is c-competitive. Under this scope, the online TSP on the line has been extensively studied and there have been decisive results regarding lower and upper bounds on the competitive ratio for both variants of the problem. Namely, a tight bound of ≈ 1.64 was given for the closed variant, while the corresponding value for the open variant was proven to be ≈ 2.04 [8, 15].

3.1.1 Our setup

First of all, to define our prediction model and algorithms, it is necessary to know the number of requests n [1]. This setting shows up in various real world scenarios. For example,

[1]Since this (slightly) modifies the original problem definition, the previous competitive ratio results for the classical problem do not necessarily hold for our setup even without predictions. We show in the Appendix that the bound of 1.64 still holds for the closed variant and that a tight bound of 2 holds for the open variant.

in the case of item collection from a horizontal/vertical storage facility, the capacity of the receiving vehicle, which awaits the successful collection of all items in order to deliver them to customers, dictates the number of items to be collected. We note that since n is known, we can assume that each prediction corresponds to a specific request determined by a given labeling, which is shared by both sets (requests and predictions). Under this assumption, we define the $LOCATIONS$ prediction model. In this model, the predictions are estimates for the positions of the requests. The error η increases along with the maximum distance of a predicted location to the actual location of the identically labeled request and is normalized by the length of the smallest interval containing the entire movement of the optimal algorithm. We also define an enhanced prediction model for the open variant named $LOCATIONS + FINAL$ (LF in short) that additionally specifies a request which is predicted to be served last by OPT. In this model, we additionally consider the error metric δ, which increases with the distance of the predicted request to the request actually served last by OPT. We also normalize δ in the same way as η. These models and their respective errors are defined formally in Section 3.2.

3.1.2 Our contributions

Throughout this work, we give upper and lower bounds for our three different settings (closed variant-$LOCATIONS$, open variant-$LOCATIONS$, and open variant-LF). The lower bounds refer to the case of perfect predictions and are established via different $attack$ strategies. That is, we describe the actions of an adversary ADV, who can control only the release times of the requests and has the goal of *maximizing* the competitive ratio of any algorithm ALG. We emphasize that ADV is given the power to observe ALG's actions and act accordingly. In more detail, ADV does not need to specify the release times in advance, but can release a request at time t, taking the actions of ALG until time t into account. This is, in fact, the most powerful kind of adversary. The upper bounds are established via our algorithms and are defined for every value of the error(s). Recall that η and δ refer to the two types of error we consider. Our algorithms and attack strategies are intuitively described in their respective sections. We now present the main ideas and our results.

Closed variant under *LOCATIONS*. We will start by intuitively describing our algorithm for this setting and then continue with our lower bound. We design the algorithm $FARFIRST$. The main idea is that we first focus entirely on serving the requests on the side with the furthest extreme, switching to the other side when all such requests are served. When serving the requests on one side, we prioritize them by order of decreasing amplitude. The intuition is that we have the least possible amount of leftover work for our second departure from the origin, which limits the ways in which an adversary may attack us. We obtain the theorem below. More details are given in Section 3.3.

Theorem 1. *The algorithm $FARFIRST$ is $min\left\{\frac{3(1+\eta)}{2}, 3\right\}$-competitive.*

We emphasize that for $\eta = 0$, this competitive ratio remarkably matches our lower bound of 1.5, making $FARFIRST$ optimal.

Our lower bound for this setting is accomplished via an attack strategy that is analogous to a cunning magician's trick. Suppose that the magician keeps a coin inside one of their hands. They then ask a pedestrian to make a guess for which hand contains the coin. If the pedestrian succeeds, they get to keep the coin. However, the magician can always make it so that the pedestrian fails, for example by having a coin up each of their sleeves and producing the one not chosen by the pedestrian. One can draw an analogy from this trick to our attack strategy, which is described in Section 3.3 in more detail. In this way, we obtain the theorem below.

Theorem 2. *For any $\epsilon > 0$, no algorithm can be $(1.5 - \epsilon)$-competitive for closed online TSP on the line under the $LOCATIONS$ prediction model.*

Open variant under *LOCATIONS*. The algorithm we present for this setting is named $NEARFIRST$. This algorithm first serves the requests on the side opposite to the one $FARFIRST$ would choose. Another divergence from $FARFIRST$ that should be noted is that for the side focused on second, $NEARFIRST$ prioritizes requests that are predicted to be *closer* to the origin, since there is no requirement to return to it, thus avoiding unnecessary backtracking. More details about the algorithm and the proof of the following theorem are given in Section 3.4.1.

Theorem 3. *The algorithm $NEARFIRST$ is $min\,\{f(\eta), 3\}$-competitive, where*

$$f(\eta) = \begin{cases} 1 + \frac{2(1+\eta)}{3-2\eta}, & \textit{for} \quad \eta < \frac{2}{3} \\ 3, & \textit{for} \quad \eta \geq \frac{2}{3} \end{cases} .$$

As in the previous setting, we utilize the "magician's trick" in order to design a similar attack strategy. We describe exactly how this is done in Section 3.4.1. This leads to the establishment of a lower bound, as stated below.

Theorem 4. *For any $\epsilon > 0$, no algorithm can be $\left(1.\overline{44} - \epsilon\right)$-competitive for open online TSP on the line under the $LOCATIONS$ prediction model.*

Open variant under *LOCATIONS+FINAL*. Our algorithmic approach to this setting is again similar to the one implemented in $NEARFIRST$. The difference is that instead of choosing the side with the near extreme first, we choose the side whose extreme is further away from the predicted endpoint of OPT. We name this algorithm $PIVOT$, to emphasize that the prediction for the last request acts as a pivot for the algorithm to decide the first side it will serve. A theorem about $PIVOT$ is presented below, the proof of which has been given in Section 3.4.2.

Theorem 5. *The algorithm $PIVOT$ is $min\,\{f(\eta, \delta), 3\}$-competitive, where*

$$f(\eta, \delta) = \begin{cases} 1 + \frac{1+2(\delta+3\eta)}{3-2(\delta+2\eta)}, & 3 - 2(\delta + 2\eta) > 0 \\ 3, & 3 - 2(\delta + 2\eta) \leq 0 \end{cases} .$$

For this setting, we reuse the attack strategy initially designed for the closed variant. The only difference is that we add another request at the origin with a release time of 4. We explain how we derive the following theorem in Section 3.4.2.

Theorem 6. *For any $\epsilon > 0$, no algorithm can be $(1.25 - \epsilon)$-competitive for open online TSP on the line under the LF prediction model.*

We briefly summarize our results in Table 3.1. Note that the lower and upper bound entries correspond to the no error case. We emphasize that these results are for the case where the number of requests n is known.

Table 3.1: Summary of results.

Setting	Lower bound	Upper bound	Best competitive ratio
Closed variant without predictions	1.64	1.64	1.64
Closed variant under LOCATIONS	1.5	1.5	$min\left\{\frac{3(1+\eta)}{2}, 3\right\}$
Open variant without predictions	2	2	2
Open variant under LOCATIONS	1.44	1.66	$min\left\{1+\frac{2(1+\eta)}{3-2\eta}, 3\right\}$
Open variant under LF	1.25	1.33	$min\left\{1+\frac{1+2(\delta+3\eta)}{3-2(\delta+2\eta)}, 3\right\}$

3.1.3 Related work

Online TSP. The online TSP for a general class of metric spaces has been studied by Ausiello et al. in [8], where the authors show lower bounds of 2 for the open variant and 1.64 for the closed variant. These bounds are actually shown on the real line. Additionally, a 2.5-competitive algorithm and a 2-competitive algorithm are given for the general open and closed variants respectively. A stronger lower bound of 2.04 was shown for the open variant in [15] by Bjelde et al., where both bounds are also matched in the real line. For the restriction of the closed online TSP to the non-negative part of the real line, Blom et al. [16] give a tight 1.5-competitive algorithm. By imposing a fairness restriction on the adversary, they also obtain a 1.28-competitive algorithm. Jaillet and Wagner [25] introduce the ”online TSP with disclosure dates”, where each request may also be communicated to the algorithm before it is released. The authors show improvements to the competitive ratios of previous algorithms as a function of the difference between disclosure and release dates.

Learning-augmented algorithms. Learning-Augmented algorithms have received significant attention since the seminal work of Lykouris and Vassilvitskii [32], where they introduced the online caching problem. Based on that model, Purohit et al. [40] proposed algorithms for the ski-rental problem as well as non-clairvoyant scheduling. Subsequently, Gollapudi and Panigrahi [23], Wang et al. [47], and Angelopoulos et al. [3] improved the initial ski-rental problem. The latter also proposed algorithms with predictions for the list update and bin packing problem and demonstrated how to show lower bounds for algorithms with predictions. Several works, including Rohatgi [41], Antoniadis et al. [4], and Wei [48], improved the initial results regarding the caching problem.

The scheduling problems with machine-learned advice have been extensively studied in the literature. Lattanzi et al. [38] considered the makespan minimization problem with restricted assignments, while Mitzenmacher [36] using predicted job processing times in different scheduling scenarios. Bamas et al. [9], and Antoniadis et al. [5] focused on the online speed scaling problem using predictions for workloads and release times/deadlines, respectively.

There is literature on classical data structures. Examples include the indexing problem, Kraska et al. [30], bloom filters, Mitzenmacher [35]. Further learning-augmented approaches on online selection and matching problems [6, 19] and a more general framework of online primal-dual algorithms [10] also emerged, and there is a survey by Mitzenmacher et al. [37].

Independent work. Compared to the problem considered in this work, a more general one, the online metric TSP, as well as a more restricted version in the half-line, have been studied in [14] under a different setting, concurrently to our work. We note that only the closed variant is considered in [14]. Since the prediction model is different (predictions for the positions as well as release times of the requests are given) and also a different error definition is used, the results are incomparable.

3.2 Preliminaries

The problem definition. In the online TSP on the line, an algorithm controls an agent that can move on the real line with at most unit speed. We have a set $Q = \{q_1, \dots, q_n\}$ of n requests. The algorithm receives the value n as input. Each request q has an associated position and release time. To simplify notation, whenever a numerical value is expected from a request q (for a calculation, finding the minimum of a set, etc.) the term q will refer to the *position* of the request. Whenever we need the release time of a request, we will use $rel(q)$. Additionally, the algorithm receives as input a set $P = \{p_1, \dots, p_n\}$ of predictions regarding the positions of the requests. That is, each p_i attempts to approximate q_i. We assume without loss of generality that Q always contains a request q_0 at the origin with release time 0 and P contains a perfect prediction $p_0 = 0$ for this request[2].

We use t to quantify time. To describe the position of the agent of an algorithm ALG at time $t \geq 0$, we use $pos_{ALG}(t)$. We may omit this subscript when ALG is clear from context. We can assume without loss of generality that $pos(0) = 0$. The speed limitation of the agent is given formally via $|pos(t') - pos(t)| \leq |t' - t|, \forall t, t' \geq 0$. A request q is considered served at time t if $\exists t' : pos(t') = q, rel(q) \leq t' \leq t$, i.e., the agent has moved to the request no earlier than it is released. We will say that a request q is *outstanding* at time t, if ALG has not served it by time t, even if $rel(q) > t$, i.e. q has not been released yet. Let t_{serve} denote the first point in time when all requests have been served by the agent. Also, let $|ALG|$ denote the makespan of an algorithm ALG, for either of the two variants. Then, for the open variant $|ALG| = t_{serve}$ while for the closed one $|ALG| = min\{t : pos(t) = 0, t \geq t_{serve}\}$. For any sensible algorithm, this is equivalent to $t_{serve} + |pos(t_{serve})|$, since the algorithm knows the number of requests and will immediately return to the origin after serving the last one. The objective is to minimize the value $|ALG|$, utilizing the predictions.

Notation. We define $L = min(Q)$ and $R = max(Q)$. Recall that Q contains a request at the origin and thus $L \leq 0$ and $R \geq 0$. We refer to each of these requests as an *extreme* request. If $|L| > |R|$, we define $Far = L, Near = R$. Otherwise, $Far = R, Near = L$. That is, Far is the request with the largest distance from the origin out of all requests. Then, $Near$ is simply the other extreme. We will also refer to the value $|q|$ as the *amplitude* of request q.

We denote with $O(t)$ the set of outstanding requests at time t. Then, $L_O(t) = min(O(t) \cup \{pos(t)\})$ and $R_O(t) = max(O(t) \cup \{pos(t)\})$. We additionally define $L(t) = min(O(t) \cup \{R\})$

[2]This can be seen to be without loss of generality by considering a "handler" algorithm ALG_0 which adds this request/prediction pair to *any* input and copies the actions of any of our algorithms ALG for the modified input. We observe that $|OPT|$ is unchanged and $|ALG_0| = |ALG|$.

and $R(t) = max(O(t) \cup \{L\})$. The difference between $L_O(t), R_O(t)$ and $L(t), R(t)$ is that the former also consider the position of ALG to determine the interval that must be traveled to serve all the requests while the latter assume that ALG is already somewhere inside the interval of outstanding requests (which may not be true due to ALG moving to a bad prediction).

For technical reasons, we have two different notions (and thus terms) for unreleased requests. We will use the same notation for convenience but the terms we introduce will slightly differ for the closed and open variant. For the closed variant, we let $L_{lim} = 0, R_{lim} = 0$ while $L_{lim} = R, R_{lim} = L$ for the open variant. Thus, we define

$$L_U[t] = min(\{q \in Q \ : \ rel(q) \geq t\} \cup \{L_{lim}\}),$$

$$R_U[t] = max(\{q \in Q \ : \ rel(q) \geq t\} \cup \{R_{lim}\}),$$

while also letting

$$L_U(t) = min(\{q \in Q \ : \ rel(q) > t\} \cup \{L_{lim}\}),$$

$$R_U(t) = max(\{q \in Q \ : \ rel(q) > t\} \cup \{R_{lim}\}).$$

The former will be used to prove the upper bounds while the latter will be used to prove the lower bounds.

Recall that we assume without loss of generality that $0 \in P$. We define $L_P = min(P)$ and $R_P = max(P)$. We will say that a prediction p is (un)released/outstanding/served if the associated request q is (un)released/outstanding/served. For a request $q \in Q$ matched with a prediction $p \in P$, we define $\pi(q) = p$ and $\pi^{-1}(p) = q$. That is, the function π takes us from the requests to the associated predictions and π^{-1} takes us from the predictions to the requests.

The *LOCATIONS* prediction model. We now introduce the $LOCATIONS$ prediction model. Let $q_1, ..., q_n$ be a labeling of the requests in Q. The predictions consist of the values $p_1, ..., p_n$, where each p_i attempts to predict the position of q_i.

Error definition for the *LOCATIONS* prediction model. To give an intuition for the metric we will introduce, let us first describe what it means for a prediction to be bad. In any well-posed definition, the further p_i is from q_i, the worse it should be graded. However, we must also take into account the "scale" of the problem, meaning the length of the interval $[L, R]$ that must be traveled by any algorithm, including OPT. The larger this interval, the more lenient our penalty for p_i should be. Therefore, we define the error as

$$\eta[Q, P] = \frac{max_i\{|q_i - p_i|\}}{|L| + |R|}.$$

Additionally, we define $M = \eta \cdot (|L| + |R|)$.

An important lemma for the *LOCATIONS* prediction model. We now present a lemma that gives us some intuition about this prediction model.

Lemma 3. *Let $L_P = min(P), R_P = max(P)$. Then, $|L_P| \geq |R_P|$ implies $|L| \geq |R| - 2M$, and $|R_P| \geq |L_P|$ implies $|R| \geq |L| - 2M$.*

Proof. The following claim constitutes the main part of our proof.

Claim 3.1. $|L_P - L| \leq M$ *and* $|R_P - R| \leq M$.

Proof. If $L_P = \pi(L) \implies |L_P - L| = |\pi(L) - L|$ or $L = \pi^{-1}(L_P) \implies |L_P - L| = |L_P - \pi^{-1}(L_P)|$ then we see that $|L_P - L| \leq M$. Thus, we assume the contrary for the rest of the proof.

Since L_P is by definition the leftmost prediction, we know that $L_P < \pi(L)$. Additionally, since L is the leftmost request, we know that $L < \pi^{-1}(L_P)$.

Let $X \leq Y \leq Z \leq W$ represent the values of the set $\{L, \pi^{-1}(L_P), L_P, \pi(L)\}$ in ascending order. It should be easy to see that X must be equal to either L or L_P. Otherwise, one of $L_P < \pi(L)$ or $L < \pi^{-1}(L_P)$ is violated, leading to a contradiction. We distinguish two cases.

Case 1. $X = L$. In this case, L_P comes after L but before $\pi(L)$ in the X, Y, Z, W ordering. Therefore, $|L - L_P| \leq |L - \pi(L)| \leq M$.

Case 2. $X = L_P$. Similarly, L comes after L_P but before $\pi^{-1}(L_P)$ in the X, Y, Z, W ordering. Thus, $|L_P - L| \leq |L_P - \pi^{-1}(L_P)| \leq M$.

The inequality $|R_P - R| \leq M$ can be seen in a symmetric way. □

Using this claim, we can now conclude the proof of Lemma 3. We focus on the case $|L_P| \geq |R_P|$; the other case is symmetrical. By Claim 3.1, we have

$$|R_P - R| \leq M \implies |R_P| - |R| \leq M \implies |R| \leq |R_P| + M.$$

Additionally, we have

$$|L_P - L| \leq M \implies |L_P| - |L| \leq M \implies |L| \geq |L_P| - M.$$

Combining these inequalities with $|L_P| \geq |R_P|$ proves the lemma. □

Enhanced prediction model for the open variant. Motivated by the performance of our algorithm under the $LOCATIONS$ prediction model, we enhance it with a prediction f' which attempts to guess the label f of a request on which OPT may finish. We name this new model LF (short for $LOCATIONS + FINAL$). The error η is unchanged. We also introduce a new error metric δ. Let $q_{f'}$ be the request associated with the prediction $p_{f'}$. We then choose q_f to be a request on which OPT may finish that minimizes the distance to $q_{f'}$. We then define the new error as

$$\delta[Q, q_f, q_{f'}] = \frac{|q_{f'} - q_f|}{|L| + |R|}.$$

Similarly to before, we define $\Delta = \delta \cdot (|L| + |R|)$.

3.3 Closed Variant

In this section, we consider the closed variant under the $LOCATIONS$ prediction model. We provide the $FARFIRST$ algorithm, which obtains a competitive ratio of 1.5 with perfect predictions and is also smooth and robust. Additionally, we give an attack strategy that implies a lower bound of 1.5 for the competitive ratio of any algorithm in this setting, making $FARFIRST$ optimal.

3.3.1 The *FARFIRST* algorithm.

Before giving the algorithm, we define the $FARFIRST$ ordering on the predictions of an input. For simplicity, we assume that the furthest prediction from the origin is positive. Let $r_1, \ldots, r_a$ be the positive predictions in descending order of amplitude and $l_1, \ldots, l_b$ be the negative predictions ordered in the same way. The $FARFIRST$ ordering is $r_1, \ldots, r_a, l_1, \ldots, l_b$. Any predictions on the origin are placed in the end. Ties are broken via an arbitrary label ordering.

We present the algorithm through an update function used whenever a request is released. This update function returns the plan of moves to be executed until the next release of a request. Note that $ext(side, set)$ returns the extreme element of the input set in the side specified, where $side = true$ means the right side. Also, the $\oplus$ symbol is used to join moves one after another. When all the moves are executed, the agent waits for the next release. This only happens when waiting on a prediction.

Algorithm 5: $FARFIRST$ update function.

```
Input  : Current position $pos$, set $O$ of unserved released requests, first unreleased
         prediction $p$ in $FARFIRST$ ordering or 0 if none exist, the side $farSide$ with
         the furthest prediction from the origin.
Output : A series of (unit speed) moves to carry out until the next request is released.
$posSide \leftarrow (pos > 0)$;
$pSide \leftarrow (p > 0)$;
if $pos = 0$ then $posSide \leftarrow farSide$ ;
if $p = 0$ then $pSide \leftarrow \overline{posSide}$ ;
return $move(ext(posSide, O \cup \{pos\})) \oplus move(ext(pSide, O \cup \{p\})) \oplus move(p)$;
```

In order to give some further intuition on $FARFIRST$, we first give the definition of a *phase*.

Definition 3. *A phase of an algorithm ALG is a time interval $[t_s, t_e]$ such that $pos_{ALG}(t_s) = 0$, $pos_{ALG}(t_e) = 0$ and $pos_{ALG}(t') \neq 0,\ \forall\, t' \in (t_s, t_e)$. That is, ALG starts and ends a phase at the origin and does not cross the origin at any other time during the phase.*

In the following, when we refer to the *far* side, we mean the side with the furthest *prediction* from the origin. The *near* side is the one opposite to that. We see that $FARFIRST$ works in at most three phases. The first phase ends when all predictions on the far side have been released and the agent has managed to return to the origin with no released and outstanding request on the far side. During this phase, any request on the far side is served as long as $FARFIRST$ does not move closer to the origin than the far side's extreme unreleased prediction. Note that some *surprise* requests may appear, i.e., far side requests that were predicted to lie on the near side. These requests are also served in this phase. The second phase lasts while at least one prediction is unreleased. During this phase, the agent serves any request released on the near side, using the predictions as guidance, similarly to the first phase. Requests released on the far side are ignored during this phase. Note that no surprises can occur here, since all far side predictions were released during the first phase. A third phase may exist if some requests were released on the far side during the second phase. These requests' amplitudes are bounded by M, since they were predicted to be positioned on the near side. This simple algorithm is consistent, smooth and robust, as implied by the following theorem.

Theorem 1. *The algorithm $FARFIRST$ is $min\left\{\frac{3(1+\eta)}{2}, 3\right\}$-competitive.*

Let us begin with the intuition behind the proof. The 3-robustness is seen using an absolute worst case scenario in which $FARFIRST$ is $|OPT|$ units away from the origin at time $|OPT|$ (due to the unit speed limitation), and all the requests to serve are on the opposite side. For the consistency and smoothness, we note that $|OPT| \geq 2(|Near| + |Far|)$. It is therefore sufficient to prove that

$$|FARFIRST| - |OPT| \leq |Near| + |Far| + 3\eta \cdot (|Near| + |Far|) = |Near| + |Far| + 3M.$$

We refer to the left hand side as the *delay* of $FARFIRST$. We now see why this bound holds intuitively. We first describe a worst case scenario. In this scenario, OPT first serves the near side completely, and then does the same for the far side, without stopping. Let t_e denote the end time of the first phase. We see that $t_e \leq |OPT| + M$, because $FARFIRST$ follows the fastest possible route serving the requests on the far side, except for a possible delay of M attributable to a misleading prediction. Note that in this worst case, all requests on the near side must have been released by t_e. Therefore, $FARFIRST$ accumulates an extra delay of at most 2 times the maximum amplitude of these requests. By Lemma 3, this value is at most $|Near| + |Far| + 2M$. There are also other possibilities than this worst case, but they also can incur a delay of at most $|Near| + |Far| + 3M$, because $|OPT|$ and $|FARFIRST|$ both increase when such cases occur.

We now give the formal proof of Theorem 1. We will first prove the robustness part of this theorem.

Lemma 4. *The algorithm $FARFIRST$ is 3-robust.*

Proof. Let t_f denote the latest release time for a fixed instance of the problem. We assume w.l.o.g. that $pos_{FARFIRST}(t_f) \leq 0$. Note that after t_f, $FARFIRST$ will move to $L(t_f)$, then to $R(t_f)$ and then back to the origin. Thus, we observe that

$$|FARFIRST| = t_f + |pos(t_f) - L(t_f)| + |L(t_f) - R(t_f)| + |R(t_f)|. \tag{3.1}$$

We distinguish two cases based on the position of $FARFIRST$ at time t_f. **Case 1.** $pos(t_f) \geq L(t_f)$. In this case, we see that

$$(3.1) \implies |FARFIRST| = t_f + pos(t_f) - L(t_f) + R(t_f) - L(t_f) + |R(t_f)| \leq$$

$$t_f + 2(|L(t)| + |R(t_f)|) \leq t_f + 2(|L| + |R|) \leq 2|OPT| \leq 3|OPT|.$$

Case 2. $pos(t_f) < L(t_f)$. Similarly, we have

$$(3.1) \implies |FARFIRST| = t_f + L(t_f) - pos(t_f) + R(t_f) - L(t_f) + |R(t_f)| \leq$$

$$2t_f + 2|R(t)| \leq 2t_f + 2|R| \leq 3|OPT|.$$

□

Now, to prove Theorem 1, it remains to show the consistency/smoothness part, which is given by the following lemma.

Lemma 5. *The algorithm $FARFIRST$ is $f(\eta)$-smooth, where $f(\eta) = \frac{3(1+\eta)}{2}$.*

To prove this lemma, we will bound $FARFIRST$'s *delay*, i.e. the value $|FARFIRST| - |OPT|$, as shown below.

$$|FARFIRST| - |OPT| \leq |Near| + |Far| + 3M \tag{3.2}$$

This is sufficient because Equation (3.2) along with the elementary bound of $|OPT| \geq 2\left(|Near| + |Far|\right)$ prove Lemma 5. Thus, we now state and prove the following claim.

Claim 3.2. *For any input, we have $|FARFIRST| - |OPT| \leq |Near| + |Far| + 3M.$*

We assume w.l.o.g. that $|L_P| \leq |R_P|$. Thus, by Lemma 3 we see that

$$|L| \leq |R| + 2M \implies 2|L| \leq |R| + |L| + 2M =$$

$$|Near| + |Far| + 2M \implies 2|L| + M \leq |Near| + |Far| + 3M.$$

Therefore, it also suffices to show that

$$|FARFIRST| - |OPT| \leq 2|L| + M \tag{3.3}$$

We now describe the way in which we will prove Equation (3.3) or Equation (3.2). Recall the definition of a *phase* given in Definition 3. We note here that we will also use the term *delay* to refer to how much later a phase ends compared to $|OPT|$. We will use another claim stating that for a single phase, $FARFIRST$ will serve the requests on the side of the phase as fast as possible or OPT is seen to finish at most M time units before $FARFIRST$ finishes the phase, thus ”resetting” the delay counter. Using this claim for the (at most) three phases of $FARFIRST$, we can indeed show Claim 3.2. In the following, we will consider a phase in the right side of the origin. We now define a term that is similar to $R_U[t]$.

Let $R_U{}'[t] = max(\{q : q \in Q, rel(q) \geq t, \pi(q) > 0\} \cup \{0\})$. It should be obvious that $R_U{}'[t] \leq R_U[t]$. Note that when $R_U{}'[t] = 0$, this means that all requests associated with positive predictions have been released by time t, thus prompting $FARFIRST$ to conclude the phase.

We should explain here that $R_U[t]$ works as a *block* for OPT (since it has to wait for a request to be released in order to serve it). Similarly $R_U{}'[t]$ works in the same way for $FARFIRST$, which must serve all requests associated with a positive prediction before ending the phase. We observe a useful relationship between these two blocks, which implies that if $FARFIRST$ is blocked on a request to the right of M, then so is OPT. This relationship is encapsulated in the following claim.

Claim 3.3. *If $R_U[t] > M$, then $R_U{}'[t] = R_U[t]$.*

Proof. We see that $\pi(R_U[t]) \geq R_U[t] - M > 0$. Therefore, $\pi(R_U[t])$ is a *positive* prediction and thus $R_U{}'[t] = R_U[t]$. □

The next definition is about the time it would take (after t) for $FARFIRST$ to serve all requests (to the right of $R_U[t]$) and then reach $R_U[t]$. If this is not more than M, we can see that $FARFIRST$ is not too far behind OPT. If it is more than M, we shall see that $FARFIRST$ has enough information to progress through the phase as fast as possible.

$D(t)$ denotes the least amount of time necessary to serve all requests to the right of $R_U[t]$ (assuming they have been released) and then move to $R_U[t]$, starting at position $pos_{FARFIRST}(t)$. This amounts to

$$D(t) = |pos_{FARFIRST}(t) - R_O(t)| + |R_O(t) - R_U[t]|.$$

This function exhibits a useful bound property. If it drops to M or below at some time t, it can only increase above M again due to a request release. This property is described more formally in the following claim. But first, another useful definition is given.

We define $R_P[t]$ as the rightmost positive prediction released at time t or later. If no such prediction exists, then $R_P[t] = 0$. Note that $FARFIRST$ never moves to the left of this prediction. We now give a relevant claim.

Claim 3.4. $|R_P[t] - R_U[t]| \leq M.$

Proof. We can see that $|R_P[t] - R_U{}'[t]| \leq M$ by the definition of these terms. If $R_U[t] > M$, the claim immediately follows by Claim 3.3.

Otherwise, $R_U[t] \leq M$. We have $R_P[t] \geq 0 \implies R_U[t] - R_P[t] \leq M$. Additionally, we know that $R_U{}'[t] \leq R_U[t]$ and $R_P[t] \leq R_U{}'[t] + M \implies R_P[t] - R_U[t] \leq M$, concluding the proof. ☐

Claim 3.5. *Let t_{drop} be a time point such that $D(t_{drop}) \leq M$. If t_{next} is the earliest release time of a request after t_{drop}, then*

$$D(t') \leq M, \, \forall \, t' \in [t_{drop}, t_{next}].$$

Proof. Observe that $R_U[t]$ is constant throughout the interval $[t_{drop}, t_{next}]$. Let R_U denote this constant value. The same is true for $R_P[t]$, which is always equal to a specific value p. We split the interval $[t_{drop}, t_{next}]$ into three parts.

Part 1. This part lasts while $FARFIRST$ is moving towards a released request to the right of $max\{p, pos(t)\}$. This decreases the value $|pos(t) - R_O(t)|$ while $|R_O(t) - R_U|$ is constant and thus $D(t)$ cannot increase.

Part 2. This part lasts while $FARFIRST$ is moving towards p. No released requests exist to the right of $pos(t)$ during this time, since that is taken care of in Part 1. Thus, we have $R_O(t) = max(pos(t), R_U)$. Either way, we see that $D(t) = |R_U - pos(t)|$ during this part. At the start of this part, we have $D(t) \leq M$. When p is reached, we still have $D(t) \leq M$, because p has a distance of at most M to R_U by Claim 3.4. Thus, we have $D(t) \leq M$ throughout this part also.

Part 3. This part lasts while $pos(t) = p$, i.e. $FARFIRST$ is waiting on top of p. It can be seen that $D(t)$ is constant throughout this part and also not larger than M. ☐

We are now ready to present and prove the main claim we discussed.

Claim 3.6. *Assume without loss of generality that $FARFIRST$ is focusing on the right side during a phase. $FARFIRST$ finishes this phase as fast as possible or does so at most M time units after OPT finishes. More precisely, if the phase spans the time interval $[t_s, t_e]$ and the rightmost request served during this phase is R_{phase}, then*

$$(t_e - t_s = 2|R_{phase}|) \vee (t_e - |OPT| \leq M).$$

First of all, note that if at least one request is unreleased at time t_e, then obviously $|OPT| \geq t_e \implies t_e - |OPT| \leq M$. Thus, we can assume in the following that *all* requests will have been released before the end of the phase.

We now draw our attention to a point in time that is very central to our proof.

Let $t_{release}$ be the latest release time of a positive request associated with a positive prediction. Note that $R_U'[t] = 0,\ \forall\, t > t_{release}$. Then, we define

$$t_{chase} = min\{t :\ t_s \leq t \leq t_{release},\ (D(t') > M,\ \forall\, t < t' \leq t_{release})\}.$$

Intuitively, t_{chase} signifies the start of a series of unit speed moves executed by $FARFIRST$ that lead to a *final state* in which $FARFIRST$ has made sufficient progress through the phase and is also not too far behind OPT. After it reaches this state, it is easier to prove Claim 3.6. We now describe what exactly we mean by this state.

Definition 4 (Final state). *We say that $FARFIRST$ has reached a final state in a phase at time t_{state} if $pos(t_{state}) \leq M$ and there are no outstanding requests or unreleased predictions to the right of position M.*

We see why this final state is important in the following claim.

Claim 3.7. *If $FARFIRST$ is in a final state at time t_{state} with $pos(t_{state}) = x_{state} \leq M$, then*

$$(t_e - t_{state} = |x_{state}|) \vee (t_e - |OPT| \leq M).$$

Proof. If $FARFIRST$ moves straight to the origin after t_{state}, the first part is true. On the other hand, there are only two possible ways for $FARFIRST$ *not* to return straight to the origin, both of which provide new lower bounds for $|OPT|$, thus "resetting" the delay. One of them is for $FARFIRST$ to wait for a prediction $p \leq M$ with $\pi^{-1}(p) \leq 0$. Because OPT also has to wait for this request and since $FARFIRST$ will ignore it for this phase, the delay is seen to be at most M after such a case. The other case is for a request q on the right side to be released that was predicted to be on the left side, implying that $q \leq M$. Again, it can take up to $2|q|$ time units for $FARFIRST$ to serve this request and return but also OPT needs to spend at least $|q|$ time units to terminate after it is released. Again, the delay is seen to be at most M. □

Now that our goal has been somewhat clarified, we proceed with the main part of the proof. We now show that after t_{chase}, $FARFIRST$ moves to a final state as soon as possible.

Claim 3.8. *Let $x_{state} = min\{M, R_O(t_{chase})\}$. Then, $FARFIRST$ reaches a final state at time t_{state} and $pos(t_{state}) = x_{state}$, where*

$$t_{state} = t_{chase} + |pos(t_{chase}) - R_O(t_{chase})| + |R_O(t_{chase}) - x_{state}|.$$

Proof. This can be seen by considering the moves followed by $FARFIRST$ after t_{chase}. First of all, we show that $FARFIRST$ moves straight to $R_O(t_{chase})$, starting at t_{chase}. If $pos(t_{chase}) = R_O(t_{chase})$, the claim is obvious. Thus, by the definition of $R_O(t)$, we can assume that $pos(t_{chase}) < R_O(t_{chase})$. It suffices to show that $FARFIRST$ moves to the right until it reaches $R_O(t_{chase})$. We split this move into two possible parts.

Part 1. This part only applies if $pos(t_{chase}) < R_O(t_{chase}) - M$. In this part, we show that $FARFIRST$ moves straight to the point $R_O(t_{chase}) - M$. Indeed, if $R_O(t_{chase})$ is released at some point during this part, then $FARFIRST$ will surely move to $R_O(t_{chase})$ (let alone $R_O(t_{chase}) - M$) in order to serve it. If $R_O(t_{chase})$ is not released during this part, then $R_U[t] = R_O(t_{chase})$ throughout this part. But because $R_U[t] = R_O(t_{chase}) > M$, Claim 3.3 implies that $R_U[t] = R_U'[t] \implies R_P[t] \geq R_U'[t] - M = R_O(t_{chase}) - M$. Therefore, $FARFIRST$ will move to $R_O(t_{chase}) - M$ because of the predictions in this case.

Part 2. This part refers to the move from the point $x = max\{R_O(t_{chase}) - M, pos(t_{chase})\}$ to $R_O(t_{chase})$. In any case (whether Part 1 applies or not), when $pos(t_x) = x$, $R_O(t_{chase})$ *must* have been released. Indeed, assume for the sake of contradiction that $rel(R_O(t_{chase})) > t_x$. Note then that $R_U[t] = R_O(t_{chase})$ until $rel(R_O(t_{chase}))$. We can see that

$$|pos(t) - R_O(t_{chase})| = |pos(t) - R_U[t]| \leq M \,\forall\, t \in [t_x, rel(R_O(t_{chase}))].$$

If $R_U[t] \leq M$ for such t, the claim can be seen by noting that $R_U[t] - M \leq 0$ and that $R_U[t] + M \geq R_U{}'[t] + M$ and because $FARFIRST$ won't exit the interval $[0, R_U{}'[t] + M]$ due to $R_P[t]$.

Otherwise, by Claim 3.3, we have that $R_U[t] = R_U{}'[t]$ for such t. This means that $FARFIRST$ will not move to the left of $R_O(t_{chase}) - M$ due to $R_P[t]$ and also the rightmost point that may be travelled to is $R_O(t_{chase}) + M$, again because of $R_P[t]$. But that would mean that there exists $t' : t_{chase} < t' \leq t_{release}$ with $D(t') \leq M$, a contradiction. Therefore since $R_O(t_{chase})$ is released at t_x, $FARFIRST$ will move towards it immediately.

It now remains to show that $FARFIRST$ will move to x_{state} immediately after reaching $R_O(t_{chase})$. If $x_{state} = R_O(t_{chase})$, the claim is obvious. Therefore, we can assume that $x_{state} = M$ and $x_{state} < R_O(t_{chase})$. We again split this move into two parts.

Part 1. This part lasts while $t \leq t_{release}$ and $x_{state} = M$ has not yet been reached. This means that throughout this part, we have

$$D(t) > M \implies R_U[t] < pos(t) - M \implies R_U{}'[t] + M < pos(t) \implies R_P[t] < pos(t).$$

Thus, since $R_P[t]$ and $R_U[t]$ are always to the left of $pos(t)$, $FARFIRST$ neither stops to wait for a prediction nor backtracks to serve a request during this part.

Part 2. This part starts after Part 1 and lasts until $x_{state} = M$ is reached. Again, $FARFIRST$ trivially does not stop to wait for a prediction, since all the positive ones are released by now. Additionally, we can see that $R_U[t] \leq M$ for this part, since all unreleased predictions are not positive. Thus, $R_U[t] \leq pos(t)$ also holds for this part, prohibiting backtracking.

We can see that in both parts $FARFIRST$ moves to the left with unit speed.

Therefore, two unit speed moves are followed after t_{chase}, one to $R_O(t_{chase})$ and one to $x_{state} = min\{M, R_O(t_{chase})\}$. Also, after these moves, $FARFIRST$ has reached a final state, because no outstanding request or unreleased prediction exists to the right of $x_{state} \leq M$. End of proof. □

We will now use claims 3.7 and 3.8 along with the definition of t_{chase} to prove Claim 3.6.

Proof of Claim 3.6. We distinguish two cases.

Case 1. $t_{chase} = t_s$. In this case, Claim 3.8 implies that $FARFIRST$ reaches a final state by time $t_{state} = t_s + |R_{phase}| + |R_{phase} - x_{state}|$, where $x_{state} = min\{R_{phase}, M\}$. Then, by Claim 3.7, we have

$$(t_e - t_{state} = |x_{state}|) \vee (t_e - |OPT| \leq M) \implies (t_e - t_s = 2|R_{phase}|) \vee (t_e - |OPT| \leq M).$$

Thus, Claim 3.6 holds in this case.

Case 2. $t_{chase} > t_s$. In this case, we first show that $t_{state} \leq |OPT| + max\{M - R_O(t_{chase}), 0\}$, where t_{state} is as described in Claim 3.8. To achieve this, we note that $D(t_{chase}) = M$. Indeed, let t_{prev} be the latest release time before t_{chase}, or t_s if none exist. If $D(t') > M$

for all $t' : t_{prev} < t' \le t_{chase}$, then the definition of t_{chase} is violated. Thus, there exists a $t' : t_{prev} < t' \le t_{chase}$ such that $D(t') \le M$ and we have $D(t_{chase}) \le M$ by Claim 3.5. We now distinguish two subcases.

Case 2.1. $M \le R_O(t_{chase}) \implies x_{state} = M$. In this case, we see that

$$D(t_{chase}) \le M \implies |pos(t_{chase}) - R_O(t_{chase})| + |R_O(t_{chase}) - R_U[t_{chase}]| \le M \implies$$
$$R_U[t_{chase}] \ge |pos(t_{chase}) - R_O(t_{chase})| + |R_O(t_{chase}) - M| \implies$$
$$t_{chase} + R_U[t_{chase}] \ge t_{chase} + |pos(t_{chase}) - R_O(t_{chase})| + |R_O(t_{chase}) - M| \implies$$
$$|OPT| \ge t_{state}.$$

Case 2.2. $M > R_O(t_{chase}) \implies x_{state} = R_O(t_{chase})$. Because $D(t_{chase}) \le M$, we must have

$$|pos(t_{chase}) - R_O(t_{chase})| \le M - R_O(t_{chase}) + R_U[t_{chase}] \implies$$
$$t_{state} \le |OPT| + M - R_O(t_{chase}).$$

In both of these subcases, by Claim 3.7 we have

$$(t_e - t_{state} = |x_{state}|) \vee (t_e - |OPT| \le M) \implies (t_e = |x_{state}| + t_{state}) \vee (t_e - |OPT| \le M) \implies$$
$$(t_e \le |OPT| + M) \vee (t_e - |OPT| \le M).$$

Claim 3.6 is now proved in all cases. □

We can finally use Claim 3.6 to prove Claim 3.2 by showing that Equation (3.3) or Equation (3.2) holds.

Proof of Claim 3.2. Let $t_s(1), t_e(1)$ be the start and end times of the first phase of $FARFIRST$ and $t_s(2), t_e(2)$ are similarly defined for the second phase. We can see that $t_s(1) = 0$ and $t_s(2) = t_e(1)$. We distinguish two cases based on the possible existence of a third phase.

Case 1. No requests are released on the right side after $t_e(1)$. Thus, we see that $|FARFIRST| = t_e(2)$. By Claim 3.6, we see that $t_e(1) \le |OPT| + M$. Using Claim 3.6 for the second phase also, we see that

$$(t_e(2) - t_s(2) = 2|L|) \vee (t_e(2) - |OPT| \le M) \implies$$
$$(t_e(2) - t_e(1) = 2|L|) \vee (t_e(2) - |OPT| \le M) \implies \text{(3.3)} \implies \text{(3.2)}.$$

Case 2. At least one request is released on the right side after $t_e(1)$. Let q_M be the rightmost such request. We can see that $q_M \le M$, since it is necessarily associated with a non-positive prediction. We can see that $|FARFIRST| = t_e(2) + 2|q_M|$. We also know that $|OPT| \ge rel(q_M) + |q_M| \ge t_e(1) + |q_M|$. By Claim 3.6 for the second phase, we have

$$(t_e(2) - t_s(2) = 2|L|) \vee (t_e(2) - |OPT| \le M) \implies$$
$$(t_e(2) - t_e(1) = 2|L|) \vee (t_e(2) - |OPT| \le M) \implies$$
$$(t_e(2) + 2|q_M| = 2|L| + t_e(1) + 2|q_M|) \vee (t_e(2) + 2|q_M| \le |OPT| + M + 2|q_M|) \implies$$
$$(|FARFIRST| \le |OPT| + 2|L| + |q_M|) \vee (|FARFIRST| \le |OPT| + M + 2|q_M|) \implies$$
$$(|FARFIRST| \le |OPT| + 2|L| + M) \vee (|FARFIRST| \le |OPT| + 3M) \implies$$
$$\text{(3.3)} \vee \text{(3.2)} \implies \text{(3.2)}.$$

□

We can now use Claim 3.2 to prove Lemma 5.

Proof of Lemma 5. We know that $|OPT| \geq 2(|L|+|R|)$, since it must at least travel to both L and R and back. Also, by Claim 3.2, we have $|FARFIRST| - |OPT| \leq |Near| + |Far| + 3M = |L| + |R| + 3M$. These inequalities imply

$$\frac{|FARFIRST|}{|OPT|} = 1 + \frac{|FARFIRST| - |OPT|}{|OPT|} \leq 1 + \frac{|L| + |R| + 3M}{2(|L| + |R|)} = \frac{3(1+\eta)}{2}.$$

□

We can finally prove Theorem 1.

Proof of Theorem 1. By Lemma 4, $FARFIRST$ is 3-robust. Additionally, by Lemma 5, $FARFIRST$ is $\left(\frac{3(1+\eta)}{2}\right)$-smooth. Thus, the theorem holds. □

3.3.2 A 1.5-attack.

Based on the magician analogy presented in Section 3.1.2, we design an attack strategy that yields the following theorem.

Theorem 2. *For any $\epsilon > 0$, no algorithm can be $(1.5 - \epsilon)$-competitive for closed online TSP on the line under the $LOCATIONS$ prediction model.*

We first describe in high level the main ideas in the proof of this theorem. In our attack strategy, we have arbitrarily many requests evenly placed in the interval $[-1, 1]$. The more of these requests we have, the closer the competitive ratio we achieve will be to 1.5.

For a given set Q_X of request *positions*, we now describe how the release times of the requests at these positions are decided. The strategy is that we release requests on both sides as long as ALG has not yet approached a released request. This is the first phase of releases and it is structured in such a way that OPT could begin serving either of the two sides as fast as possible. In the magician analogy we described, this corresponds to the time before the pedestrian chooses a hand.

When ALG approaches a released request, we ”freeze” the requests on ALG's side. That is, if ALG moves close to a released request on the left side, say one placed at $-\frac{1}{2}$, all requests in the interval $[-\frac{1}{2}, 0]$ have their release time delayed such that OPT can still serve the entire right side and then come back to serve the left side by $t = 4$. This corresponds to the magician producing the coin on the right hand while the pedestrian has chosen the left hand. However, ALG is now faced with a dilemma. Should it wait for these ”frozen” requests or should it travel all the way to 1 in order to serve the right side first? We will see that both options are bad, in the sense that $|ALG|$ can be seen to be arbitrarily close to 6. We now proceed with the formal proof.

We describe a family F_C of inputs that is structured as follows. For a given rank $n \geq 2$, we place exactly n requests evenly spaced across the interval $[-1, 1]$. For an instance f of the family F_C, $\alpha(f)$ is defined as the distance between any consecutive pair of requests in f.

Claim 3.9. *If an instance f of the family F_C has rank n, then $\alpha(f) = \frac{2}{n-1}$.*

Proof. There are n requests that delimit an interval of length 2. Thus, there are $n-1$ equal subintervals, whose lengths' sum is equal to 2. Therefore, each subinterval has length $\frac{2}{n-1}$. □

All that is left to determine is the release times of the requests. We split the release times into two "phases". The first phase takes place for as long as $L_U(t) < pos_{ALG}(t) < R_U(t)$. During this phase, ADV releases any request with distance d from the origin at time $2-d$. Note that this release method allows OPT to eagerly start serving any side of the origin first without waiting for requests to release.

Now for the second phase's releases, assuming that ALG exits the interval to its left side (i.e. commits to the left extreme), the requests to the right side are released as during the first phase. However, for any unreleased request to the left side with distance d from the origin, its release time is delayed to $4-d$. If ALG exits from the right instead, ADV releases the left requests as in the first phase and delays the right requests. The input (positions and release times of requests) is now fully specified. For the following, we will define t_{commit} as the start time of the second phase. That is,

$$t_{commit} = min(\{t \,:\, \neg(L_U(t) < pos_{ALG}(t) < R_U(t))\}).$$

We immediately observe the following inequality, which guarantees that the second phase of the requests starts in a timely manner.

Claim 3.10. $1 \leq t_{commit} \leq 2$.

Proof. For the sake of contradiction, assume that $t_{commit} < 1$. Then, $|pos(t_{commit})| \geq 1$ since $[L_U(t), R_U(t)] = [-1, 1]$ for $t < 1$. However, because any algorithm is limited to unit speed, $t_{commit} < 1 \implies |pos(t_{commit})| < 1$, a contradiction.

On the other hand, assume that $t_{commit} > 2$. This means that $L_U(2) < pos(2) < R_U(2)$. But, since the first phase has not stopped until $t = 2$, we have $L_U(2) \geq 0, R_U(2) \leq 0$, which clearly leads to a contradiction. □

We now state a lemma ensuring that OPT finishes in the absolute least time possible for any such input. This allows us to maximize the competitive ratio we achieve against ALG.

Lemma 6. *For any instance f in the family F_C, $|OPT| = 4$.*

Proof. We observe that the requests of one side (the one ALG did not exit from) are released such that OPT can serve them all and return to the origin by $t = 2$. Additionally, the other side's requests are released such that OPT never has to stop for them either, i.e. it can serve them all and return to the origin by $t = 4$. Thus, $|OPT| = 4$. □

However, ALG has commited to one side (by exiting the interval) and we will prove that it requires at least $6 - 2\alpha(f)$ time units to terminate. This will be our main lemma. We state it here for reference but will prove it later.

Lemma 7. *For any instance f in the family F_C and for any ALG, we have that $|ALG| \geq 6 - 2\alpha(f)$.*

Before proving this lemma, we give some more claims. For the following, we assume without loss of generality that ALG exits the interval $[L_U(t), R_U(t)]$ from the left, i.e. it commits to the left side.

Claim 3.11. *For any instance f in the family F_C,*

$$L_U(t_{commit}) - \alpha(f) \le pos_{ALG}(t_{commit}) \le L_U(t_{commit}).$$

Proof. The claim can be seen by examining two cases. If ALG exited the unreleased requests interval itself by moving out of it, then $pos_{ALG}(t_{commit}) = L_U(t_{commit})$. In the other case, ALG was forced out of the interval by a request release. Thus, right before this release (which occurs at precisely t_{commit}), ALG was inside the interval. The previous interval was $[L_U(t_{commit}) - \alpha(f), R_U(t_{commit}) + \alpha(f)]$. Thus, the inequality holds. □

We now draw attention to one particular value, which constitutes the backbone of our attack. We define $d_{commit} = |L_U(t_{commit})|$, where t_{commit} is the start of the second phase of releases.

We now show some claims that allow us to use this value to get a lower bound for $|ALG|$.

Claim 3.12. *For any instance f in the family F_C, $t_{commit} \ge 2 - d_{commit} - \alpha(f)$.*

Proof. If $L_U(t_{commit}) = -1 \implies d_{commit} = 1$, then by Claim 3.10, we have $t_{commit} \ge 1 \ge 2 - d_{commit} - \alpha(f)$. Therefore, we can assume that a request L_{prev} exists with $L_{prev} = L_U(t_{commit}) - \alpha(f) < L_U(t_{commit})$. We see that if $t < 2 - d_{commit} - \alpha(f)$, then $L_U(t) \le L_{prev}$, because L_{prev} is unreleased until $2 - d_{commit} - \alpha(f)$. Therefore, $t_{commit} \ge 2 - d_{commit} - \alpha(f)$, since otherwise we would have $L_U(t_{commit}) \le L_{prev}$, a contradiction. □

The following claim states that ALG has essentially made no progress until t_{commit}. If t_{commit} is close to 2, we can easily see why this is bad for ALG. On the other hand, an early commit means that d_{commit} will be large (due to Claim 3.12), posing problems again for ALG.

Claim 3.13. *ALG has not served any request during the first phase, i.e. up to time t_{commit}.*

Proof. This is due to the fact that ALG has not exited the interval of unreleased requests until t_{commit}. Therefore, it cannot have moved to a released request. Since ALG has to move to a request to serve it, the claim holds. □

Now we are ready to prove Lemma 7.

Proof of Lemma 7. Let us examine the options that ALG has in order to terminate after t_{commit}. By Claim 3.13, we know that ALG has not yet served the requests at $-1, L_U(t_{commit}), 1$. We examine cases based on the order in which it chooses to do so from t_{commit} on.

Case 1. ALG serves 1 before -1. Then, ALG at the very least needs to travel from $pos_{ALG}(t_{commit})$ to 1, then to -1 and then back to the origin. Using Claims 3.12 and 3.11, this takes at least

$$|ALG| \ge t_{commit} + |pos_{ALG}(t_{commit}) - 1| + |1 - (-1)| + |-1 - 0| \ge$$

$$(2 - d_{commit} - \alpha(f)) + (d_{commit} + 1) + 2 + 1 = 6 - \alpha(f).$$

Case 2. ALG serves -1, then 1 and then $L_U(t_{commit})$. Again using Claims 3.12 and 3.11, this takes

$$|ALG| \ge t_{commit} + |pos_{ALG}(t_{commit}) - (-1)| + |(-1) - 1| + |1 - L_U(t_{commit})| + |L_U(t_{commit}) - 0| \ge$$

$$(2 - d_{commit} - \alpha(f)) + (1 - d_{commit} - \alpha(f)) + 2 + (1 + d_{commit}) + d_{commit} = 6 - 2\alpha(f).$$

Case 3. ALG serves $L_U(t_{commit})$ before 1. In this case, ALG has to first wait for $L_U(t_{commit})$ to be released and then go to serve 1. Because $L_U(t_{commit})$ is a request to the left of the origin released during the second phase, we have

$$|ALG| \geq rel(L_U(t_{commit})) + |L_U(t_{commit}) - 1| + |1 - 0| \geq (4 - d_{commit}) + (1 + d_{commit}) + 1 = 6.$$

These cases are exhaustive and thus Lemma 7 is proved. □

With all the above, we can finally prove Theorem 2.

Proof of Theorem 2. By Lemma 7 and Lemma 6, we have a competitive ratio of at least $\frac{6-2\alpha(f)}{4}$ for any algorithm ALG. By Claim 3.9, we can see that $\alpha(f)$ can be arbitrarily small and thus this competitive ratio can be arbitrarily close to 1.5, proving our claim. □

3.4 Open Variant

In this section, we consider the open variant. We have two prediction models for this variant. The first one is the $LOCATIONS$ prediction model and the second is the enhanced $LOCATIONS + FINAL$ model (LF in short). For both settings, we give algorithms and lower bounds.

3.4.1 The *LOCATIONS* prediction model

Under the $LOCATIONS$ prediction model, we design the $NEARFIRST$ algorithm, which achieves a competitive ratio of $1.\overline{66}$ with perfect predictions and is also smooth and robust. We complement this result with a lower bound of $1.\overline{44}$ using a similar attack strategy to the one used for the closed variant.

The $NEARFIRST$ algorithm. As we mentioned in the introduction, $NEARFIRST$ is similar to $FARFIRST$ and actually slightly simpler. In essence, $NEARFIRST$ simply picks a direction in which it will serve the requests. Then, it just serves the requests either from left to right or from right to left, using the predictions as guidance. The pseudocode for $NEARFIRST$ is given below. Recall that $move(x) \oplus move(y)$ is used to indicate a move to x followed by a move to y.
We present the following theorem regarding the competitive ratio of $NEARFIRST$.

Theorem 3. *The algorithm $NEARFIRST$ is $min\{f(\eta), 3\}$-competitive, where*

$$f(\eta) = \begin{cases} 1 + \frac{2(1+\eta)}{3-2\eta}, & \textit{for} \quad \eta < \frac{2}{3} \\ 3, & \textit{for} \quad \eta \geq \frac{2}{3} \end{cases}.$$

We first describe the main ideas used in the proof of this theorem. As in the case of $FARFIRST$, the 3-robustness holds because at time $|OPT|$, $NEARFIRST$ has "leftover work" of at most $2|OPT|$ time units (to return to the origin and then copy OPT). For the consistency/smoothness, we draw our attention to the request q_f served last by OPT. For the following, we assume that $NEARFIRST$ serves the requests left to right. Let

Algorithm 6: $NEARFIRST$ update function.

```
Input   : Current position $pos$, set $O$ of unserved released requests, set $P$ of
          predictions.
Output  : A series of (unit speed) moves to carry out until the next request is released.
$P' \leftarrow$ the unreleased predictions in $P$;
if $P'$ is empty then
    if $pos < \frac{max(O)+min(O)}{2}$ then return $move(min(O)) \oplus move(max(O))$ ;
    else return $move(max(O)) \oplus move(min(O))$ ;
end
if $|min(P)| < |max(P)|$ then return $move(min(P' \cup O)) \oplus move(min(P'))$ ;
else return $move(max(P' \cup O)) \oplus move(max(P'))$ ;
```

$d = |q_f - R|$. We will show that the delay of $NEARFIRST$ is bounded by $M + d$. Let t_{q_f} be the time when $NEARFIRST$ has served all requests to the left of q_f, including q_f. It turns out that $t_{q_f} \leq |OPT| + M$, because $NEARFIRST$ serves this subset of requests as fast as possible, except for a possible delay of M due to a misleading prediction. Then, in this worst case, $NEARFIRST$ accumulates an extra delay of at most d, proving our claim.

Finally, we bound OPT from below as a function of d. We see that OPT can either serve the requests L, R, q_f in the order L, R, q_f or in the order R, L, q_f. The worst case is the latter, where we see that $|OPT| \geq 2|R| + |L| + (|L| + |R| - d) = 3|R| + 2|L| - d$. Since $d \leq |L| + |R|$, we obtain

$$\frac{|NEARFIRST|}{|OPT|} = 1 + \frac{|NEARFIRST| - |OPT|}{|OPT|} \leq 1 + \frac{M + |L| + |R|}{2|R| + |L|}.$$

Because $NEARFIRST$ considers L the near extreme due to the predictions, by Lemma 3 we find that $|R| \geq \frac{1-2\eta}{2}(|L| + |R|)$, which in turn proves our bound.

We now give the formal proof of Theorem 3. First of all, we present two lemmas that are very important. Their proofs are deferred to the Appendix, since they also refer to the $PIVOT$ algorithm, which is introduced later.

Lemma 8. *Let ALG be either $NEARFIRST$ or $PIVOT$. Then, ALG is 3-robust.*

Lemma 9. *Let ALG be either $NEARFIRST$ or $PIVOT$. Also, let q_f be the request served last by OPT. Assume without loss of generality that ALG serves requests from left to right. Let $d = |q_f - R|$. Then, we have $|ALG| - |OPT| \leq M + d$.*

The robustness part of Theorem 3 is implied by Lemma 8.

Now, to prove Theorem 3, it remains to show the consistency/smoothness part, which is given by the following lemma.

Lemma 10. *The algorithm $NEARFIRST$ is $f(\eta)$-smooth for $\eta < \frac{2}{3}$, where $f(\eta) = 1 + \frac{2(1+\eta)}{3-2\eta}$.*

Proof. Assume w.l.o.g. that $NEARFIRST$ serves the left extreme first. By Lemma 9, we see that $|NEARFIRST| - |OPT| \leq M + d$, where d is the distance of R to the request q_f

served last by OPT. We distinguish two cases based on the order in which OPT serves the requests in $\{L, R, q_f\}$. **Case 1.** OPT serves L, R and then q_f. It can be seen then that

$$\frac{|NEARFIRST|}{|OPT|} \le 1 + \frac{M+d}{2|L|+|R|+d}.$$

The derivative of the right hand side with respect to d is

$$\frac{|R|+2|L|-M}{(d+|R|+2|L|)^2}.$$

Because $\eta < \frac{2}{3}$, it must hold that $M < \frac{2}{3}(|L|+|R|) \implies \dfrac{|R|+2|L|-M}{(d+|R|+2|L|)^2} > 0$. Thus, we may maximize d to get an upper bound that is valid for any value of d. Because $d \le |L|+|R|$ we see that

$$\frac{|NEARFIRST|}{|OPT|} \le 1 + \frac{M+|L|+|R|}{3|L|+2|R|}.$$

Case 2. OPT serves R, L and then q_f. It can be seen then that

$$\frac{|NEARFIRST|}{|OPT|} \le 1 + \frac{M+d}{2|R|+|L|+(|L|+|R|-d)} = 1 + \frac{M+d}{3|R|+2|L|-d}.$$

We can see that the right hand side is an increasing function of d. Thus, we may again maximize d to get an upper bound.

$$\frac{|NEARFIRST|}{|OPT|} \le 1 + \frac{M+|L|+|R|}{2|R|+|L|}. \tag{3.4}$$

In both cases, the bound of Equation (3.4) is valid. Noting that $|L_P| \le |R_P|$ (because $NEARFIRST$ chose to go to the left first), we now use Lemma 3 to finalize our proof.

$$|L_P| \le |R_P| \implies |R| \ge |L| - 2M \implies 2|R| \ge |L|+|R|-2M =$$

$$(|L|+|R|)(1-2\eta) \implies |R| \ge \frac{1-2\eta}{2}(|L|+|R|) \implies$$

$$1 + \frac{M+|L|+|R|}{2|R|+|L|} \le 1 + \frac{(1+\eta)(|L|+|R|)}{(1+\frac{1-2\eta}{2})(|L|+|R|)} = 1 + \frac{2(1+\eta)}{3-2\eta} \implies$$

$$\frac{|NEARFIRST|}{|OPT|} \le 1 + \frac{2(1+\eta)}{3-2\eta}.$$

□

We now give the proof of Theorem 3.

Proof of Theorem 3. By Lemma 8, $NEARFIRST$ is 3-robust. Additionally, by Lemma 10, $NEARFIRST$ is $\left(1+\frac{2(1+\eta)}{3-2\eta}\right)$-smooth. Thus, Theorem 3 holds. □

A $1.\overline{44}$-attack. For this setting, we use an attack strategy that is very similar to the one introduced in Section 3.3. This allows us to obtain the following theorem.

Theorem 4. *For any $\epsilon > 0$, no algorithm can be $\left(1.\overline{44} - \epsilon\right)$-competitive for open online TSP on the line under the $LOCATIONS$ prediction model.*

The logic behind the attack we give here is exactly the same as the one used to prove Theorem 2. There are two main differences demanded by the nature of the open variant.

One is that the first phase is shorter in this attack. Instead of stopping when ALG exits $[L_U(t), R_U(t)]$, the phase now stops when ALG exits the interval $[3L_U(t) + 2, 3R_U(t) - 2]$. This is so that both options of ALG (switch to the other side or wait for the ”frozen” requests) are equally hurtful.

The other difference lies in the release times of the second phase. Each request on the side chosen by ALG now has its release time delayed to $2 + d$ (instead of $4 - d$), where d is the request's distance from the origin. This is so OPT can finish by $t = 3$, which is the fastest possible even if all requests are released immediately.

In the following, we assume without loss of generality that ALG exits the interval $[3L_U(t) + 2, 3R_U(t) - 2]$ from the left side. We define the family F_O of inputs just like we defined F_C in the proof of Theorem 2. Of course, the release times are different as explained in the previous paragraphs.

An immediate observation that we have already mentioned is the following lemma.

Lemma 11. *For any instance f in the family F_O, $|OPT| = 3$.*

Proof. Since ALG exits the interval from the left side, each request q_r on the right side is released at time $2 - |q_r|$. Thus, by moving to 1 and back, OPT serves all the requests on the right side. Additionally, each request q_l on the left side is released no later than $2 + |q_l|$, allowing OPT to serve these requests by just moving to -1 after reaching the origin at $t = 2$. Therefore, OPT can serve all the requests by time 3, i.e. $|OPT| = 3$. □

The next piece of the puzzle is a lower bound on ALG. This is given by the following lemma.

Lemma 12. *For any instance f in the family F_O and any algorithm ALG, $|ALG| \geq \frac{13}{3} - 3\alpha(f)$, where $\alpha(f)$ is the distance between consecutive requests in f.*

To prove this lemma, we will use some claims, many of which are very similar to claims used for Lemma 7. To present these claims, we introduce two important terms.

We denote with t_{commit} the start time of the second phase, i.e.

$$t_{commit} = min(\{t \ : \ \neg(3L_U(t) + 2 < pos_{ALG}(t) < 3R_U(t) - 2)\}).$$

Additionally, we draw attention to the value $d_{commit} = |L_U(t_{commit})|$, which is very important for the attack.

We now present the claims which are very similar to those used for the closed variant. Claims 3.14, 3.15 and 3.16 can be seen in the same way as Claims 3.10, 3.11 and 3.12, respectively.

Claim 3.14. $1 \leq t_{commit} \leq 1 + \frac{1}{3}$.

Claim 3.15. $pos(t_{commit}) \le 3L_U(t_{commit}) + 2$.

Claim 3.16. *For any instance f in the family F_O, $t_{commit} \ge 2 - d_{commit} - \alpha(f)$.*

Moreover, we also have the following claim.

Claim 3.17. *For any instance f of the family F_O, $d_{commit} \ge \frac{2}{3} - \alpha(f)$.*

Proof. By Claim 3.16, we have $d_{commit} \ge 2 - t_{commit} - \alpha(f)$. Additionally, Claim 3.14 implies that $t_{commit} \le \frac{4}{3}$. The claim follows. □

We are now ready to prove Lemma 12.

Proof of Lemma 12. We distinguish cases based on the order in which ALG chooses to serve $-1, 1, L_U(t_{commit})$ after t_{commit}. **Case 1.** ALG serves 1 before -1. By Claims 3.16, 3.15 and 3.17, this takes at least

$$|ALG| \ge t_{commit} + |pos(t_{commit})| + 2 + 1 \ge$$
$$2 - d_{commit} - \alpha(f) + |3L_U(t_{commit}) + 2| + 2 + 1 =$$
$$2 - d_{commit} - \alpha(f) + 3d_{commit} - 2 + 2 + 1 = 3 + 2d_{commit} - \alpha(f) \ge$$
$$3 + \frac{4}{3} - 3\alpha(f) \ge \frac{13}{3} - 3\alpha(f).$$

Case 2. ALG serves $L_U(t_{commit})$ before 1. By the definition of the second phase's release times and Claim 3.17, we have

$$|ALG| \ge rel(L_U(t_{commit})) + |L_U(t_{commit})| + 1 = 2 + d_{commit} + d_{commit} + 1$$
$$3 + 2d_{commit} \ge \frac{13}{3} - 2\alpha(f).$$

Case 3. ALG serves in the order -1, 1, $L_U(t_{commit})$. By Claim 3.17, we easily obtain

$$|ALG| \ge 2 + 2 + \frac{2}{3} - \alpha(f) \ge \frac{13}{3} - \alpha(f).$$

These cases are exhaustive and thus Lemma 12 follows. □

We can now use Lemmas 11 and 12 to prove Theorem 4.

Proof of Theorem 4. By Lemma 11, we have $|OPT| = 3$. On the other hand, by Lemma 12, we see that $|ALG| \ge \frac{13}{3} - 3\alpha(f)$. Thus, we obtain a competitive ratio of at least $\frac{\frac{13}{3} - 3\alpha(f)}{3}$. For arbitrarily small $\alpha(f)$, this value can be arbitrarily close to $1.\overline{44}$, proving the Theorem. □

3.4.2 The *LOCATIONS+FINAL* prediction model

In our final setting we consider the open variant under the LF prediction model. We give the $PIVOT$ algorithm, which is $1.\overline{33}$-competitive with perfect predictions and is also smooth and robust. We also reuse the attack strategy described for the closed variant to achieve a lower bound of 1.25.

The $PIVOT$ algorithm. The final algorithm we present works in the same way as $NEARFIRST$, except for the order in which it focuses on the two sides of the origin. Instead of heading to the near extreme first, $PIVOT$ prioritizes the side whose extreme is further away from the predicted endpoint of OPT, which is provided by the LF prediction model. The pseudo-code for $PIVOT$ is given below. Note that $P_{f'}$ refers to the element in P with label f'.

Algorithm 7: $PIVOT$ update function.

```
Input  : Current position $pos$, set $O$ of unserved released requests, set $P$ of
         predictions, label $f'$ of $OPT$'s predicted endpoint.
Output : A series of (unit speed) moves to carry out until the next request is released.
$P' \leftarrow$ the unreleased predictions in $P$;
if $P'$ is empty then
    if $pos < \frac{max(O)+min(O)}{2}$ then return $move(min(O)) \oplus move(max(O))$ ;
    else return $move(max(O)) \oplus move(min(O))$ ;
end
if $P_{f'} > \frac{max(P)+min(P)}{2}$ then return $move(min(P' \cup O)) \oplus move(min(P'))$ ;
else return $move(max(P' \cup O)) \oplus move(max(P'))$ ;
```

As for the previous algorithms, we show a theorem that pertains to $PIVOT$'s competitive ratio for different values of the η and δ errors.

Theorem 5. *The algorithm $PIVOT$ is $min\{f(\eta,\delta),3\}$-competitive, where*

$$f(\eta,\delta) = \begin{cases} 1 + \frac{1+2(\delta+3\eta)}{3-2(\delta+2\eta)}, & 3-2(\delta+2\eta) > 0 \\ 3, & 3-2(\delta+2\eta) \leq 0 \end{cases}.$$

The proof is very similar to the one used for $NEARFIRST$'s competitive ratio. In fact, the robustness is shown in exactly the same way. For the consistency/smoothness, the delay is bounded by $M+d$ in the same way, where d is the distance of the last request q_f served by OPT to the extreme served second by $PIVOT$. The same lower bounds for $|OPT|$ hold as well. We additionally bound d as a function of the error-dependent values Δ and M. When there is no error, we can bound d to be at most $\frac{|L|+|R|}{2}$ instead of $|L|+|R|$, which gives a better competitive ratio than that of $NEARFIRST$. An important distinction is that we do not make use of Lemma 3, since the algorithm does not consider the amplitudes of L and R.

The formal proof of Theorem 5 is given here. The robustness part of Theorem 5 is implied by Lemma 8. To prove Theorem 5, it remains to show the consistency/smoothness part, which is given by the following lemma.

Lemma 13. *The algorithm $PIVOT$ is $f(\eta,\delta)$-smooth for $3-2(\delta+2\eta) > 0$, where*

$$f(\eta,\delta) = 1 + \frac{1+2(\delta+3\eta)}{3-2(\delta+2\eta)}.$$

We assume without loss of generality that $PIVOT$ first serves the left extreme. By Lemma 9, we see that $|PIVOT| - |OPT| \leq M + d$, where d is the distance of R to the request q_f served last by OPT. We now show a bound on the value of d that depends on η and δ.

Claim 3.18. $d = |R - q_f| \leq (|L|+|R|)(\frac{1}{2}+\delta+2\eta).$

Proof. We first show two inequalities that will be used later to prove the claim. Note that $|R - R_P| \leq M$ and $|L - L_P| \leq M$ by Claim 3.1. The first inequality is

$$|R - R_P| \leq M \implies R - R_p \leq M \implies R_P - q_f \geq R - q_f - M \implies$$

$$R_P - q_f \geq |R - q_f| - M. \tag{3.5}$$

Similarly, we see that

$$|L - L_P| \leq M \implies L_P - L \geq -M \implies q_f - L + M \geq q_f - L_P$$

$$|q_f - L| + M \geq q_f - L_P. \tag{3.6}$$

Because of $PIVOT$'s choice to go left first, we see that

$$|R_P - p_{f'}| \leq |L_P - p_{f'}| \implies R_P - p_{f'} \leq p_{f'} - L_P \implies p_{f'} \geq \frac{R_P + L_P}{2} \implies$$

$$q_{f'} \geq \frac{R_P + L_P}{2} - M \implies q_f \geq \frac{R_P + L_P}{2} - M - \Delta \implies$$

$$q_f - L_P + M + \Delta \geq R_P - q_f - M - \Delta \overset{(3.5),(3.6)}{\implies}$$

$$|q_f - L| + 2M + \Delta \geq |R - q_f| - 2M - \Delta \implies$$

$$|q_f - L| + |R - q_f| + 2M + \Delta \geq 2|R - q_f| - 2M - \Delta \implies$$

$$d = |R - q_f| \leq (|L| + |R|)(\frac{1}{2} + \delta + 2\eta).$$

□

Using Claim 3.18, we can prove Lemma 13.

Proof of Lemma 13. We distinguish two cases based on the order in which OPT serves the requests of the set $\{L, R, q_f\}$. **Case 1.** OPT serves in the order L, R, q_f. Thus, we know that

$$\frac{|PIVOT|}{|OPT|} \leq 1 + \frac{M + d}{2|L| + |R| + d}.$$

The derivative of the right part with respect to d is

$$\frac{|R| + 2|L| - M}{(d + |R| + 2|L|)^2}.$$

Since we have assumed $3 - 2(\delta + 2\eta) > 0 \implies \eta < 1$, this value is always positive. Therefore, we can set d to the maximum value described in Claim 3.18 to obtain the following bound.

$$\frac{|PIVOT|}{|OPT|} \leq 1 + \frac{(|L| + |R|)(\frac{1}{2} + \delta + 2\eta) + \eta(|L| + |R|)}{2|L| + |R| + (|L| + |R|)(\frac{1}{2} + \delta + 2\eta)} \leq$$

$$1 + \frac{1 + 2\delta + 6\eta}{3 + 2\delta + 4\eta} \leq 1 + \frac{1 + 2(\delta + 3\eta)}{3 - 2(\delta + 2\eta)}.$$

Case 2. OPT serves in the order R, L, q_f. In that case, we have

$$\frac{|PIVOT|}{|OPT|} \leq 1 + \frac{d + M}{2|R| + |L| + (|L| + |R| - d)} \overset{3.18}{\leq}$$

$$1 + \frac{(|L|+|R|)(\frac{1}{2}+\delta+2\eta)+\eta(|L|+|R|)}{2|R|+|L|+(|L|+|R|)(\frac{1}{2}-\delta-2\eta)} \leq 1 + \frac{1+2\delta+6\eta}{3-2\delta-4\eta} =$$
$$1 + \frac{1+2(\delta+3\eta)}{3-2(\delta+2\eta)}.$$

In both cases, we have shown the smoothness bound. Therefore, the proof is complete. □

We now give the proof of Theorem 5.

Proof of Theorem 5. By Lemma 8, $PIVOT$ is 3-robust. Also, by Lemma 13, it is $\left(1 + \frac{1+2(\delta+3\eta)}{3-2(\delta+2\eta)}\right)$-smooth. Thus, Theorem 5 follows. □

A 1.25-attack. We make use of the original attack strategy of Section 3.3 yet again to obtain a lower bound for this setting. Our final theorem is presented here.

Theorem 6. *For any $\epsilon > 0$, no algorithm can be $(1.25 - \epsilon)$-competitive for open online TSP on the line under the LF prediction model.*

To prove this theorem, we will again utilize the attack strategy given in the proof of Theorem 2. The inputs generated are the same, except for a new request q_0 placed at the origin and released at $t = 4$. Let ${F_C}'$ denote this new family of inputs. We observe the following lemmas.

Lemma 14. *For any instance f in the family ${F_C}'$, $|OPT| = 4$.*

Proof. We see that the requests of the side which ALG did not exit from are released such that OPT can serve them all and return to the origin by $t = 2$. Additionally, the other side's requests are released such that OPT never has to stop for them either, i.e. it can serve them all and return to the origin by $t = 4$. The request on the origin is released at exactly $t = 4$, so this is also served right as OPT returns to the origin from the second trip. Thus, $|OPT| = 4$. □

In the following, $\alpha(f)$ will refer to the distance of consecutive requests in f, disregarding q_0.

Lemma 15. *For any instance f in the family ${F_C}'$, $|ALG| \geq 5 - 2\alpha(f)$.*

Proof. Suppose for the sake of contradiction that $|ALG| < 5 - 2\alpha(f)$. We can see that $|pos_{ALG}(|ALG|)| \leq 1$. Thus, an algorithm ALG' could copy ALG until it serves all requests and then return to the origin. That would mean that ALG' solves the closed variant of f such that $|ALG'| < 6 - 2\alpha(f)$. Observe that there exists an instance $f' \in F_C$ that is identical to f except for q_0. We can see that ALG' also solves f' in less than $6 - 2\alpha(f) = 6 - 2\alpha(f')$ time units, since f' only contains a subset of the requests in f. Therefore, we have a contradiction to Lemma 7. □

We can now prove Theorem 6.

Proof of Theorem 6. By Lemma 14, we see that $|OPT| = 4$. We also see that $|ALG| \geq 5 - 2\alpha(f)$ by Lemma 15. Thus, we get a competitive ratio of at least $\frac{5-2\alpha(f)}{4}$, which can be arbitrarily close to 1.25, concluding the proof. □

3.5 Experimental Evaluation

We have generated synthetic instances and corresponding predictions and tested our algorithms on them. In this section, we explain how this data was generated and present the results we acquired.

Note that mirroring of the positions of the requests and/or uniform scaling of the positions and release times does not affect the competitive ratio of any algorithm. Therefore, we choose to generate the inputs as explained below.

Generating inputs. In the following, any reference of randomness will correspond to a uniform distribution. We have a maximum number of requests $n_{max} \geq 2$ and a maximum release time r_{max}. Our generator first randomly chooses an integer number of requests $n \in [2, n_{max}]$. Then it randomly chooses a value $c' \in [1, c]$. We then generate the *positions* of the requests as follows. We always have a request at -1 and one at c'. The other $n-2$ requests are randomly placed in the interval $[-1, c']$. The release time of each request is randomly chosen from $[0, r_{max}]$.

Generating predictions. We now briefly explain how the predictions of the $LOCATIONS$ model are generated. Each input generated also comes with a prediction ”mould”. This mould contains n scalars $m_i \in [-1, 1]$, one for each request. At least one of these scalars has an absolute value of 1. For a given error η, we calculate M and then add an offset of $m_i \cdot M$ to the position of request q_i to get the prediction p_i. In this way, at least one prediction is guaranteed to have a distance of M to its associated request.

For the LF prediction model, we simply try each label of the generated input as a different prediction. Each label choice corresponds to a different error δ, which is calculated after choosing the label.

Results. We generated 7500 random input-predictions pairs with at most 20 requests. A value of $c = 2$ was chosen, since higher values of c in general only benefit our algorithms. The maximum release time was set to 6. Again, higher release times in general lead to better competitive ratios for our algorithms, because they increase $|OPT|$.

The error η of these predictions varied from 0 to 1. We ran $FARFIRST$ and $NEARFIRST$ on each of these instances. Additionally, for each of these instances, we ran $PIVOT$ with each of the instance's request labels as the prediction of the LF prediction model. Thus, the $PIVOT$ algorithm was ran approximately 75000 times.

We did not compare the results of our algorithms to the classical algorithms because that would be unfair. That is because our algorithms have the benefit of knowing the number of requests n which helps in practice, even if the theoretical lower bounds are almost identical. In contrast, the theoretically optimal classical online algorithms resort to waiting techniques, which in turn almost always maximizes their competitive ratio to the theoretical bound.

The experiments were executed on a typical modern laptop computer (CPU: AMD Ryzen 7 4700U 2.0 Ghz 8 cores, RAM: 16GB). The execution time did not exceed 2 minutes. We present our results via various graphs in the following subsections.

3.5.1 *FARFIRST*

Figure 3.1 shows the maximum competitive ratio observed for the $FARFIRST$ algorithm in all instances with error η up to the value of the x axis. In figure 3.2, we have also provided a plot that depicts the maximum competitive ratio observed for x % of the best instances with error η up to the value of the y axis. We note that the grid turns red near the very edge, which means that high competitive ratios are rare.

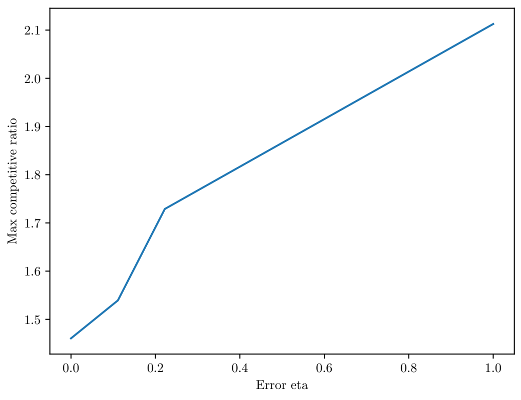

Figure 3.1: *FARFIRST*'s competitive ratio for increasing error. As can be seen in the figure, the competitive ratio never surpasses ≈ 2.15 for $\eta \leq 1$. Additionally, we find that the competitive ratio even with zero error is close to the theoretical upper bound of 1.5. It should also be noted that the theoretical lower bound of 1.64 (without predictions) is broken for η roughly up to 0.2.

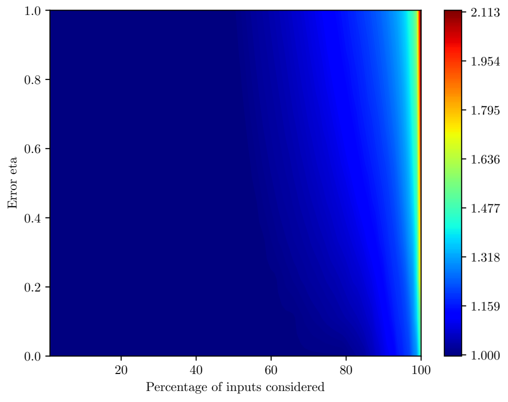

Figure 3.2: *FARFIRST*'s competitive ratio for increasing error and percentage of inputs considered, sorted by the competitive ratio that $FARFIRST$ obtains on them. The narrowness of the red portion of the grid suggests that high competitive ratios are rare. We note that the colors of the grid are generally blue, i.e. $FARFIRST$ exhibits a relatively low competitive ratio in most cases.

3.5.2 *NEARFIRST*

The figures presented here are analogous to those of Section 3.5.1. Figure 3.3 shows the maximum competitive ratio observed for the $NEARFIRST$ algorithm in all instances with error η up to the value of the x axis. In figure 3.4, a plot analogous to that seen in figure 3.2 is shown for the $NEARFIRST$ algorithm. The red portion of the grid is again quite limited as in the case for $FARFIRST$.

3.5.3 *PIVOT*

In this final subsection we condsider the $PIVOT$ algorithm. In figure 3.5, the color of the pixel in coordinates (x, y) corresponds to the maximum competitive ratio observed for all instances with errors $\delta \leq x$ and $\eta \leq y$. We should explain here that the colors change abruptly in this figure since the δ error does not vary smoothly in the generated predictions. This is because we only have a discrete set of choices for the label f' of the LF prediction model through which δ is calculated.

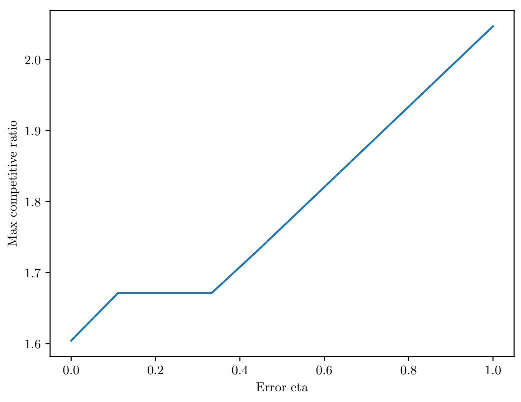

Figure 3.3: *NEARFIRST*'s competitive ratio for increasing error. As can be seen in the figure, the competitive ratio never surpasses ≈ 2.05 for $\eta \leq 1$. Additionally, we find that the competitive ratio even with zero error is close to the theoretical upper bound of $1.\overline{66}$. It should also be noted that the theoretical lower bound of 2 (without predictions) is broken even for η very close to 1.

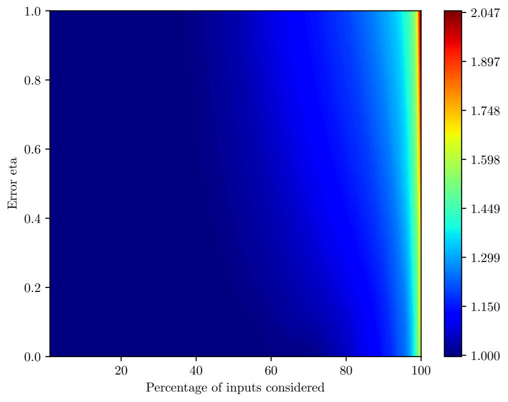

Figure 3.4: *NEARFIRST*'s competitive ratio for increasing error and percentage of inputs considered, sorted by the competitive ratio that NEARFIRST obtains on them. The narrowness of the red portion of the grid suggests that high competitive ratios are rare. We note that the colors of the grid are generally blue, i.e. $NEARFIRST$ exhibits a relatively low competitive ratio in most cases.

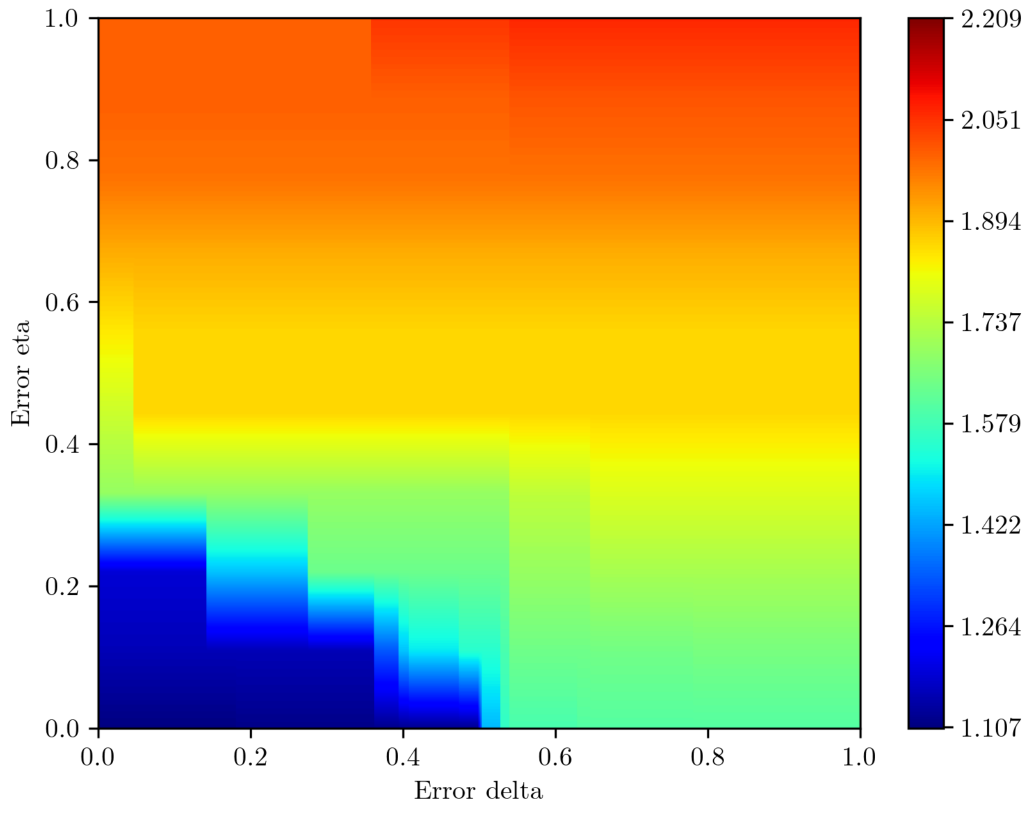

Figure 3.5: *PIVOT*'s competitive ratio for increasing errors δ and η. The color of the pixel in coordinates (x, y) corresponds to the maximum competitive ratio observed for all instances with errors $\delta \leq x$ and $\eta \leq y$. We observe that the competitive ratio is more sensitive to η than to δ, as was to be expected by the corresponding theoretical bound. With perfect predictions, the maximum competitive ratio is not greater than ≈ 1.11, which is considerably lower than the theoretical upper bound of $1.\overline{33}$. In general, the competitive ratio increases smoothly along the main diagonal of the grid. Finally, $PIVOT$'s competitive ratio surpasses the lower bound of 2 (without predictions) only for large values of δ, η.

4. CONCLUSIONS

Throughout this thesis, we investigated various classical online problems and the algorithms designed for them. We observe that the worst case guarantees we have for these algorithms are in general very pessimistic, even if they are indeed optimal. This is due to the fact that a classical online algorithm has to be prepared for every possible input and thus must implement a global strategy which is bound to have drawbacks in some cases.

We demonstrated that this problem can be overcome with the introduction of learning-augmented algorithms, since the particularities of the input may be utilized in this case, given that the predictions are sufficiently accurate. At the same time, we still have global worst case guarantees, which is not true for artificial intelligence/machine learning in general.

Finally, we have examined the online TSP on the line and provided lower bounds as well as algorithms for three different learning-augmented settings. An immediate extension of our results would be to bridge the gap between the lower and upper bounds we have shown for the open variant. Also, it would be interesting to establish error-dependent lower bounds and/or optimal consistency-robustness tradeoffs. Moreover, an improvement would be to remove the assumption of knowing the number of requests n. A technique that could perhaps allow an algorithm to achieve that is to periodically make sure that the algorithm terminates in case no new requests appear. Finally, more general versions of online TSP could be investigated like the case of trees.

TABLE OF TERMINOLOGY

Ξενόγλωσσος όρος	**Ελληνικός όρος**
Online Problems	Προβλήματα με Άμεση Ανταπόκριση
Online Learning-Augmented Algorithms	Άμεσοι Αλγόριθμοι με Προβλέψεις
Online TSP on the Line	Πρόβλημα του Πλανώδιου Πωλητή στον Άξονα με Άμεση Ανταπόκριση
Offline Algorithm	Αλγόριθμος με Καθυστερημένη Ανταπόκριση
Request	Αίτημα
Real line	Άξονας των πραγματικών αριθμών
Origin	Αρχικό σημείο
Makespan	Χρονοκαθυστέρηση
Open variant	Ανοιχτή έκδοση
Closed variant	Κλειστή έκδοση
Competitive Ratio	Λόγος Ανταγωνιστικότητας
Lower bound	Κάτω φράγμα
Prediction model	Μοντέλο προβλέψεων
Consistency	Συνέπεια
Smoothness	Ομαλότητα
Robustness	Ευρωστία
x-competitive	x-ανταγωνιστικός
x-consistent	x-συνεπής

ABBREVIATIONS - ACRONYMS

TSP	Traveling Salesman Problem
LF	LOCATIONS+FINAL
LRU	Least Recently Used
FIFO	First In First Out
FWF	Flush When Full
PQR	Possibly Queue Requests
AVR	AVERAGE RATE
OA	OPTIMAL AVAILABLE
LAS	LEARNING AUGMENTED SCHEDULING

APPENDIX A. LEMMAS FOR THE CASE OF KNOWN *N*

Lemma 16. *For any $\epsilon > 0$, no algorithm can be $(2-\epsilon)$-competitive for open online TSP on the line without predictions when the number of requests n is known. Also, there exists an algorithm that matches this lower bound.*

Proof. A very simple attack can be used to show the lower bound of 2. If $pos_{ALG}(1) \leq 0$, we present a request at 1 with a release time of 1. In the other case, the request's position is -1. It is easy to see that $|OPT| = 1$, while $|ALG| \geq 2$, proving the bound. There also exists a very simple algorithm that matches this bound. Such an algorithm need only wait until all requests have been released and then copy OPT's actions, which are at this point computable. The waiting part does not take more than $|OPT|$ and neither does the moving part, which implies a competitive ratio of 2 for such an algorithm. □

Lemma 17. *For any $\epsilon > 0$, no algorithm can be $(1.64-\epsilon)$-competitive for closed online TSP on the line without predictions when the number of requests n is known. Also, there exists an algorithm that matches this lower bound.*

Proof. By taking a close look at the attack strategy described in Section 3.3 of [8], we observe that the number of requests is never higher than a specific value n_{max}. In fact, it turns out that $n_{max} = 3$, i.e. the attack never uses more than 3 requests. We modify this attack so that it can also be used when n is known. Any instance of the modified attack will have *exactly* n_{max} requests. Thus, the algorithm will be informed that there will indeed be n_{max} requests.

We first give a brief description of the original attack strategy for context. Let $\rho = \frac{9+\sqrt{17}}{8} \approx 1.64$, $I = [-(2\rho-3),(2\rho-3)]$, $I' = [-(7-4\rho),(7-4\rho)]$. Note that I is contained in I' and both of them are contained in $[-1,1]$. If $pos_{ALG}(1) \notin I$, then a single request at -1 or 1 (released at $t=1$) suffices to achieve the competitive ratio. Assuming that $pos_{ALG}(1) \in I$, we simultaneously present two requests at -1 and 1 at $t=1$. At $t=3$, ALG cannot have possibly served both of these requests. If $pos_{ALG}(3) \in I'$, then another request at -1 or 1 (released at $t=3$) is sufficient. Therefore, we continue assuming that $pos_{ALG}(3) \notin I'$. This means that ALG is close to one extreme and still has not served the other. When ALG crosses the origin to serve the other extreme at time $3+x$, a request is placed at either $1+x$ or $-(1+x)$ (depending on which extreme ALG has not served). The competitive ratio turns out to be at least ρ in this (final) case also.

We now describe our modification of this strategy. Initially, the original attack strategy is followed. Let q_{win} be the last request released by the original attack strategy, after the release of which the competitive ratio is guaranteed to be at least 1.64 in case no new requests appear. Let $n_{original}$ be the number of requests released via the original attack strategy. If $n_{original} < n_{max}$, then $n_{max} - n_{original}$ extra requests are released at time $rel(q_{win})$, placed arbitrarily between the origin and q_{win}. These extra requests are served by OPT on the way back from q_{win}, without incurring extra cost. In other words, $|OPT|$ does not increase with the addition of these requests. Also, $|ALG|$ certainly cannot decrease since we only *added* requests. Therefore, the same lower bound holds even for known n.

The algorithm is exactly the same as the one for unknown number of requests, since it can just ignore the number n and still achieve the same competitive ratio. □

APPENDIX B. OMITTED PROOFS FROM SECTION 3.4

In this subsection, we give the formal proofs of two lemmas which we used to prove Theorems 3 and 5.

Lemma 8. *Let ALG be either $NEARFIRST$ or $PIVOT$. Then, ALG is 3-robust.*

Proof. Let t_f denote the latest release time for a fixed instance of the problem. We assume w.l.o.g. that $pos_{ALG}(t_f) \leq \frac{L(t_f)+R(t_f)}{2}$. Note that after t_f, ALG will move to $L(t_f)$ and then to $R(t_f)$. Thus, we observe that

$$|ALG| = t_f + |pos(t_f) - L(t_f)| + |L(t_f) - R(t_f)|. \tag{B.1}$$

We distinguish two cases based on the position of ALG at time t_f.

Case 1. $pos(t_f) \geq L(t_f)$. In this case, we see that

$$\text{(B.1)} \implies |ALG| = t_f + pos(t_f) - L(t_f) + R(t_f) - L(t_f) \leq$$
$$t_f + \frac{L(t_f)+R(t_f)}{2} - L(t_f) + R(t_f) - L(t_f) =$$
$$t_f + \frac{3(|L(t_f)| + |R(t_f)|)}{2} \leq 2.5|OPT| \leq 3|OPT|.$$

Case 2. $pos(t_f) < L(t_f)$. Similarly, we have

$$\text{(B.1)} \implies |ALG| = t_f + L(t_f) - pos(t_f) + R(t_f) - L(t_f) \leq$$
$$2t_f + |R(t_f)| \leq 3|OPT|.$$

□

Lemma 9. *Let ALG be either $NEARFIRST$ or $PIVOT$. Also, let q_f be the request served last by OPT. Assume without loss of generality that ALG serves requests from left to right. Let $d = |q_f - R|$. Then, we have $|ALG| - |OPT| \leq M + d$.*

To prove this lemma, we first give some definitions. Note first that $L_U[t]$ is sort of a "checkpoint" for OPT, meaning that OPT must be located at $L_U[t]$ for some point in time on or after t in order to serve that request. Then, it must move from $L_U[t]$ to q_f. This idea helps us keep track of $|OPT|$ so we can compare it with $|ALG|$.

$D(t)$ denotes the least amount of time necessary to serve all requests to the left of $L_U[t]$ (assuming they have been released) and then move to $L_U[t]$, starting at position $pos_{ALG}(t)$. This amounts to

$$D(t) = |pos_{ALG}(t) - L_O(t)| + |L_O(t) - L_U[t]|.$$

This function exhibits a useful bound property. If it drops to M or below at some time t, it can only increase above M again due to a request release. This property is described more formally in the following claim. But first, another useful definition is given.

We define $L_P[t]$ as the leftmost prediction that is released on or after t. That is, $L_P[t] = min(\{p \in P : rel(\pi(q)) \geq t\})$. If this set is empty, then $L_P[t] = R$.

Using this definition, the following claim can be seen in the same way as Claim 3.5.

Claim B.1. *Let t_{drop} be a time point such that $D(t_{drop}) \leq M$. If t_{next} is the earliest release time of a request after t_{drop}, then*

$$D(t') \leq M, \, \forall \, t' \in [t_{drop}, t_{next}].$$

We now draw our attention to a point in time that is very central to our proof.

Let $t_{release}$ be the latest release time of a request. Note that $L_U[t] = R, \, \forall \, t > t_{release}$. Then, we define

$$t_{chase} = min\{t : \; t_s \leq t \leq t_{release}, \; (D(t') > M, \, \forall \, t < t' \leq t_{release})\}.$$

In essence, similarly to the definition in the previous section, t_{chase} denotes the time after which ALG gets to finish as soon as possible without waiting for predictions or backtracking for requests. In the following, we assume for simplicity and without loss of generality that ALG *always* serves the requests left to right, even if at time $t_{release}$ it is clear that going to the right first is faster. It is true that our algorithm may indeed make such a decision at time $t_{release}$, but that is a trivial optimization that does not invalidate our proof, since it can only decrease $|ALG|$ and by extension, the value $|ALG| - |OPT|$. Under this assumption, we proceed by showing that after t_{chase}, ALG moves to $L_O(t_{chase})$ and then straight to $L_U[t_{chase}]$, serving all intermediate requests on the way. In fact, it also keeps moving to the right until it reaches R and finishes. The following claim can be seen in the same way as Claim 3.8.

Claim B.2. *Let $t' = t_{chase} + D(t_{chase})$. Then, $pos(t') = L_U[t_{chase}]$.*
Also, $|ALG| = t' + |pos(t') - R|$.

We now give the proof of Lemma 9.

Proof of Lemma 9. We distinguish two cases.

Case 1. $t_{chase} = t_s$. This easily implies that $|ALG| = 2|L| + |R|$ by Claim B.2. It remains to show that $|OPT| \geq 2|L| + |R| - M - d$.

If OPT follows the order $L \to R \to q_f$, then

$$|OPT| \geq 2|L| + |R| + d \geq 2|L| + |R| - M - d.$$

On the other hand, if OPT follows the order $R \to L \to q_f$, then

$$|OPT| \geq 2|R| + |L| + (|L| + |R| - d) \geq 3|R| + 2|L| - d \geq 2|L| + |R| - M - d.$$

Case 2. $t_{chase} > t_s$. It can be seen then by Claim B.1 that $D(t_{chase}) \leq M$. It is easy to see that $|OPT| \geq t_{chase} + |L_U[t_{chase}] - q_f|$. At the same time, by Claim B.2 we see that

$$|ALG| = t_{chase} + D(t_{chase}) + |L_U[t_{chase}] - R| \leq$$

$$t_{chase} + M + |L_U[t_{chase}] - q_f| + |q_f - R| \leq |OPT| + M + d.$$

□

BIBLIOGRAPHY

[1] Lingqing Ai, Xian Wu, Lingxiao Huang, Longbo Huang, Pingzhong Tang, and Jian Li. The multi-shop ski rental problem. *arXiv*, 2014.

[2] Lachlan L.H. Andrew, Minghong Lin, and Adam Wierman. Optimality, fairness, and robustness in speed scaling designs. *SIGMETRICS Perform. Eval. Rev.*, 38(1):37–48, jun 2010.

[3] Spyros Angelopoulos, Christoph Dürr, Shendan Jin, Shahin Kamali, and Marc Renault. Online computation with untrusted advice. *arXiv*, 2019.

[4] Antonios Antoniadis, Christian Coester, Marek Elias, Adam Polak, and Bertrand Simon. Online metric algorithms with untrusted predictions. *arXiv*, 2020.

[5] Antonios Antoniadis, Peyman Jabbarzade Ganje, and Golnoosh Shahkarami. A novel prediction setup for online speed-scaling. *arXiv*, 2021.

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

[7] N. Ascheuer, M. Grötschel, S. O. Krumke, and J. Rambau. Combinatorial online optimization. In *Operations Research Proceedings 1998*, pages 21–37, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[8] G. Ausiello, M. Demange, L. Laura, and V. Paschos. Algorithms for the on-line quota traveling salesman problem. *Information Processing Letters*, 92(2):89–94, 2004.

[9] Etienne Bamas, Andreas Maggiori, Lars Rohwedder, and Ola Svensson. Learning augmented energy minimization via speed scaling. In *Advances in Neural Information Processing Systems*, volume 33, pages 15350–15359. Curran Associates, Inc., 2020.

[10] Étienne Bamas, Andreas Maggiori, and Ola Svensson. The primal-dual method for learning augmented algorithms. *arXiv*, 2020.

[11] Nikhil Bansal, David Bunde, Ho-Leung Chan, and Kirk Pruhs. Average rate speed scaling. In *Algorithmica*, volume 60, pages 240–251, 04 2008.

[12] Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Speed scaling to manage energy and temperature. *J. ACM*, 54(1), mar 2007.

[13] L. A. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Systems Journal*, 5(2):78–101, 1966.

[14] Giulia Bernardini, Alexander Lindermayr, Alberto Marchetti-Spaccamela, Nicole Megow, Leen Stougie, and Michelle Sweering. A universal error measure for input predictions applied to online graph problems. *arXiv*, 2022.

[15] Antje Bjelde, Jan Hackfeld, Yann Disser, Christoph Hansknecht, Maarten Lipmann, Julie Meißner, Miriam SchlÖter, Kevin Schewior, and Leen Stougie. Tight bounds for online tsp on the line. *ACM Trans. Algorithms*, 17(1), dec 2021.

[16] Michiel Blom, Sven O. Krumke, Willem de Paepe, and Leen Stougie. The online-tsp against fair adversaries. In *Algorithms and Complexity*, pages 137–149, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[17] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, USA, 1998.

[18] Eugene B. Dynkin. The optimum choice of the instant for stopping a markov process. *Soviet Math. Dokl*, 1962.

[19] Paul Dütting, Silvio Lattanzi, Renato Paes Leme, and Sergei Vassilvitskii. Secretaries with advice. *arXiv*, 2020.

[20] Hiroshi Fujiwara, Takuma Kitano, and Toshihiro Fujito. On the best possible competitive ratio for multislope ski rental. In *Proceedings of the 22nd International Conference on Algorithms and Computation*, ISAAC'11, page 544–553, Berlin, Heidelberg, 2011. Springer-Verlag.

[21] Marco E. Gerards, Johann L. Hurink, and Philip K. Hölzenspies. A survey of offline algorithms for energy minimization under deadline constraints. *J. of Scheduling*, 19(1):3–19, feb 2016.

[22] John P. Gilbert and Frederick Mosteller. Recognizing the maximum of a sequence. *Journal of the American Statistical Association*, 61(313):35–73, 1966.

[23] Sreenivas Gollapudi and Debmalya Panigrahi. Online algorithms for rent-or-buy with expert advice. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2319–2327. PMLR, 09–15 Jun 2019.

[24] Anupam Gupta, Amit Kumar, Martin P´al, and Tim Roughgarden. Approximation via cost sharing: Simpler and better approximation algorithms for network design. *J. ACM*, 54(3):11–es, jun 2007.

[25] Patrick Jaillet and Michael Wagner. Online routing problems: Value of advanced information as improved competitive ratios. *Transportation Science*, 40:200–210, 05 2006.

[26] Karlin, Kenyon, Randall, and Dana. Dynamic tcp acknowledgment and other stories about e/(e - 1). *Algorithmica*, 36, 07 2003.

[27] Anna R. Karlin, Mark S. Manasse, Lyle A. McGeoch, and Susan Owicki. Competitive randomized algorithms for non-uniform problems. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '90, page 301–309, USA, 1990. Society for Industrial and Applied Mathematics.

[28] Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel D. Sleator. Competitive snoopy caching. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 244–254, 1986.

[29] Thomas Kesselheim, Klaus Radke, Andreas Abels, and Berthold Vöcking. An optimal online algorithm for weighted bipartite matching and extensions to combinatorial auctions. In *ESA*, 09 2013.

[30] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. *arXiv*, 2017.

[31] D. V. Lindley. Dynamic programming and decision theory. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 10(1):39–51, 1961.

[32] Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. *arXiv*, 2018.

[33] Tomomi Matsui and Katsunori Ano. Lower bounds for bruss' odds problem with multiple stoppings. *Mathematics of Operations Research*, 41(2):700–714, 2016.

[34] A. Meyerson. The parking permit problem. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, pages 274–282, 2005.

[35] Michael Mitzenmacher. A model for learned bloom filters and optimizing by sandwiching. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[36] Michael Mitzenmacher. Scheduling with predictions and the price of misprediction. *arXiv*, 2019.

[37] Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. *arXiv*, 2020.

[38] Benjamin Moseley, Sergei Vassilvitskii, Silvio Lattanzi, and Thomas Lavastida. Online scheduling via learned weights. In *SODA 2020*, 2020.

[39] Harilaos N. Psaraftis, Marius M. Solomon, Thomas L. Magnanti, and Tai-Up Kim. Routing and scheduling on a shoreline with release times. *Management Science*, 36(2):212–223, 1990.

[40] Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[41] Dhruv Rohatgi. Near-optimal bounds for online caching with machine learned advice. *arXiv*, 2019.

[42] J. S. Rose. Selection of nonextremal candidates from a random sequence. *J. Optim. Theory Appl.*, 38(2):207–219, oct 1982.

[43] Steven S. Seiden. A guessing game and randomized online algorithms. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, STOC '00, page 592–601, New York, NY, USA, 2000. Association for Computing Machinery.

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

[45] Krzysztof J. Szajowski. Optimal choice of an object with ath rank. *Mathematica Applicanda*, 10:51–65, 1982.

[46] Robert J. Vanderbei. The postdoc variant of the secretary problem. *Mathematica Applicanda*, Vol. 49, no. 1:3–13, 12 2021. Opracowanie rekordu ze środków MNiSW, umowa Nr 461252 w ramach programu ”Społeczna odpowiedzialność nauki” - moduł: Popularyzacja nauki i promocja sportu (2021).

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

[48] Alexander Wei. Better and simpler learning-augmented online caching. *arXiv*, 2020.

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

[50] Mike Worboys. The travelling salesman problem (a guided tour of combinatorial optimisation). *The Mathematical Gazette*, 70(454):327–328, 1986.

[51] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 374–382, 1995.