Length-Weighted Disjoint Path Allocation
Advice and Parametrization

Elisabet Burjons, Fabian Frei(✉), Jasmin Smula, and David Wehner

Department of Computer Science, ETH Zurich, Zürich, Switzerland
{eburjons,fabian.frei,david.wehner}@inf.ethz.ch

Abstract. We modify one of the foundational online problems, *Disjoint Path Allocation*, to include weighted requests. We provide a comprehensive competitive analysis, incorporating the viewpoints of both advice complexity and parametrized complexity. Our bounds feature a consistent parametrization and closely trace the trade-off between advice complexity and competitiveness.

Jointly dedicated to an Unceasingly Requested Algorithmist on his threescore Jubilee.

1 Motivation

Imagine an incredibly intelligent and industrious individual. Take the reasonably random example of a professor in computer science at a renowned university in the heart of Europe. Said professor has to meticulously manage his or her time schedule—we will arbitrarily stick with "his" for the purpose of this paper. He is invited to an influential school to give its teachers a brief talk on the education in his field, illuminating computer science's core principals and providing tried and tested tools to guide our future generations. As always, the organizers wish for an immediate confirmation whether he will embrace this offer. Undoubtedly, he would love to do so; nevertheless, he needs to check whether he is already booked at the requested time.[1]

Even if the booking remains a viable possibility, however, he is of course prudent enough to not accept any invitation precipitately. After all, it might interfere with another opportunity yet to come—envision an extensive exposition of his educational efforts and an ensuing exemplary execution for the government's leading officials. Such a feasible future request might fill the available time window even better, thereby minimizing his idle time. As a conscientious person valuing reliability, going back on a made commitment is no option to him, limiting the flexibility even further. In order to cope successfully with all the diverse and demanding duties, a good guideline for accepting or rejecting requests is thus indispensable.

We try to help our professor in finding and analyzing the best acceptance strategies.

[1] This is the most probable case. Inexplicably, he still finds time to mentor his doctoral students.

H.-J. Böckenhauer et al. (Eds.): Hromkovič Festschrift, LNCS 11011, pp. 231–256, 2018.
https://doi.org/10.1007/978-3-319-98355-4_14

2 Introduction and Definitions

In this section, we introduce the model of our problem, the necessary formal framework, and the tools used for its analysis.

2.1 Length-Weighted Disjoint Path Allocation

The task at hand can be described as follows. First, we are given a time span of length $\ell \in \mathbb{N}$. It is measured in a reasonable time unit, assumed to be indivisible. Certainly, we should not be micro-managing single seconds. The time span can be visualized as a path of length ℓ, as it might appear in a day planner. Switching from the motivating model to a more tangible example, suppose there is a meeting room that can be booked, initially available over the entire time span. Inevitably, an unknown number of reservation requests will start pouring in. Every request corresponds to a *subpath* and demands immediate *acceptance* or *rejection*. Once taken, a decision cannot be revoked. Our goal is to *maximize* the room's occupancy, the overall time it is in use.

This is of course an *optimization problem*, but also a typical *online problem*. Such problems appear abundantly in all areas of computer science. They are characterized by the fact that a solving algorithm $\mathcal{A}$ is forced to produce parts of the output without complete knowledge of the input.

We now define our problem of *Length-Weighted Disjoint Path Allocation* on paths, LWDPA for short, more formally. An instance I consists of a path P of length ℓ and n requests r_i, which are just subpaths of P. An algorithm either *accepts* or *rejects* each request. It cannot accept a request that *overlaps* a previously accepted request, where two requests are said to *overlap* if they share at least one edge. In this case, we say that they *block* each other. The requests accepted by an algorithm $\mathcal{A}$ on an instance I constitute the *solution* $\mathcal{A}(I)$ of $\mathcal{A}$ on I. A solution is thus a selection of *non-overlapping* requests. For a fixed solution, we refer to the leaves of the accepted requests, that is, the starting and ending points of the chosen subpaths, as *transition points*.

The *gain* of a solution to I computed by $\mathcal{A}$ is the total length of all accepted requests, denoted by $\text{gain}(\mathcal{A}(I))$. A solution is *optimal* if it has the maximum possible gain. An *optimal algorithm* OPT produces an optimal solution $\text{OPT}(I)$ with gain $\text{gain}(\text{OPT}(I))$ on every instance I. We refer to the accepted requests of a fixed optimal solution as the *optimal requests*.

In accordance with the conventional way of defining related problems—see Subsect. 2.2—we assume that all requests are unique; in other words, no subpath is offered more than once. Moreover, we assume that the algorithm is not given the number n of requests beforehand and also does not recognize the last request as such. This is a severe constraint for any algorithm.

2.2 Connection with Classic Disjoint Path Allocation

Among all researched online problems, the problem of *Disjoint Path Allocation* (DPA) is one of the earliest. It has been thoroughly analyzed in many different

settings [1–3,8]. The difference to our *Length-Weighted Disjoint Path Allocation* (LWDPA) lies in a single but significant detail: For DPA, the gain of a solution is defined as the number of requests it contains; for LWDPA, on the other hand, we weight the requests by their length. Thus, the gain is the total length of all accepted requests.

Both DPA and LWDPA can be altered to allow preemption, that is, the removal of provisionally accepted requests from the solution. The preemptive model has been the focus of the work by Kováčová [11]. Nevertheless, it contains a few results for the non-preemptive case. Her Theorems 8, 9, 12 and 13 correspond to the unparametrized special case of our optimality results in Subsect. 3.1, namely Theorems 1 to 4. Theorem 18 gives an upper bound that is asymptotically improved upon in Theorem 7 by choosing $b > 0$.

DPA is traditionally motivated by a network wherein permanent communication channels between pairs of nodes are established. It is unknown what the connections are used for; consequently, the only goal is to maximize their number. In contrast, our motivating example of LWDPA illustrates why maximizing the number of covered edges will be preferred instead under the right circumstances.

2.3 Competitive Analysis

Competitive analysis was introduced by Sleator and Tarjan as a tool to measure the performance of online algorithms [12]. The basic idea is to assess the quality of the solution computed by an online algorithm by comparing it to an optimal solution that is calculated *offline*, that is, by an algorithm with prior knowledge of the whole input. Note that no limit on the computational complexity is imposed; enforcing online decisions is restrictive enough. An offline algorithm can, therefore, always deliver an optimal solution, if necessary by applying brute force.

In the present case of a maximization problem, the following two notions enable us to express the mentioned comparison succinctly:

Definition 1 (Strict Competitiveness). $\mathcal{A}$ *is* strictly c-competitive *if*

$$\forall I\colon \ \frac{\text{gain}(\text{OPT}(I))}{\text{gain}(\mathcal{A}(I))} \leq c.$$

The best attainable strict competitiveness is 1. This is only achieved if $\mathcal{A}$ always produces an optimal solution. The poorer $\mathcal{A}$'s handling of any worst-case instance, the larger a c we need to choose.

Definition 2 (Strict Competitive Ratio). *The strict competitive ratio of an online algorithm $\mathcal{A}$ is the infimum over all c that satisfy the condition in Definition 1.*

A thorough introduction to online computation and competitive analysis can be found in the textbook by Borodin and El-Yaniv [4].

2.4 Advice Complexity

The competitive analysis of online algorithms compares them to offline algorithms. One might say that this sets the bar far too high. Often, an online algorithm has no chance to come even near the performance of an offline algorithm.

Here, the concept of *advice complexity* comes into play, offering a more fine-grained classification of online problems. The aim is to answer the question of how much information about the complete input is required for an online algorithm to guarantee any given competitiveness. Arguably the most fundamental question in this regard is how much knowledge about the input suffices for an online algorithm to ensure an optimal solution. With less information it is generally more difficult to solve a problem well; hence, a trade-off between the necessary amount of information and the achievable competitive ratio occurs.

We allow for any kind of information and measure it in bits. We model this by an oracle with infinite computational power that delivers, in dependence of the given instance, an infinite binary string of *advice bits*. This string is provided to the online algorithm along with the instance in the form of an infinite tape. The online algorithm may read a prefix of arbitrary length from the tape and adjust its behavior accordingly. We refer to the read prefix as the *advice string* and call its length the *advice complexity* of the algorithm on the given instance.

An online algorithm that reads at most b advice bits can also be interpreted as a collection of 2^b classic deterministic algorithms, one for each advice string. Conversely, we need $\log B$ advice bits[2] to simulate B distinct deterministic algorithms.[3] This viewpoint will come in handy in our proofs.

Across all possible online algorithms, there is a minimum number of advice bits that must be read to process any problem instance optimally. This minimal advice complexity can be interpreted as a fundamental property of the problem, its *information content*.

The concept of advice complexity was introduced by Dobrev et al. [6] and subsequently refined by Hromkovič et al. [9], Böckenhauer et al. [3], and Emek et al. [7]. In particular the work by Hromkovič et al. [9] elaborated on the intriguing idea of advice complexity as a tool to measure a problem's information content. Advice complexity has since been successfully applied to numerous online problems. For a recent and well-written introduction to both online problems and the concept of advice, we refer to Komm [10].

2.5 Parametrization

For many online problems, bounds are naturally given as functions of n, the number of requests. For DPA as well as LWDPA, however, there is the alternative

[2] Throughout this paper, log denotes the logarithm to base 2.

[3] Note that even $\lceil \log B \rceil$ advice bits are necessary. However, we will often omit ceiling and floor brackets and assume divisibility in our proofs for the sake of better readability.

to express results in terms of the path length ℓ instead. We present results for both choices.

Our analysis of the LWDPA problem is consistently parametrized by $k_{\min}$ and $k_{\max}$, a guaranteed lower and upper bound, respectively, on the length of the requests in the instances. These two parameters are known to both the algorithm and the oracle. We denote the ratio between these two bounds by $\varphi := k_{\max}/k_{\min}$. Naturally, we have $1 \leq k_{\min} \leq k_{\max} \leq \ell$. When we restrict the instances to requests of a single length, DPA and LWDPA coincide. This corresponds to the particular case of $k_{\min} = k_{\max}$, for which good bounds are already proven in the bachelor's thesis by Dietiker [5]; we therefore assume $k_{\min} < k_{\max}$. Our parametrized results are a true generalization in the sense that setting $k_{\min} = 1$ and $k_{\max} = \ell$ yields meaningful bounds on the strict competitive ratio in the classic sense.

Why would we want to parametrize the LWDPA problem in this way? We get a good intuition by going back to the motivating example of a meeting room that is available for booking over a time span of length ℓ. Blocks of 5 minutes could be a reasonable time unit, providing sufficient flexibility without an excessive number of options. Since nobody properly enjoys concise meetings, we might want to enforce a minimum reservation length of half an hour. With our parametrization, we can model this by setting $k_{\min}$ to 6, yielding a minimum of $k_{\min} \cdot 5 = 30$ minutes. One might object that we could have just as well stretched the time unit to half an hour, with no need for introducing new parameters. This, however, would leave no option between 30 minutes and a full hour, thereby preventing a reasonable booking time for, say, a 45-minute exercise class. Imposing upper bounds on the reservation length via $k_{\max}$, on the other hand, could make sense in order to avoid permanent reservations and give everybody a booking opportunity.

Altogether, we see that the proposed parametrization in terms of $k_{\min}$ and $k_{\max}$ is well justified and will help us to deal better with boundary conditions of this sort. Note that $k_{\min}$ and $k_{\max}$ are only bounds on the request lengths known to the algorithm and not necessarily the actual lengths of the shortest and longest requests. In the example, we know that every meeting will last at least half an hour; however, this does not mean we can count on the appearance of a booking request for a 30-minute meeting.

3 Results

This section presents all of our results for LWDPA in consistent parametrization. It is structured as follows.

In Subsect. 3.1, we give upper and lower bounds on the amount of advice bits needed to achieve optimality. In Subsect. 3.2, we prove a tight bound on the strict competitive ratio of the problem without advice.

Subsection 3.3 bridges the gap between these two extremes; we present results that trace the trade-off between strict competitiveness and advice complexity. The first one, Theorem 6, is an upper bound witnessed by an intelligent greedy

algorithm. It performs exceptionally well with only few advice bits. The performance deteriorates quickly with increasing advice complexity, however. Theorem 7 cures this shortcoming with an advice-assisted preprocessing; it is well-suited for situations where much advice is available. The last two theorems give the complementing lower bounds. Theorem 8 is a good match to Theorem 6; it features a hard instance set that is tailor-made against the greedy approach. Theorem 9, on the other hand, is a rather technical but all the more flexible result, yielding multiple useful corollaries.

The conclusion in Sect. 4 recapitulates the results and discusses what light they shed on the advice complexity behavior of LWDPA.

3.1 Optimality

This subsection answers the question of how much advice an online algorithm needs in order to produce an optimal solution to every instance. Theorems 1 and 3 give an answer that depends on n, whereas Theorems 2 and 4 address the question in terms of the path length ℓ. It is worth noting that we have two closely matching pairs of upper and lower bounds and even a perfect match in the unparametrized case of $k_{\min} = 1$ and $k_{\max} = \ell$.

Theorem 1. *Optimality can be achieved with n advice bits.*

Proof. The oracle chooses an arbitrary but fixed optimal solution. Then, the oracle specifies for every request whether it is part of this solution or not, using n advice bits. Whenever the algorithm receives a request, it reads the next advice bit from the tape and accepts or rejects accordingly. This strategy does not depend on the values of $k_{\min}$ and $k_{\max}$. □

Theorem 2. *Optimality can be achieved with $(\ell - 1)/k_{\min} \cdot (1 + 2\log(k_{\min}))$ advice bits.*

Proof. Before we describe how the oracle determines the advice given to the algorithm $\mathcal{A}$, let us consider the subpaths of P consisting of $k_{\min}$ vertices. We first show that in any optimal solution to an instance I, any such subpath can contain at most two transition points. For the sake of contradiction, fix some subpath S of length $k_{\min} - 1$ and some optimal solution $\textsc{Opt}(I)$ and assume that S contains at least three transition points of $\textsc{Opt}(I)$. Then we can pick three consecutive transition points from S and call them (from left to right) u, v, and w. Since v is the middle transition point, $\textsc{Opt}(I)$ must contain at least one of the requests (u, v) and (v, w). However, since S only contains $k_{\min}$ vertices, the subpaths (u, v) and (v, w) both have length at most $k_{\min} - 1$ and are thus shorter than the minimum guaranteed request length. Hence, they can neither be part of the instance I nor $\textsc{Opt}(I)$, yielding the contradiction.

As depicted in Fig. 1, we now divide the $\ell - 1$ inner vertices of the path P into $(\ell - 1)/k_{\min}$ segments containing $k_{\min}$ vertices each. For the given input sequence I, the oracle chooses some arbitrary but fixed optimal solution $\textsc{Opt}(I)$ and uses the advice bits to indicate the positions of the transition points

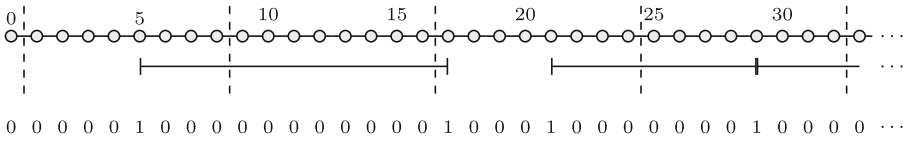


Fig. 1. An example of a possible optimal solution $\textsc{Opt}(I)$. The dashed lines indicate the segments containing $k_{\min}$ vertices each. The bit string underneath indicates all transition points of $\textsc{Opt}(I)$, enabling $\mathcal{A}$ to act optimally on I. The algorithm can deduce this string by reading only $1 + 2\log k_{\min}$ advice bits per segment.

of $\textsc{Opt}(I)$ in each segment. To calculate the number of advice bits necessary, let us count the number of cases the oracle has to distinguish. Taking into account that the number of transition points in a segment might be two, one, or zero, the number of different possible combinations that have to be distinguished is

$$\binom{k_{\min}}{2} + k_{\min} + 1 = \frac{k_{\min}(k_{\min}-1) + 2k_{\min} + 2}{2} = \frac{k_{\min}^2 + k_{\min} + 2}{2},$$

which can be bounded from above by $2k_{\min}^2$. The number of advice bits necessary to encode that many different possible combinations of transition points per segment is

$$\frac{\ell - 1}{k_{\min}} \cdot \log\bigl(2k_{\min}^2\bigr) = \frac{\ell - 1}{k_{\min}} \cdot (1 + 2\log(k_{\min})).$$

The algorithm $\mathcal{A}$ reads exactly this many advice bits. (It can do so even before the first request arrives.) From this information, $\mathcal{A}$ can derive all transition points of $\textsc{Opt}(I)$ except the first and last vertex of P, which can always be treated as transition points. Now it only has to accept every presented request whose starting and ending points coincide with two successive transition points, and reject all others. This way, $\mathcal{A}$ never accepts any requests that block any requests from $\textsc{Opt}(I)$. Hence, $\mathcal{A}$ accepts all requests from $\textsc{Opt}(I)$ and must be optimal itself. □

For these two upper bounds, it was sufficient to provide an algorithm with the given advice complexity that performs optimally on all instances. For the corresponding lower bounds, we now take the adversary's position and show that there is always an instance that the algorithm will solve suboptimally, unless given a certain number of advice bits.

Theorem 3. *At least* $n - 1 - \lfloor (n-1)/(k_{\max} - k_{\min} + 1) \rfloor$ *advice bits are needed for optimality.*

Proof. We describe a set $\mathcal{I}$ of instances such that no deterministic algorithm can solve two of them optimally. This implies that an optimal algorithm must be able to distinguish them all, so $\log(|\mathcal{I}|)$ advice bits are required.

To give a lower bound in n, we can choose the path length as large as necessary. We fix it to be $\ell := nk_{\min}$ for all instances from $\mathcal{I}$. Every instance

is presented in k blocks that consist of $i_1, \ldots, i_k$ requests, respectively; hence $i_1 + \cdots + i_k = n$. The first block has i_1 requests of lengths $k_{\min}, \ldots, k_{\min} + i_1 - 1$; they are all aligned on the left endpoint and presented in increasing order. All following blocks are constructed in the same way: Block j has i_j requests of lengths $k_{\min}, \ldots, k_{\min} + i_j - 1$, aligned as far as possible to the left, without overlapping the previous block. See Fig. 2 for an example. Every new request protrudes further to the right: by 1 if it stays in the same block, and by $k_{\min}$ if it starts a new one. Thus, the path length $\ell = n \cdot k_{\min}$ is indeed just large enough to accommodate all of them.

The optimal solution, obtained by accepting the last request of each block and highlighted in Fig. 2, is unique. Hence, already a single mistake prevents optimality. We prove now that a deterministic algorithm cannot handle any two such instances $I \neq I'$ optimally, but will make a mistake on one of them. The requests for I and I' are identical up to some point. Let r_j be the first request that differs between them. Like every request, r_j either starts a new block or continues an existing one. Only in the former case is r_{j-1} the last request in a block and has to be accepted. Since the deterministic algorithm cannot distinguish the two instances before r_j is presented, it is bound to make a mistake on one of them.

It remains to give a good estimate on the number of instances in $\mathcal{I}$. For this, we consider how these instances can be constructed step by step, adding the n requests one after another. The first request in the first block is identical for all instances; it has length $k_{\min}$. In each of the following $n - 1$ steps we can either start a new block with a request of length $k_{\min}$ or—if the preceding request is shorter than $k_{\max}$—extend the current block downward by a request that is one unit longer. The second option is excluded only if the preceding request has the maximum length $k_{\max}$. How often can this situation, namely a request of length $k_{\max}$ preceding another one, occur at most? There are $n - 1$ requests preceding another one. Also, any block featuring a request of length $k_{\max}$ comprises $k_{\max} - k_{\min} + 1$ requests. Therefore, the answer is

$$d := \left\lfloor \frac{n-1}{k_{\max} - k_{\min} + 1} \right\rfloor.$$

Thus, we always have at least $n - 1 - d$ binary choices during the construction of an instance and the number of different instances is bounded from below by $|\mathcal{I}| \geq 2^{n-1-d}$. As a result, the required number of advice bits is at least $\log |\mathcal{I}| \geq n - 1 - d$. □

Theorem 4. *At least* $\ell/k_{\min} - 1 - \lfloor (\ell/k_{\min} - 1)/(k_{\max} - k_{\min} + 1) \rfloor$ *advice bits are needed for optimality.*

Proof. The approach from the proof of Theorem 3 works here as well. In fact, we can use the exact same set $\mathcal{I}$ of hard instances. To obtain a lower bound that now depends on ℓ, we are free to choose the number n of requests conveniently. Since we had $\ell = nk_{\min}$ for all instances from $\mathcal{I}$ before, we now set $n := \ell/k_{\min}$. Using this to substitute n in Theorem 3 directly yields the desired result. □

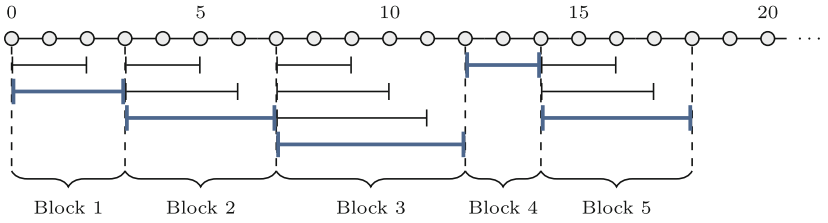


Fig. 2. An instance $I \in \mathcal{I}$ with $k_{\min} = 2$, $k_{\max} = 5$, $n = 13$ and $k = 5$. Requests are presented from top to bottom and from left to right. The highlighted optimal solution contains the last request of each block. Each block is adjacent to the previous one.

3.2 No Advice

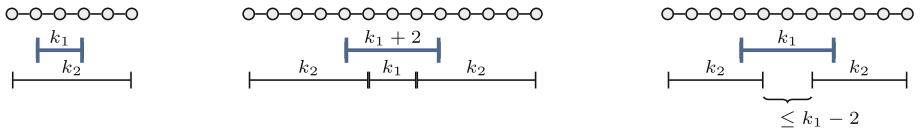
Having explored the boundaries on advice complexity for optimality, we now turn to the opposite end of the spectrum: What is the best strict competitive ratio achievable by a classic online algorithm operating without advice? First, we analyze the strict competitive ratio on only a subset of the requests, for which even better lower and upper bounds on the request lengths might apply. In Lemma 1, we give a case-based upper bound for this scenario, and we extract a more general bound that encompasses all cases in Corollary 1. Although the advantage of considering only subsets of requests may not be obvious at first, this will prove to be useful for Theorem 6 in the next section. Additionally, we can directly derive from Lemma 1 a greedy algorithm that is given in Theorem 5, together with a perfectly matching lower bound.

Lemma 1. *Let I be some input sequence for* LWDPA *and let $I' \subseteq I$ be a subset of its requests. Moreover, let k_1 and k_2 be lower and upper bounds, respectively, on the lengths of all requests from I', with $k_{\min} \leq k_1 \leq k_2 \leq k_{\max}$. Then, the strict competitive ratio of* GREEDY*—the algorithm that accepts every request that is not blocked yet—on the requests from I' is*

$$c \leq \begin{cases} k_2 & \text{if } k_1 = 1, \\ (2k_2 + k_1)/(k_1 + 2) & \text{if } k_1 \geq 2 \text{ and } k_1 + 1 < k_2 < k_1^2/4, \\ 2k_2/k_1 & \text{otherwise.} \end{cases}$$

Proof. The worst case for the greedy algorithm occurs in different situations, depending on the lengths k_1 and k_2. We distinguish several cases depending on the length of an accepted request in comparison to the length of the subpath which the blocked requests could have covered otherwise. All relevant cases are depicted in Fig. 3.

If $k_1 = 1$, only the cases depicted in Fig. 3a and 3b can occur. In the case corresponding to Fig. 3a, GREEDY accepts one request of length $k_1 = 1$, which later blocks an optimal request of length k_2, leading to a strict competitive ratio of k_2. In the case corresponding to Fig. 3b, GREEDY accepts a request of length $k_1 + 2$, which blocks three requests appearing later: One request of



(a) Case 1. **(b)** Case 2. **(c)** Case 3.

Fig. 3. The highlighted requests are those accepted by GREEDY. The three figures show which requests can be presented afterwards such that the worst case for GREEDY occurs, depending on k_1 and k_2. In case 1, a request of length k_1 is accepted, blocking an optimal request of length k_2. This situation can always occur, independently of the values k_1 and k_2. Case 2 is only possible if $k_2 \geq k_1 + 2$. Here, an accepted request of length $k_1 + 2$ can block one request of length k_1 and two requests of length k_2. In case 3, two requests of length k_2 are blocked by one accepted request of length k_1. This can only happen if $k_1 \geq 2$.

length $k_1 = 1$ and two requests of length k_2, yielding a strict competitive ratio of $(2k_2 + 1)/3 \leq k_2$. The strict competitive ratio in the case $k_1 = 1$ is thus $c \leq k_2$.

If $k_1 \geq 2$, we further distinguish two subcases. If $k_2 \leq k_1+1$, the case depicted in Fig. 3b cannot occur. We can quickly convince ourselves that, under this assumption, the worst case realized in the situation of Fig. 3c is where GREEDY accepts one request of length k_1 which blocks two requests of length k_2. This gives us a strict competitive ratio of $c \leq 2k_2/k_1$ in this case.

If $k_1 \geq 2$ and $k_2 \geq k_1 + 2$, the worst case occurs when requests are presented according to Fig. 3b or 3c. (Requests may also be presented according to Fig. 3a; however, requests arriving according to Fig. 3c always ensure an even worse performance of the greedy algorithm.) We have to distinguish two further subcases; $k_2 \geq k_1^2/4$ and $k_2 < k_1^2/4$. In the former case, the worst-case strict competitive ratio is $2k_2/k_1$, as

$$\frac{2k_2}{k_1} = \frac{2k_2(k_1+2)}{k_1(k_1+2)} = \frac{2k_2k_1 + 4k_2}{k_1(k_1+2)} \geq \frac{2k_2k_1 + k_1^2}{k_1(k_1+2)} = \frac{2k_2 + k_1}{k_1+2}.$$

Analogously, in the case that $k_2 < k_1^2/4$, the strict competitive ratio obtained in the worst case is $(2k_2 + k_1)/(k_1 + 2)$. □

Corollary 1. *Let I be some input sequence for* LWDPA *and let $I' \subseteq I$ be a subset of its requests. Moreover, let k_1 and k_2 be lower and upper bounds, respectively, on the lengths of all requests from I', with $k_{\min} \leq k_1 \leq k_2 \leq k_{\max}$. Then, the strict competitive ratio of* GREEDY *on the requests from I' is bounded from above by $2k_2/k_1 + 1$.*

Proof. We only have to consider the three different cases from Lemma 1 and make sure the strict competitive ratio is bounded from above by $2k_2/k_1 + 1$ in each case. It is obvious that the claim holds for the first and the last case. For the second case, we make the following transformation.

$$\frac{2k_2 + k_1}{k_1 + 2} < \frac{2k_2 + k_1}{k_1} = \frac{2k_2}{k_1} + 1. \qquad \square$$

Theorem 5. *There is an online algorithm without advice, namely* GREEDY*, that has a strict competitive ratio of c, and no online algorithm without advice can have a strict competitive ratio better than c, with*

$$c = \begin{cases} k_{\max} & \text{if } k_{\min} = 1, \\ (2k_{\max} + k_{\min})/(k_{\min} + 2) & \text{if } k_{\min} \geq 2 \text{ and } k_{\min} + 1 < k_{\max} < k_{\min}^2/4, \\ 2k_{\max}/k_{\min} & \text{otherwise.} \end{cases}$$

Proof. The first part of the statement follows directly from Lemma 1 by setting $I' := I$, $k_1 := k_{\min}$, and $k_2 := k_{\max}$.

For the second part, we make the following considerations. Since we cannot use any advice bits, we only have one deterministic algorithm available. Thus, let A be some arbitrary but fixed deterministic algorithm for LWDPA. We show that for any combination of $k_{\min}$ and $k_{\max}$, there is a hard set of two input sequences for A that leads to the strict competitive ratio mentioned in the theorem. The hard input sequences we consider are derived from the worst-case scenarios depicted in Fig. 3. If $k_{\min} = 1$, we consider the input consisting of only one request of length 1 and the input consisting of this request and a second one of length $k_{\max}$ underneath it as depicted in Fig. 3a. The algorithm A can either accept the first request, leading to a strict competitive ratio of $k_{\max}$ on the second of these two inputs, or reject it, leading to a strict competitive ratio of $1/0$ on the first of the two inputs. Overall, A cannot have a strict competitive ratio better than $k_{\max}$.

Similarly, we can construct hard input sequences for the case that $k_{\min} \geq 2$ and $k_{\min} + 1 < k_{\max} < k_{\min}^2/4$ according to the scenario depicted in Fig. 3b, and those for the remaining case according to Fig. 3c.

In all three cases, we choose the length of the path just long enough for all requests to fit in; this is $\ell := k_{\max}$ in the first case, $\ell := 2k_{\max} + k_{\min}$ in the second, and $\ell := 2k_{\max} + k_{\min} - 2$ in the third case. $\square$

3.3 Trade-Off

Having covered the two extreme cases, classic online algorithms and optimal online algorithms with advice, we now we now strive to find the best possible behavior in between, balancing out advice complexity and strict competitiveness. Theorem 6 proves an upper bound by describing a class-based greedy algorithm. It is a great choice when only few advice bits are given, but less so with increasing advice complexity. Corollary 2 shows that it cannot undercut a strict competitive ratio logarithmic in $\varphi := k_{\max}/k_{\min}$, even when reading arbitrarily many advice bits. This problem is remedied by Theorem 7, which incorporates the approach of Theorem 6, but also makes efficient use of ample available advice by performing a sort of preprocessing.

Finally, we prove two lower bounds. Theorem 8 is evidently a counterpart to Theorem 6, with the same order of magnitude for a constant number of advice bits. Theorem 9, on the other hand, states a very versatile, albeit quite technical result. The two Corollaries 5 and 6 bring it in a more applicable form. The first one improves on the lower bound of Theorem 8 in the case of a strict competitive ratio logarithmic in φ; the second one yields the missing lower bound for even larger advice complexities, complementing Theorem 7 and Corollary 3.

Theorem 6. *There is an online algorithm reading b bits of advice that has a strict competitive ratio of $c \leq 2^b \cdot \left(2 \cdot \sqrt[2^b]{\varphi} + 1\right)$.*

Proof. The following is a variation of the randomized strategy *Classify and Randomly Select* for DPA, first proposed by Awerbuch et al. [1], adapted to LWDPA in our advice setting.

We divide the requests of the input sequence I into 2^b classes according to their lengths. Let class C_i contain all requests of length k_i with

$$k_{\min} \cdot \sqrt[2^b]{\varphi^{i-1}} \leq k_i < k_{\min} \cdot \sqrt[2^b]{\varphi^{i}}$$

for all i with $1 \leq i \leq 2^b$, and let C_{2^b} additionally contain all requests of length $k_{\max}$. This way, each request is contained in exactly one class since $k_{\min} \cdot \sqrt[2^b]{\varphi^0} = k_{\min}$ and $k_{\min} \cdot \sqrt[2^b]{\varphi^{2^b}} = k_{\min} \cdot \varphi = k_{\max}$. Furthermore, let A_i be the deterministic algorithm that accepts all requests from class C_i greedily and rejects all others. The oracle specifies the algorithm $A^* \in \{A_1, A_2, \ldots, A_{2^b}\}$ that has the largest gain on I among all algorithms A_i. Let us denote this maximum gain by $a^* := \text{gain}(A^*(I))$. To specify A^*, the oracle uses b advice bits.

Now let us consider some arbitrary but fixed optimal solution $\text{OPT}(I)$. Plugging $k_1 := k_{\min} \cdot \sqrt[2^b]{\varphi^{i-1}}$ and $k_2 := k_{\min} \cdot \sqrt[2^b]{\varphi^{i}}$ into Corollary 1, we can deduce that for all i, $\text{OPT}(I)$ has a gain of at most

$$a^* \cdot \left(\frac{2k_2}{k_1} + 1\right) = a^* \cdot \left(\frac{2k_{\min} \cdot \sqrt[2^b]{\varphi^{i}}}{k_{\min} \cdot \sqrt[2^b]{\varphi^{i-1}}} + 1\right) = a^* \cdot \left(2 \cdot \sqrt[2^b]{\varphi} + 1\right)$$

on the requests from C_i.

As the overall gain of $\text{OPT}(I)$ can be at most the sum of the gains on all 2^b classes, we obtain

$$\text{gain}(\text{OPT}(I)) \leq 2^b \cdot a^* \cdot \left(2 \cdot \sqrt[2^b]{\varphi} + 1\right),$$

and hence the strict competitive ratio of A^* on I is

$$\frac{\text{gain}(\text{OPT}(I))}{\text{gain}(A^*(I))} \leq \frac{2^b \cdot a^* \cdot \left(2 \cdot \sqrt[2^b]{\varphi} + 1\right)}{a^*} = 2^b \cdot \left(2 \cdot \sqrt[2^b]{\varphi} + 1\right).$$

□

As mentioned above, the algorithm of Theorem 6 is most efficient for little advice. For no advice at all, $b = 0$, we get $c \leq 2\varphi + 1$; this coincides with the upper bound that we get from Corollary 1 for the adviceless greedy algorithm

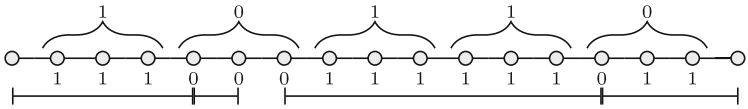


Fig. 4. An example with path length $\ell = 16$. The requests are those of the fixed optimal solution on the bases of which the advice is given. There are $\ell - 1 = 15$ inner vertices, gathered into groups of size s. The top part of the figure gives the advice string that results from groups of size $s = 3$. Underneath, the advice string for $s = 1$ is given. From this string, we can easily deduce the advice string for general $s \geq 1$ by conjoining the bits groupwise.

on the complete input sequence I by plugging in $k_1 := k_{\min}$ and $k_2 := k_{\max}$. For a single bit of advice, $b = 1$, the bound reads $c \leq 2 \cdot (2\sqrt{\varphi} + 1)$. This is a marked improvement for large φ, whose influence is dampened significantly. We pay with an additional factor of $2^b = 2$, however. This problem exacerbates quickly with increasing advice complexity. In fact, there is a threshold beyond which more advice even worsens the bound. The standard method for finding extrema reveals it to be $b = \log(\ln \varphi/(1 + W(1/(2\mathrm{e}))))$, where $W(x)$ is the Lambert W function. For this value, Theorem 6 yields its lowest upper bound on the strict competitive ratio: There is a strictly $(4.41 \cdot \log \varphi)$-competitive online algorithm that reads $\log \log \varphi - 0.73$ advice bits. The following corollary provides a more tangible bound in the same order of magnitude.

Corollary 2. *There is a strictly* $(5 \log \varphi)$*-competitive online algorithm that reads* $\log \log \varphi$ *advice bits.*

Suppose now that a rather large amount of advice is available, φ or $\ell/2$ bits for example. Only a tiny fraction of advice can be put to good use in the approach of Theorem 6; most of it would be wasted on the algorithm. Theorem 7 shows how to use this otherwise superfluous advice in a kind of preprocessing. The underlying idea is that the oracle first gives information about transition points, similarly to the optimal algorithm of Theorem 2. If the advice does not suffice to communicate their locations exactly, we reveal them only approximately in a way that is useful nonetheless. In essence, this first bulk of advice allows us to predict all long optimal requests with reasonable accuracy and ensure that enough of them will be accepted. The remaining advice can be spent in the spirit of Theorem 6 on the remaining requests, with a virtual upper bound of $2s - 1$ taking the place of $k_{\max}$.

Theorem 7. *For any* $s \geq 1$ *and any* $b \geq 0$*, there is an online algorithm that is strictly* $\max\{6 - 8/(s+1), 2^b(2 \cdot \sqrt[2^b]{(2s-1)/k_{\min}} + 1)\}$*-competitive and uses* $(\ell - 1)/s + b$ *advice bits.*

Proof. As usual, the oracle fixes an optimal solution; let $r_1, \ldots, r_k$ be its requests. Any request r_i is a subpath with two leaves plus a set R_i of inner vertices. This set is empty if and only if the request has length 1. Since two requests $r_i \neq r_j$

of a solution cannot overlap, any two sets $R_i \neq R_j$ are separated by at least one vertex; we will use this below. The given path of length ℓ contains exactly $\ell - 1$ inner vertices. We gather these into groups G_i, each of size s, as illustrated in Fig. 4 for $s = 3$. For every group G_i, the oracle communicates to the algorithm by a single bit b_i whether the group's vertices are enclosed within a single optimal request, that is, $b_i = 1 \iff \exists j \in \{1, \ldots, k\} \colon G_i \subseteq R_j$. This requires one bit per group, hence $(\ell - 1)/s$ bits in total.

How does this information help us? We begin with the case $s = 1$. Here, the advice reveals the exact location of all vertices of all R_i. Remember that R_i is the set of inner vertices of r_i. Since these sets are pairwise separated, we can directly infer the exact position of all requests r_i containing any inner vertex, that is, all optimal requests of length 2 or greater. Hence, the algorithm can *reserve* these subpaths and accept the fitting requests whenever they appear. The requests of length 1, in contrast, need to be accepted exactly if the *reservations* allow for it. Indeed, either they are blocked by longer optimal requests and can therefore not be part of the chosen solution; or they can and must be included in the solution by its optimality. The algorithm will thus reconstruct an optimal solution, using only $\ell - 1$ advice bits.

Now to the general case $s \geq 1$. The advice bit of a group G_i is still 1 if and only if there is a request r_j that encloses all its vertices, that is, $G_i \subseteq R_j$. If this is true for two neighboring groups, they must be contained within the same request, that is, $G_i, G_{i+1} \subseteq R_j$, since the R_j are pairwise disjoint. The groups are also disjoint, so $|G_i| + |G_{i+1}| \leq |R_j|$. By the same reasoning, we see that any substring B of the advice string that consists of $|B|$ consecutive ones stands for $|B|$ groups whose $|B| \cdot s$ vertices are inner vertices of a single optimal request r_j; hence $|B| \cdot s \leq |R_j|$.

Denote the set of all maximal substrings of ones by $\mathcal{B}$. For any $B \in \mathcal{B}$, we can state an upper limit of $|R_j| \leq |B|s + 2(s - 1)$. Otherwise, the substring would not be maximal: The upper bound is attained if and only if R_j overlaps the vertices of the groups corresponding to B by $s - 1$ vertices on each side. If the overlap were any larger on either side, then the respective neighboring bit of the substring B would also have been set to 1 instead of 0, contradicting the maximality.

This insight about R_j translates directly to r_j, the optimal request associated with B; just add the two leaves: For the upper limit, r_j covers exactly $|B| + 2$ groups, those of B plus one additional on each side; its length is therefore $(|B| + 2)s - 1$. The lower limit is attained if r_j only contains the two leaves in addition to the inner vertices of the $|B|$ groups. In this case, the length of r_j is $|B|s + 1$. Thus, the optimal solution's gain associated with block B stays in the range $[|B|s + 1, (|B| + 2)s - 1]$.

Now, we might hope for our algorithm to accept at least one request within the described boundaries for each substring; this would result in a lower bound of $\sum_{B \in \mathcal{B}} |B|s + 1 \leq \text{gain}'(\mathcal{A}(I))$, where gain$'$ denotes the gain from requests that cover at least one group. Since there exists at least one fitting request for each block, namely r_j, the algorithm could just wait for the first one. Due to

the mentioned potential overlap on either side, it may happen, however, that an accepted request for one substring blocks all suited requests of the next substring. In fact, this is only possible for two neighboring substrings that are separated by a single 0. This might be the case for all substrings, however. So we forego our unrealistic hopes on every second substring $B \in \mathcal{B}$ and focus on the others. For those, we can now reserve subpaths of size $(|B|+2)s-1$, large enough to ensure an accepted request within the given range.

Among the two possible choices of alternate substrings of ones, the algorithm picks that with the larger grand total of ones. This way, the actual lower bound gets cut down to no less than half of what we have calculated above,

$$\frac{1}{2}\sum_{B\in\mathcal{B}} |B|s+1 \le \mathrm{gain}'(\mathcal{A}(I)).$$

For the optimal solution, we have $\mathrm{gain}'(\textsc{Opt}(I)) \le \sum_{B\in\mathcal{B}}((|B|+2)s-1)$. The strict competitive ratio on the reserved subpaths can therefore be bounded by 6:

$$\frac{\mathrm{gain}'(\textsc{Opt}(I))}{\mathrm{gain}'(\mathcal{A}(I))} \le 2 + 2\frac{\sum_{B\in\mathcal{B}}\Big(2s-2\Big)}{\sum_{B\in\mathcal{B}}\Big(|B|s+1\Big)} \le 2 + 2\tfrac{2s+2-4}{s+1} = 6 - \frac{8}{s+1}.$$

We have not yet accounted for the gain on the unreserved subpaths. For this part, all optimal requests went undetected by any advice bit. Hence, they have a length of $2s-1$ or less, as seen by setting $|B|=0$ in the range established above. The algorithm can thus greedily accept all requests with lengths up to $2s-1$ and be strictly $(2s-1)$-competitive on this remaining part. Employing the more sophisticated greedy approach of Theorem 6, we achieve a strict competitive ratio of $2^b(2\cdot\sqrt[2^b]{(2s-1)/k_{\min}}+1)$ by granting b additional advice bits. This way, the algorithm uses $(\ell-1)/s+b$ advice bits overall and guarantees

$$\frac{\mathrm{gain}(\textsc{Opt}(I))}{\mathrm{gain}(\mathcal{A}(I))} \le \max\Big\{6-\tfrac{8}{s+1}, 2^b\Big(2\cdot\sqrt[2^b]{(2s-1)/k_{\min}}+1\Big)\Big\}. \qquad \square$$

The following corollary extracts a more accessible result from Theorem 7 that will later be complemented by Corollary 6.

Corollary 3. *For any constant $k \ge 6$, the number of advice bits sufficient to achieve a strict competitive ratio of $c \le k$ is at most*

$$2\cdot\frac{\ell-1}{2^{k/5}\cdot k_{\min}+1} + \log\left(\frac{k}{5}\right) \in \mathcal{O}\left(\frac{\ell}{k_{\min}}\right).$$

Proof. We plug the values $s := (2^{k/5}\cdot k_{\min}+1)/2$ and $b := \log(k/5)$ into Theorem 7 and obtain that there exists an algorithm with a strict competitive ratio of at most

$$c = \max\left\{6-\frac{8}{s+1}, \tfrac{k}{5}\Big(2\cdot\sqrt[k/5]{2^{k/5}}+1\Big)\right\} \le \max\left\{6, \tfrac{k}{5}(2\cdot 2+1)\right\} = k$$

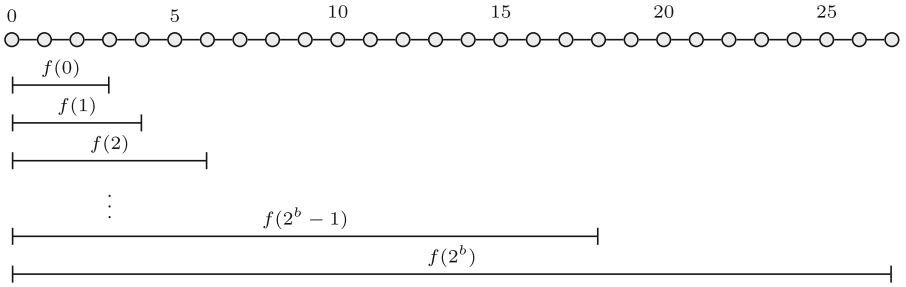


Fig. 5. The $2^b + 1$ requests $r_0, \ldots, r_{2^b}$ of $I_{2^b} \in \mathcal{I}$.

and an advice complexity of

$$b = \frac{(\ell - 1) \cdot 2}{2^{k/5} \cdot k_{\min} + 1} + \log\left(\frac{k}{5}\right) \in \mathcal{O}\left(\frac{\ell}{k_{\min}}\right). \qquad \square$$

The following lower bound is a good match for Theorem 6 in the case of a constant number of advice bits.

Theorem 8. *No online algorithm reading at most b advice bits can have a strict competitive ratio better than $\sqrt[2^b]{\varphi}$.*

Proof. Let $f(i) := k_{\min} \cdot \sqrt[2^b]{\varphi^i}$ for all i with $0 \leq i \leq 2^b$. Note that $f(0) = k_{\min}$ and $f(2^b) = \varphi \cdot k_{\min} = k_{\max}$. For any number b of advice bits, we consider the following set $\mathcal{I} = \{I_0, \ldots, I_{2^b}\}$ of instances. Instance I_{2^b} presents $2^b + 1$ left-aligned requests $r_0, \ldots, r_{2^b}$ of increasing length $f(i)$, as illustrated in Fig. 5. Instance $I_i \in \mathcal{I}$ presents only the first $i + 1$ of these requests, namely $r_0, \ldots, r_i$.

With b advice bits, we only have 2^b deterministic algorithms available, and thus at least two out of the $2^b + 1$ instances from $\mathcal{I}$ must be processed by the same deterministic algorithm. Hence, let A denote a deterministic algorithm that processes at least two instances from $\mathcal{I}$ and let I_i and I_j with $i < j$ be two of the instances processed by A. The gain of A on any instance I_s is

$$\text{gain}(A(I_s)) = \begin{cases} f(t) & \text{if } A \text{ accepts a request } r_t \text{ with } t \leq s, \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

The optimal gain for any I_s is $f(s)$, of course. The two instances I_i and I_j are identical for the first $i+1$ requests; hence, A must treat them the same way up to this point. We distinguish two cases and show that A performs badly on at least one of the instances I_i or I_j in both cases:

First, if A does not accept a request $r_t \in \{r_0, \ldots, r_i\}$, then its gain on I_i is $\text{gain}(A(I_i)) = 0$ and the strict competitive ratio is unbounded. Second, if A does accept a request $r_t \in \{r_0, \ldots, r_i\}$, then its gain on I_j is $\text{gain}(A(I_j)) = f(t) \leq f(i)$, which leads to a strict competitive ratio of

$$c = \frac{f(j)}{f(t)} \geq \frac{f(j)}{f(i)} \geq \frac{f(i+1)}{f(i)} = \frac{\sqrt[2^b]{\varphi^{i+1}} \cdot k_{\min}}{\sqrt[2^b]{\varphi^i} \cdot k_{\min}} = \sqrt[2^b]{\varphi}.$$

This means that no matter which algorithm A is used for two instances from $\mathcal{I}$, on one of these two instances the strict competitive ratio cannot be better than $\sqrt[2^b]{\varphi}$. □

The following proof of Theorem 9 adapts a method from Gebauer et al. [8] that has been used to analyze the unweighted and non-parametrized disjoint path allocation problem on paths. First, we need a result that is proven in the mentioned paper. It is a slight variant of a bound for the tail of the hypergeometric distribution.

Fact 1 (Corollary 1 from Gebauer et al. [8]). *Let X be a discrete random variable with a hypergeometric distribution with parameters M, N, and n, and let $t \geq 0$. Then, for every $M' \leq M$, we have*

$$\Pr\left(X \leq n \cdot \frac{M'}{N} - tn\right) \leq \mathrm{e}^{-2t^2 n}\,.$$

We construct a set $\mathcal{I}$ of instances which is hard for many deterministic algorithms at once and thereby show that many deterministic algorithms are necessary to achieve a good strict competitive ratio on all instances from $\mathcal{I}$. For the time being, we choose three parameters x, y, and z with $x, y, z \in \mathbb{N}_{\geq 1}$ and construct the set $\mathcal{I}$ subject to those parameters to obtain a result as general as possible. The parameters are chosen in such a way that the following requirements are satisfied.

$$x \leq \log \varphi, \tag{1}$$
$$y \leq \log\left(\ell / k_{\min}\right) - 2x, \quad \text{and} \tag{2}$$
$$z \leq \sqrt{2^y / \ln x}. \tag{3}$$

The general result we obtain eventually, depending on the variables x and z, is the following.

Theorem 9. *Let $x, z \in \mathbb{N}_{\geq 1}$. For any $x \leq \log \varphi$ and $z \leq \sqrt{\ell/(k_{\min} \cdot 4^x \cdot \ln x)}$, any online algorithm for* LWDPA *needs to read at least*

$$b = \frac{\ell}{k_{\min}} \cdot \frac{\log \mathrm{e}}{z^2 \cdot 4^x} - \log x$$

advice bits to obtain a strict competitive ratio of

$$c \leq \frac{x+2}{2 \cdot \left(1 + \frac{x}{z}\right)}.$$

Proving this will be the goal of our subsequent considerations. Afterwards, we will also see how to choose reasonable values for the parameters to get less general but more expressive statements. More specifically, we will derive two corollaries from Theorem 9, which will give us almost matching lower bounds to two of the upper bounds proven above. Before we prove Theorem 9, however, we will have to make the necessary arrangements.

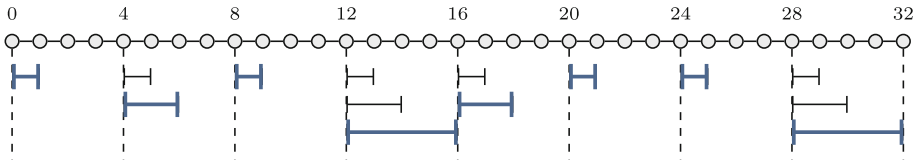


Fig. 6. An example for $\ell = 32$, $k_{\min} = 1$, $x = 2$, and $y = 1$. The dashed lines indicate the segments. There are $2^{x+y} = 8$ segments of width $2^2 \cdot k_{\min} = 4$ each, and requests appear in $x + 1 = 3$ phases; 8 requests of length 1 in phase 0, then 4 of length 2 in phase 1, and 2 of length 4 in phase 2. Good requests are marked by thick lines.

Let us start with a description of how to construct the hard instance set $\mathcal{I}$. In each instance, requests are presented in $x + 1$ phases, starting with phase 0. We divide the path into 2^{x+y} segments of width $2^x \cdot k_{\min}$ each, and all requests presented during the whole computation are always aligned with the left border of the respective segment. Phase 0 contains 2^{x+y} requests of length $2^0 \cdot k_{\min}$ each, one in each segment. In each phase i with $1 \leq i \leq x$, we choose exactly half of the segments that contained requests in phase $i-1$ and present a request of twice the size in those segments. Let us call a request *good* if it has no further requests underneath and *bad* otherwise. Figure 6 shows an example.

Let us define $h := x + y$. We make the following observation.

Observation 1. *For each i with $0 \leq i \leq x$, phase i contains 2^{h-i} requests of length $2^i \cdot k_{\min}$ each.*

We have to check two points concerning the presentation of requests: First, whether the path is long enough to hold all segments, and second, whether all requests have lengths between $k_{\min}$ and $k_{\max}$. Our requirement (2) ensures the first point, as

$$2^{x+y} \cdot 2^x \cdot k_{\min} = 2^{2x+y} \cdot k_{\min} \leq 2^{\log\left(\frac{\ell}{k_{\min}}\right)} \cdot k_{\min} = \ell.$$

Furthermore, according to Observation 1, all requests are between $2^0 \cdot k_{\min}$ and $2^x \cdot k_{\min}$ in length. So all requests are large enough, and requirement (1) ensures that no requests are larger than $k_{\max}$, since

$$2^x \cdot k_{\min} \leq 2^{\log \varphi} \cdot k_{\min} = \varphi \cdot k_{\min} = k_{\max}.$$

For all instances constructed in the way described before, the gain of their respective optimal solutions is the same, as stated in the following observation.

Observation 2. *For each instance $I \in \mathcal{I}$, the gain of the optimal solution is*

$$\text{gain}(\textsc{Opt}(I)) = (x + 2) \cdot k_{\min} \cdot 2^{h-1}.$$

Proof. The optimal solution contains exactly all good requests. For each phase, these are exactly half of the requests presented, except for the last phase, in

which all requests are good and hence contained in the optimal solution. Thus, using Observation 1, the gain of the optimal solution on each instance I is

$$\text{gain}(\textsc{Opt}(I)) = \left(\sum_{i=0}^{x-1} \frac{2^{h-i} \cdot 2^i k_{\min}}{2}\right) + 2^{h-x} \cdot 2^x k_{\min} = (x+2)k_{\min} \cdot 2^{h-1}.$$

□

The set $\mathcal{I}$ can be represented by a tree in which each vertex corresponds to a subset of instances. The depth of this tree is x (i.e., it has $x+1$ levels); the root on level 0 represents all instances from $\mathcal{I}$, and every vertex on level i represents the set of all instances with the same particular set of requests in phases $0, \ldots, i$. Every leaf is located on level x and represents a single instance from $\mathcal{I}$. The same tree representation of $\mathcal{I}$ has also been used by Gebauer et al. [8]. However, it is worth mentioning that in our case, the vertices of the tree have a different number of children since we construct the instances from $\mathcal{I}$ in a different way.

Consider some vertex v on level i and the corresponding set of instances $\mathcal{I}_v$. Now consider any deterministic algorithm A, given an arbitrary instance from $\mathcal{I}_v$ as input. Then A is in the same state during the whole computation up to and including phase i, independently of the exact instance from $\mathcal{I}_v$ that A is given, meaning that it sees and accepts the exact same requests up until the end of phase i.

From now on, let A be arbitrary but fixed. For a given vertex v on level i, let $a(i)$ be the number of accepted requests of A on any instance from $\mathcal{I}_v$ during phase i, and let $g(i)$ be the gain of A achieved during phase i. Obviously, due to Observation 1,

$$g(i) = a(i) \cdot 2^i \cdot k_{\min}. \tag{4}$$

Moreover, let us define $\hat{g}(i)$ to be the gain that A achieves in all phases up to and including phase i and, in accordance with this definition, $\hat{g}(-1)$ to be the total gain of requests accepted before the start of computation. Hence,

$$\hat{g}(i) := \sum_{j=0}^{i} g(j) \quad \text{and} \quad \hat{g}(-1) := 0. \tag{5}$$

Now let us define the sets of requests $B(i)$ and $B^+(i)$ as follows. Let $B(i)$ be the set of requests from phase i that are already blocked at the beginning of phase i. Let $B^+(i)$ be the set of requests from phase i that are blocked at the beginning of phase $i+1$, including those that were accepted during phase i and are now blocking themselves.

We call a phase i *bad* for A if $B(i)$ contains at least

$$d_i := \frac{1}{2^i} \cdot \left(\frac{\hat{g}(i-1)}{k_{\min}} - \frac{i}{z} \cdot 2^h\right) \tag{6}$$

requests and *good* otherwise. Moreover, let us call a vertex v on level i *bad* for A if, when given an instance from $\mathcal{I}_v$ as its input, phase i is bad for A, and

otherwise let us call v *good* for A. Since each instance corresponds to a vertex on level $x+1$, an instance is bad for A if and only if phase $x+1$ is bad for A.

Observation 3. *If at least d_{i+1} requests from $B^+(i)$ are bad, then phase $i+1$ is bad for A.*

Proof. As defined before, $B^+(i)$ is the set of requests from phase i that are blocked at the beginning of phase $i+1$. If at least d_{i+1} requests of those are bad, then at least d_{i+1} requests are presented in phase $i+1$ that are already blocked at the beginning of phase $i+1$; this matches the definition of a bad phase. □

Lemma 2. *For each bad vertex, the fraction of bad vertices among its children is at least*

$$1 - \mathrm{e}^{-2^y/z^2}.$$

Proof. Let v be a bad vertex on level i. Since v is bad, phase i must be bad for A when A is given any instance from $\mathcal{I}_v$ as input. Thus, $B(i)$ must contain at least d_i requests. As A accepts $a(i)$ requests in phase i, the set $B^+(i)$ contains at least $d_i + a(i)$ requests. According to Observation 3, if at least d_{i+1} requests from $B^+(i)$ are bad, phase $i+1$ is bad. Hence, a sufficient condition for a child w of v to be bad is that at least d_{i+1} requests from $B^+(i)$ are bad when A is given an instance from $\mathcal{I}_w$ as input. Thus, we have the following scenario. There are $N := 2^{h-i}$ requests in phase i; out of these, $M \geq M' := d_i + a(i)$ are blocked at the beginning of phase $i+1$ and hence contained in $B^+(i)$. Each child w of v corresponds to the set of instances in which the same set of $n := 2^{h-i-1}$ requests from phase i are bad. This in turn corresponds to the following. We have an urn containing N balls, out of which $M \geq M'$ are black; we draw n balls without replacement, and we are interested in the probability that the number of black balls drawn is at least d_{i+1}.

Let X_i be the random variable counting the number of bad requests from $B^+(i)$ in this scenario. X_i has a hypergeometric distribution with parameters M, N, and n, and we are interested in $\Pr(X_i \geq d_{i+1})$.

We can bound the probability

$$\Pr\left(X_i \leq n \cdot \frac{M'}{N} - tn\right) = \Pr\left(X_i \leq \tfrac{d_i + a(i) - t \cdot 2^{h-i}}{2}\right)$$

from above for any $t \geq 0$ by using Fact 1. We obtain

$$\Pr\left(X_i \leq \frac{d_i + a(i) - t \cdot 2^{h-i}}{2}\right) \leq \mathrm{e}^{-2t^2 \cdot n} = \mathrm{e}^{-2t^2 \cdot 2^{h-i-1}} = \mathrm{e}^{-t^2 \cdot 2^{h-i}}. \tag{7}$$

Using (5) and (6), we get

$$d_{i+1} = \frac{1}{2^{i+1}}\left(\frac{\hat{g}(i)}{k_{\min}} - \frac{i+1}{z} \cdot 2^h\right) = \frac{1}{2^{i+1}}\left(\frac{\hat{g}(i-1)}{k_{\min}} + \frac{g(i)}{k_{\min}} - \frac{i}{z} \cdot 2^h - \frac{1}{z} \cdot 2^h\right).$$

Doing further transformations using (4) and again (6), we obtain

$$d_{i+1} = \frac{\frac{1}{2^i} \cdot \left(\frac{\hat{g}(i-1)}{k_{\min}} - \frac{i}{z} \cdot 2^h + 2^i \cdot a(i) - \frac{1}{z} \cdot 2^h \right)}{2} = \frac{d_i + a(i) - \frac{1}{z} \cdot 2^{h-i}}{2}.$$

Combining this result with (7) and choosing $t := 1/z$ yields

$$\Pr(X_i \geq d_{i+1}) \geq 1 - \Pr\left(X_i \leq \frac{d_i + a(i) - \frac{1}{z} \cdot 2^{h-i}}{2} \right) \geq 1 - \mathrm{e}^{-2^{h-i}/z^2},$$

and since $h - i \geq y$ for all i with $0 \leq i \leq x$, we finally obtain that for all such i,

$$\Pr(X_i \geq d_{i+1}) \geq 1 - \mathrm{e}^{-2^y/z^2}. \qquad \square$$

From the fraction of bad vertices among the children of bad vertices, we can now draw conclusions concerning the fraction of bad vertices on each level.

Lemma 3. *The fraction of bad vertices on level i is at least*

$$\left(1 - \mathrm{e}^{-2^y/z^2}\right)^i.$$

Proof. We prove the claim by induction on i. For the base case, we consider the root, the only vertex on level 0. The root is bad if phase 0 is bad for A, and phase 0 is bad for A if at least d_0 requests from phase 0 are already blocked at the beginning of phase 0. This is obviously the case as $d_0 = 0$ according to (5) and (6). Hence, the root is a bad vertex and thus the fraction of bad vertices on level 0 is 1, which is at least

$$\left(1 - \mathrm{e}^{-2^y/z^2}\right)^0 = 1.$$

For the induction step, let us assume that the claim holds for level $i-1$. Let us define $v(i)$ to be the number of all vertices on level i and $b(i)$ the number of bad vertices among those. Moreover, let $c(i)$ denote the number of children of each vertex on level i. As a final definition, let $f(i)$ be the fraction of bad vertices among the children of each bad vertex on level i. Lemma 2 states that $f(i) \geq 1 - \mathrm{e}^{-2^y/z^2}$ for all i, and by the induction hypothesis it holds that $b(i-1)/v(i-1) \geq (1 - \mathrm{e}^{-2^y/z^2})^{i-1}$. Using Lemma 2, the induction hypothesis, and the fact that $b(i) \geq b(i-1) \cdot c(i-1) \cdot f(i-1)$ and $v(i) = v(i-1) \cdot c(i-1)$, the fraction $b(i)/v(i)$ of bad vertices on level i can be bounded from below by

$$\frac{b(i-1) \cdot f(i-1)}{v(i-1)} \geq \left(1 - \mathrm{e}^{-2^y/z^2}\right)^{i-1} \cdot \left(1 - \mathrm{e}^{-2^y/z^2}\right) = \left(1 - \mathrm{e}^{-2^y/z^2}\right)^i. \qquad \square$$

From this we can conclude that for each deterministic online algorithm, a large number of instances must be bad.

Corollary 4. *For any deterministic online algorithm A, the fraction of instances of $\mathcal{I}$ which are bad for A is at least $(1 - \mathrm{e}^{-2^y/z^2})^x$.*

Proof. This follows directly from Lemma 3 and the fact that each single instance from $\mathcal{I}$ corresponds to a vertex on level x. □

Now we want to show that any deterministic algorithm can only accept few requests on each bad instance.

Lemma 4. *Let A be an arbitrary but fixed deterministic online algorithm for* LWDPA*, and let $I \in \mathcal{I}$ be a bad instance for A. Then the gain of A on I is*

$$\mathrm{gain}(A(I)) = \hat{g}(x) \leq 2^h \cdot k_{\min} \cdot \left(1 + \frac{x}{z}\right).$$

Proof. According to the definition of bad vertices and instances, an instance (corresponding to a vertex on level x) is bad if there are at least d_x requests from phase x blocked at the beginning of phase x. In this phase, A is presented 2^{h-x} requests of length $2^x \cdot k_{\min}$ due to Observation 1, so the gain of A in phase x is

$$\begin{aligned} g(x) &\leq 2^x \cdot k_{\min} \cdot \left(2^{h-x} - d_x\right) \\ &= 2^x \cdot k_{\min} \cdot \left(\frac{2^h}{2^x} + \frac{1}{2^x} \cdot \left(\frac{x}{z} \cdot 2^h - \frac{\hat{g}(x-1)}{k_{\min}}\right)\right) \\ &= 2^h \cdot k_{\min} \cdot \left(1 + \frac{x}{z}\right) - \hat{g}(x-1). \end{aligned}$$

For the total gain at the end of A's computation, we obtain

$$\hat{g}(x) = g(x) + \hat{g}(x-1) \leq 2^h \cdot k_{\min} \cdot \left(1 + \frac{x}{z}\right).$$

□

Now we can bound the strict competitive ratio of A on any bad instance from below.

Lemma 5. *The strict competitive ratio of any deterministic online algorithm on any bad instance from $\mathcal{I}$ is*

$$c \geq \frac{x+2}{2 \cdot \left(1 + \frac{x}{z}\right)}.$$

Proof. Let A be some arbitrary but fixed deterministic online algorithm for LWDPA and let $I \in \mathcal{I}$ be a bad instance for A. Using Lemma 4 and Observation 2, we obtain

$$c = \frac{\mathrm{gain}(\textsc{Opt}(I))}{\mathrm{gain}(A(I))} \geq \frac{(x+2) \cdot k_{\min} \cdot 2^{h-1}}{2^h \cdot k_{\min} \cdot \left(1 + \frac{x}{z}\right)} = \frac{x+2}{2 \cdot \left(1 + \frac{x}{z}\right)}.$$

□

From this we can conclude that any online algorithm for LWDPA needs to read a large number of advice bits to obtain a good strict competitive ratio. Hence, we are finally ready to prove Theorem 9.

Proof (of Theorem 9). We interpret an online algorithm that reads b advice bits in the usual way as a set of deterministic algorithms $\{A_1, \ldots, A_{2^b}\}$. According to Lemma 5, each such algorithm A_i can have a strict competitive ratio of at most c only on instances which are good, and according to Corollary 4, for each A_i the fraction of good instances from $\mathcal{I}$ is at most

$$1 - \left(1 - \mathrm{e}^{-2^y/z^2}\right)^x \leq \frac{x}{\mathrm{e}^{2^y/z^2}}, \tag{8}$$

where we used Bernoulli's Inequality. Requirement (3) from the very beginning makes sure that (8) always yields a reasonable result, i.e., that the right-hand side of (8) never becomes larger than 1, since

$$\frac{x}{\mathrm{e}^{2^y/z^2}} \leq \frac{x}{\mathrm{e}^{2^y/(2^y/\ln x)}} = \frac{x}{\mathrm{e}^{\ln x}} = 1.$$

Thus, the number of deterministic algorithms that are necessary to guarantee a strict competitive ratio of c on all instances from $\mathcal{I}$ is at least $(\mathrm{e}^{2^y/z^2})/x$. Hence, the number of necessary advice bits is

$$b \geq \log\left(\frac{\mathrm{e}^{2^y/z^2}}{x}\right) = \frac{2^y}{z^2} \cdot \log \mathrm{e} - \log x.$$

Now we are almost done. As our final step, we observe that the variable y that appears in the formula for b does not have any influence on c. Since the number of necessary advice bits grows with y, we can set it to the maximum possible value according to requirement (2). Thus, we set $y := \log(\ell/k_{\min}) - 2x = \log\left(\ell/\left(k_{\min} \cdot 2^{2x}\right)\right)$, and as a result, we finally obtain that the number of necessary advice bits is

$$b \geq \frac{2^{\log\left(\frac{\ell}{k_{\min} \cdot 4^x}\right)}}{z^2} \cdot \log \mathrm{e} - \log x = \frac{\ell}{k_{\min}} \cdot \frac{\log \mathrm{e}}{z^2 \cdot 4^x} - \log x.$$

Plugging the value that we just fixed for y into requirement (3) yields the range of $z \leq \sqrt{\ell/(k_{\min} \cdot 4^x \cdot \ln x)}$ given in the theorem. □

By plugging in values for x and z into this theorem, we can derive two results which nicely complement Corollaries 2 and 3, respectively. The first one states that many advice bits are necessary to obtain a strict competitive ratio better than $1/4 \cdot \log \varphi$.

Corollary 5. *Let δ be an arbitrary constant with $0 < \delta < 1/2$ and let $\varepsilon > 2\delta$. Any online algorithm for* LWDPA *that achieves a strict competitive ratio of $\delta/2 \cdot \log \varphi$ needs to read at least $\omega\left(\varphi^{1-\varepsilon} \cdot \ell/k_{\max}\right)$ advice bits when $\varphi \in \omega(1)$.*

Proof. Let $x := \delta \log(k_{\max}/k_{\min}) = \delta \log \varphi$ and $z := \log^2 \varphi$. Then, $4^x = 2^{2x} = 2^{2\delta \log \varphi} = \varphi^{2\delta}$. We obtain

$$\frac{\frac{\ell}{k_{\min}}}{z^2 \cdot 4^x} = \frac{\frac{\ell}{k_{\max}} \cdot \frac{k_{\max}}{k_{\min}}}{\log^4 \varphi \cdot \varphi^{2\delta}} = \frac{\ell}{k_{\max}} \cdot \frac{\varphi}{\log^4 \varphi \cdot \varphi^{2\delta}} = \frac{\ell}{k_{\max}} \cdot \frac{\varphi^{1-2\delta}}{\log^4 \varphi}.$$

The number of advice bits necessary to achieve a strict competitive ratio of c is

$$b \geq \frac{\ell}{k_{\min}} \cdot \frac{\log \mathrm{e}}{z^2 \cdot 4^x} - \log x = \frac{\varphi^{1-2\delta}}{\log^4 \varphi} \cdot \frac{\ell}{k_{\max}} \cdot \log \mathrm{e} - \log \delta - \log \log \varphi.$$

For any constant $\varepsilon > 2\delta$, we have $(\varphi^{1-2\delta}/\log^4 \varphi) \in \omega\big(\varphi^{1-\varepsilon}\big)$ if $\varphi \in \omega(1)$. Therefore, the number of advice bits necessary to obtain a strict competitive ratio of c is in $\omega\big(\varphi^{1-\varepsilon} \cdot \ell/k_{\max}\big)$, for any constant $\varepsilon > 2\delta$. The value of c obtained by plugging in the values for x and z as chosen above is

$$c \geq \frac{x+2}{2(1+\frac{x}{z})} \geq \frac{\delta \log \varphi + 1}{2(1+\frac{\delta}{\log \varphi})} = \frac{(\delta \log \varphi + 1) \log \varphi}{2(\log \varphi + \delta)} = \frac{\delta \log \varphi}{2} \cdot \frac{\log \varphi + \frac{1}{\delta}}{\log \varphi + \delta}.$$

Using this together with $\delta < 1/\delta$, we can bound c from below by $(\delta \log \varphi)/2$. □

Combining this with Corollary 2, we observe a surprising fact. While $\log \log \varphi$ advice bits are sufficient to obtain a strict competitive ratio of $5 \log \varphi$, we need $\omega\big(\varphi^{1-\varepsilon} \cdot \ell/k_{\max}\big)$ advice bits to decrease the strict competitive ratio by only another constant factor. The next corollary complements Corollary 3, stating that many advice bits are necessary to obtain a constant strict competitive ratio. The two bounds match up to a constant factor.

Corollary 6. *Let $k_{\min} \in o(\ell)$. For any constant $k \leq (\log \varphi + 2)/4$, the number of advice bits necessary to achieve a strict competitive ratio of $c \leq k$ is at least*

$$\frac{\ell}{k_{\min}} \cdot \frac{\log \mathrm{e}}{(4k-2)^2 \cdot 4^{4k-2}} - \log(4k-2) \in \Omega\left(\frac{\ell}{k_{\min}}\right).$$

Proof. We set $x := z := 4k-2$ and plug these values into Theorem 9. To be able to apply this theorem, it must hold that $x \leq \log \varphi$ and $z \leq \sqrt{\ell/(k_{\min} \cdot 4^x \cdot \ln x)}$. The first requirement is trivially fulfilled as $x = 4k-2 \leq 4 \cdot (\log \varphi + 2)/4 - 2 = \log \varphi$. The second one is also satisfied since $k_{\min} \in o(\ell)$ and hence $\sqrt{\ell/(k_{\min} \cdot 4^x \cdot \ln x)} \in \omega(1)$, while $z \in \mathcal{O}(1)$. Due to Theorem 9, any algorithm needs to read at least

$$b \geq \frac{\ell}{k_{\min}} \cdot \frac{\log \mathrm{e}}{(4k-2)^2 \cdot 4^{4k-2}} - \log(4k-2) \in \Omega\left(\frac{\ell}{k_{\min}}\right)$$

advice bits to achieve a strict competitive ratio of

$$c \leq \frac{x+2}{2 \cdot (1+\frac{x}{z})} = \frac{4k}{2 \cdot (1+1)} = k.$$

□

Thus, combining Corollaries 3 and 6, we obtain that $\Theta(\ell/k_{\min})$ advice bits are necessary and sufficient to achieve any constant strict competitive ratio.

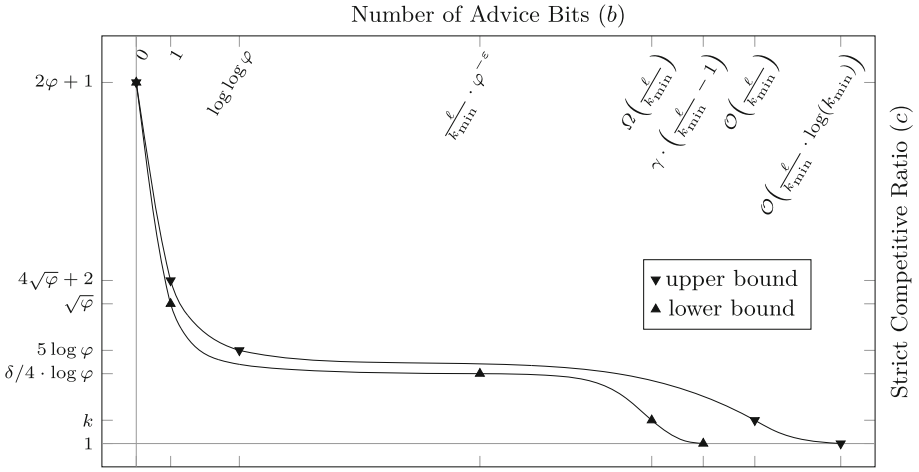


Fig. 7. A schematic plot of the results proven in this paper.

4 Conclusion

In this paper, we thoroughly studied the length-weighted disjoint path allocation problem on paths parametrized by given lower and upper bounds on the request lengths $k_{\min}$ and $k_{\max}$.

We examined the trade-off between the number of advice bits an online algorithm reads and the strict competitive ratio it can achieve. The strict competitive ratios we considered cover the entire possible range; from optimality to the worst-case strict competitive ratio when no advice is given at all. We described several algorithms for LWDPA and were able to give nearly or perfectly matching lower bounds in most cases. Our results imply a very interesting threshold behavior of the LWDPA problem.

As depicted schematically in Fig. 7, we showed the following. When an online algorithm for LWDPA is not given any advice at all, the strict competitive ratio cannot be better than $2\varphi + 1$, where $\varphi = k_{\max}/k_{\min}$, and a simple greedy approach achieves this ratio. With any constant number b of advice bits, the achievable strict competitive ratio lies between $\sqrt[2^b]{\varphi}$ and $2^b \cdot (2 \cdot \sqrt[2^b]{\varphi} + 1)$, which is matching up to a constant factor. When the algorithm is allowed to read $\log \log \varphi$ advice bits, a relatively good strict competitive ratio can be obtained already, namely $5 \log \varphi$. However, to decrease it further by only a constant factor to $\delta/4 \cdot \log \varphi$ for any $\delta < 1$, we showed that $\ell/k_{\min} \cdot \varphi^{-\varepsilon}$ advice bits are necessary, for any $\varepsilon > \delta$. We proved that $\Theta(\ell/k_{\min})$ advice bits are necessary and sufficient to achieve any constant strict competitive ratio c with $6 \leq c \leq (\log \varphi + 2)/4$. Finally, in order to be optimal, n or $(\ell - 1)/k_{\min} \cdot (1 + 2 \log k_{\min})$ advice bits are sufficient; the number of advice bits necessary to be optimal is at least $\gamma \cdot (n - 1)$ or $\gamma \cdot (\ell/k_{\min} - 1)$ for some constant γ that depends on both $k_{\max}$ and $k_{\min}$, but is always at least $1/2$ and approaches 1 with growing difference between $k_{\min}$

and $k_{\max}$. Thus, there is only a factor of $\mathcal{O}(\log k_{\min})$ between the lower and upper bounds for optimality in ℓ and a constant factor between those in n; moreover, in the unparametrized case, the bounds in ℓ are perfectly matching and those in n match up to an additive constant of 1.

Acknowledgements. We thank Hans-Joachim Böckenhauer and Dennis Komm for valuable discussions.

References

1. Awerbuch, B., Bartal, Y., Fiat, A., Rosén, A.: Competitive non-preemptive call control. In: Proceedings of SODA 1994, pp. 312–320. SIAM (1994)
2. Barhum, K., et al.: On the power of advice and randomization for the disjoint path allocation problem. In: Geffert, V., Preneel, B., Rovan, B., Štuller, J., Tjoa, A.M. (eds.) SOFSEM 2014. LNCS, vol. 8327, pp. 89–101. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-04298-5_9
3. Böckenhauer, H.-J., Komm, D., Královič, R., Královič, R., Mömke, T.: On the advice complexity of online problems. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 331–340. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10631-6_35
4. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
5. Dietiker, S.: The advice complexity of the online disjoint path allocation problem. Bachelor thesis, ETH Zürich (2013)
6. Dobrev, S., Královič, R., Pardubská, D.: Measuring the problem-relevant information in input. Theor. Inform. Appl. (RAIRO) **43**(3), 585–613 (2009)
7. Emek, Y., Fraigniaud, P., Korman, A., Rosén, A.: Online computation with advice. Theoret. Comput. Sci. **412**(24), 2642–2656 (2011)
8. Gebauer, H., Komm, D., Královič, R., Královič, R., Smula, J.: Disjoint path allocation with sublinear advice. In: Xu, D., Du, D., Du, D. (eds.) COCOON 2015. LNCS, vol. 9198, pp. 417–429. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21398-9_33
9. Hromkovič, J., Královič, R., Královič, R.: Information complexity of online problems. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 24–36. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15155-2_3
10. Komm, D.: An Introduction to Online Computation: Determinism, Randomization, Advice. Springer, Switzerland (2016). https://doi.org/10.1007/978-3-319-42749-2
11. Kováčová, I.: Advice complexity of disjoint path allocation. Theor. Inform. Appl. (RAIRO) **50**(2), 171–191 (2016)
12. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Commun. ACM **28**(2), 202–208 (1985)