RAIRO-Theor. Inf. Appl. 50 (2016) 171–191
DOI: 10.1051/ita/2016020

Available online at:
www.rairo-ita.org

ADVICE COMPLEXITY OF DISJOINT PATH ALLOCATION

Ivana Kováčová[1]

Abstract. This paper contributes to the research of advice complexity of online problems. Namely, we discuss the disjoint path allocation problem in various versions, based on the choice of values of the calls, and ability to preempt. The advice complexity is measured relative to either the length of the input sequence of requests, or the length of the input path. We provide lower and upper bounds on advice complexity of optimal online algorithms for these problems, and some bounds on trade-off between competitiveness and advice complexity. One of the results is an improved lower bound of $n-1$ on advice complexity for the non-preemptive version with constant values of calls. For all considered variations, the newly provided lower and upper bounds on advice complexity of optimal algorithms are linear, and therefore asymptotically tight.

Mathematics Subject Classification. 68Q25, 68R10, 68W27.

1. Introduction

In the classical approach to the design and analysis of algorithms it is assumed that an algorithm has the complete knowledge of the entire input. However, this assumption is unrealistic in a number of practical applications. In many fundamental real-world algorithmic problems the input comes sequentially and the output has to be produced continuously without knowing the entire input. These problems are called *online problems*. The concept of online problem has been intensively investigated since its introduction in the late sixties by Graham [7]. We refer the reader to Borodin, El-Yaniv [4], and Albers [1] for a comprehensive survey on online problems. Despite the fact that online problems are studied for almost 50 years, the field still offers many fundamental unsolved problems. Moreover, a large number of online problems has been examined only for a very limited scope of parameters, thus providing only restricted understanding.

In recent years, a research of usefulness of various kinds of information has been emerging. The main question in this type of research is to study information with respect to a particular problem, that is, how can additional information reduce the complexity of the problem. In an online setting, the research of additional information (about future request) is investigated using the notion of advice complexity introduced by Dobrev *et al.* [5]. Advice complexity of a problem measures the amount of problem relevant information that is not available to the algorithm from the beginning of the computation. Many online problems has been studied with respect to their advice complexity recently [2,3,6,8].

In this paper, we contribute to the research of advice complexity for online problems by analyzing various versions of the disjoint path allocation problem (DPA for short). The input of this problem consists of n calls

Keywords and phrases. Advice complexity, online problems, disjoint path allocation.

[1] Department of Computer Science, Comenius University, Bratislava, Slovakia. seleceniova@dcs.fmph.uniba.sk

Article published by EDP Sciences

(paths between two nodes on an input line of length L), and the goal is to maximize the gain of accepted non-overlapping calls.

The advice complexity of DPA was previously investigated only in the non-preemptive DPA_1 variant, for which the value of a call is one. When the advice complexity is measured with respect to the length of the input path L, Barhum *et al.* [2] proved that $L-1$ bits are both necessary and sufficient to obtain optimal algorithm. In the case the advice complexity is measured with respect to the length of the input sequence n, Komm [8] provided upper bound of n, and lower bound of $n/2$. When the trade-off between the number of advice bits and the competitiveness is considered, Böckenhauer *et al.* [3] proved that to get c-competitive algorithm for non-preemptive DPA_1, the minimum of $(n/c+3)\log n + O(1)$ and $n\log(c/(c-1)^{(c-1)/c}) + 3\log n + O(1)$ bits are sufficient.

In this paper, we extend the existing results by providing various linear upper and lower bounds on the advice complexity of optimal online algorithms. In the case the advice complexity is measured according to the length of the input sequence n, we improve the known lower bound for the non-preemptive version of DPA_1 to $n-1$. For the non-preemptive version of DPA_ℓ (for which the value of a call is proportional to its length) the bounds are tight, and equal to n. In the preemptive case, the bounds for both DPA_1 and DPA_ℓ are equal, the lower bound is $n/2-1$, and the upper bound is $0.78n + O(1)$.

For the advice complexity measured with respect to the length of the input path L, we show that the bounds on the advice complexity for non-preemptive DPA_ℓ are tight, equal to $L-1$. For the preemptive cases, the upper bound of $0.67L$ is achieved for both DPA_1 and DPA_ℓ. The lower bounds differ, and are equal to $0.405L$ for DPA_1 and $L/3$ for DPA_ℓ.

In Section 4, we investigate trade-off between competitiveness and the number of advice bits needed. We construct c-competitive algorithm (for all considered variations of DPA) that uses at most $L/c + \log_2 c + 2$ bits of advice. The same algorithm also works for measurement of advice complexity with respect to n, but additional $2\lceil\log_2\lceil\log_2 n\rceil\rceil + \lceil\log_2 n\rceil$ bits of advice are needed for encoding of n, which is not known to the algorithm. This result improves the known upper bound given by Böckenhauer *et al.* For the lower bound, we prove that, for small values of c, any c-competitive algorithm needs at least $(1+(1-\alpha)\log_2(1-\alpha)+\alpha\log_2\alpha)L/3$ bits of advice, where $\alpha = (2-c)/c$ for DPA_1, and $\alpha = (3-2c)/c$ for DPA_ℓ. The analogous bounds hold when the advice complexity is measured with respect to n.

2. Preliminaries

The disjoint path allocation problem belongs to the class of *online maximization problems*. The most commonly used form of measurement of the output quality of online algorithms is *competitive analysis* introduced by Sleator and Tarjan [10]. For maximization problems, an algorithm $\mathtt{A}$ is said to be c-competitive if there exists a constant α such that, for every input sequence I,

$$c \cdot \mathrm{gain}(\mathtt{A}(I)) + \alpha \geq \mathrm{gain}(\mathtt{Opt}(I)),$$

where $\mathtt{Opt}$ is an optimal offline algorithm for the problem, and $\mathrm{gain}(\mathtt{A}(I))$ is the gain of the solution produced by $\mathtt{A}$ on input I. If $\alpha = 0$, then $\mathtt{A}$ is called *strictly c-competitive*. We call an algorithm *optimal* if it is strictly 1-competitive. For formal definition of these concepts we refer the reader to Komm [8].

We investigate online algorithms with advice as defined by Böckenhauer *et al.* [3]. For every input, an oracle that knows the input produces advice for the algorithm. The algorithm accesses the advice bits in the same way as randomized algorithm accesses random bits – it reads them from a special tape. The oracle has to provide enough information so the algorithm never hits the end of the advice tape. The advice complexity of the algorithm is then the number of bits read from the advice tape.

As mentioned in the introduction, we study two versions of DPA, denoted by DPA_1 and DPA_ℓ, which vary in the evaluation of the solution. In both cases, one can study either non-preemptive or preemptive algorithms.

Definition 2.1. Let us consider a path $P = (V, E)$, where $V = \{v_0, \ldots, v_L\}$ is a set of vertices. An input sequence $I = (c_1, \ldots, c_n)$ of DPA consists of calls. Every call is a triple $c_i = (s_i, t_i, val_i)$ where s_i, t_i are source and target vertices respectively, and $val_i > 0$ is a value of the call. A call can be either rejected or accepted. All accepted calls have to be mutually edge-disjoint. The goal is to maximize the value of the accepted calls. Furthermore, a preemptive version of the problem can be defined. In that case, an algorithm can at any time preempt a previously accepted call.

The special case where for every call $val_i = 1$ is denoted by DPA_1, the special case where for every call $val_i = |t_i - s_i|$ is denoted by DPA_ℓ.

Remark 2.2. For online algorithms, an usual assumption is that they cannot revoke their past decisions. However, there are problems in which preempting (cancelling) a past decision makes sense. The disjoint path allocation problem is one of them.

3. Advice complexity of the disjoint path allocation problem

In this section we investigate the advice complexity of several variants of the disjoint path allocation problem. We consider two parameters that can be modified. The first is the value of a call, which can be either constant, as in the commonly studied version, or proportional to the length of the call. As the second one, we consider the ability to preempt, *i.e.*, to reject some of the previously accepted calls in any time step. Furthermore, we study the advice complexity of these problems with respect to either the length L of the input path, or the length n of the input sequence. Since the advice complexity of the original version of this problem is well studied, it is naturally a good candidate for investigating how these variations influence the complexity of the problem.

3.1. DPA_1 with respect to L

We start our study of advice complexity with the most investigated version of the online disjoint path allocation problem, DPA_1, in which the value of every call is equal to one. Even though DPA_1 is relatively well studied, preemption has not been examined in the context of advice complexity so far. As we have mentioned earlier, there are two natural parameters for the disjoint path allocation problem, for which it makes sense to measure the advice complexity. In this section, we restrict ourselves to one of them, the length L of the input path. The following section is devoted to the advice complexity with respect to the number of calls n in the given input sequence.

The goal of this section is to compare the advice complexity of non-preemptive and preemptive versions of DPA_1. We prove that the ability to preempt helps, as the reader may already suspect.

3.1.1. DPA_1 without preemption

Let us first consider the non-preemptive version of the problem. As most of the results about DPA_1 are made in this setting, it is not very surprising that both upper and lower bounds on the advice complexity of optimal algorithms are already known. In addition, these bounds are tight.

Theorem 3.1 (Barhum *et al.* [2])**.** *To solve* DPA_1 *optimally,* $L - 1$ *advice bits are necessary and sufficient.*

3.1.2. DPA_1 with preemption

This version has not been studied from the perspective of advice complexity, and standard methods for proving lower bounds on advice complexity can cause significant issues connected with the ability to preempt – there might be some "compatible" instances that are in conflict for the non-preemptive version, but preemption allows to solve them using the same advice string. At the same time, the existence of such instances might reduce the advice complexity of the problem.

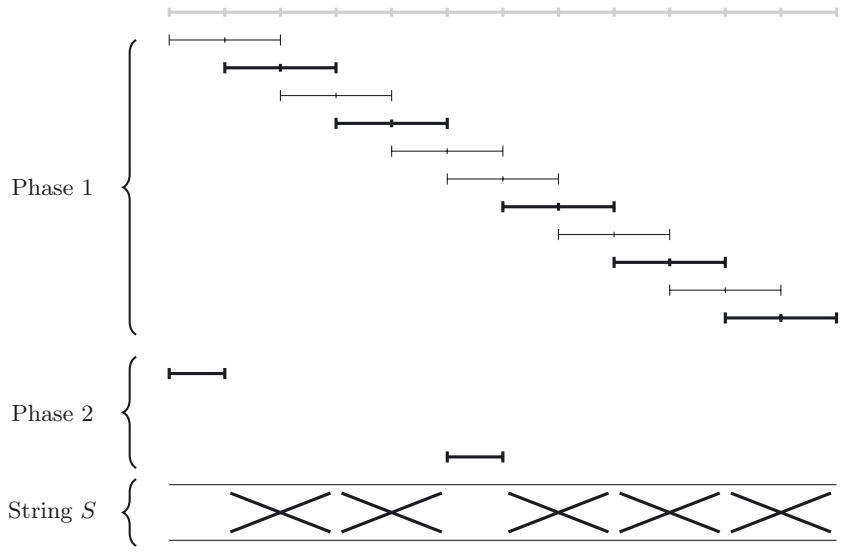


FIGURE 1. An example of a constructed instance used in the proof of Theorem 3.2. The optimal solution is displayed in bold.

In the next theorem, we use a sequence called the Padovan sequence. The Padovan sequence is recursively defined by the equation $P_k = P_{k-2} + P_{k-3}$, with the initial values set to $P_0 = P_1 = P_2 = 1$ [13]. The kth Padovan number can be estimated as $(r^{k+5})/(2r+3)$, where $r \approx 1.3247$ [9,11].

Theorem 3.2. *To solve* DPA_1 *with preemption optimally, at least* $\log_2(P_L) \approx 0.405L$ *bits of advice are necessary, where* P_k *is the* k*th number from the Padovan sequence.*

Proof. Let us start with the construction of instances, for which any algorithm needs different advice string in order to be optimal.

The instances consist of two phases: Phase 1 is the same for every instance, and it is composed of $L-1$ requests of length 2 in some fixed order. Let this order be as shown in Figure 1. Phase 2 is different for every instance, and it can be described according to a special string, which we call S, shown in Figure 1. The string S is composed of "crosses" of size 2 and "spaces" of size 1, with the restriction that consecutive spaces are not allowed. Phase 2 then consists of calls of length 1 corresponding to the spaces in S projected to the input line.

The set of instances that we are interested in corresponds to the number of all strings consisting of spaces and crosses with the above-mentioned restriction. To count them, we use a recursive approach, that is, we want to know how to create bigger strings from smaller ones. There are two possibilities: the first is that a longer string ends with a cross, so any string of length $L-2$ can be its prefix, and the second option is that a string ends with a space, which forces a cross in the previous position. In this case there can be anything of length $L-3$ before the cross. No different ending is feasible, because two consecutive empty spaces are not allowed. This corresponds to the Padovan recurrence as defined before the theorem.

Our next step is to show that any two instances from this set of instances need a different advice string. The optimal solution for an instance corresponding to a particular string contains calls of length two at the positions corresponding to the crosses in the string. Calls of length one from the optimal solution correspond to the spaces. Assume, by contradiction, that two different instances I_1 and I_2 have the same advice. As the optimal calls from the first phase uniquely determine the second phase, the optimal solutions for these instances have to differ in the choice of the first phase calls. In other words, there exists a call of length 2 such that it is in the optimal solution of I_1, but not in the optimal solution of I_2. With the same advice, an algorithm either

decides to accept the call or to reject it for both instances, which implies that in one of them the algorithm is not optimal. This is based on two observations, firstly, all the calls of length 1 are contained in the optimal solution. Moreover, these calls uniquely determine the choice of the calls of length 2 from the first phase in the optimal solution. Secondly, by accepting a "non-optimal" call of length 2, an algorithm loses the ability to accept two calls overlapping with this non-optimal call, since the optimal solution is compact, *i.e.*, there are no gaps between optimal calls.

From the estimate of kth Padovan number, it follows that the advice complexity of the preemptive version of DPA_1 is at least $(L+5)\log_2 r - \log_2(2r+3)$, which is approximately $0.405L$. □

Using ideas of the previous proof, we can show that there exists a matching upper bound for a restricted version of the problem, where only calls of length 1 and 2 are admissible.

Without preemption, the upper bound on advice complexity is $L-1$, however, using preemption, the upper bound can be significantly decreased. The main idea is to construct an algorithm `A` that exploits preemption and to partition the set of all instances into a small number of groups that can have the same advice string without affecting the optimality of the algorithm. The main problem is to find out which instances can have the same advice string and how to do this partitioning as well as possible.

The algorithm `A` works as follows. It reads an advice string representing the optimal solution. If a request from the optimal solution arrives, the algorithm accepts it. If the advice string represents multiple optimal solutions, the algorithm always preempts a larger request (from some optimal solution) in favour of a smaller one (also from some optimal solution). The algorithm does not accept any requests that are not described as optimal in the advice string at hand.

It is not necessary to consider all instances of the problem explicitly. At first, if two instances have the same optimal solution, the optimality of the algorithm `A` depends solely on the provided advice. Knowing the optimal solution is both necessary and sufficient for the algorithm. Next, let an optimal solution of an instance I_1 be a superset of an optimal solution of an instance I_2 (*i.e.*, the solution for I_2 does not contain some calls from the solution for I_1). Consider the advice string for I_1 that describes the optimal solution of I_1. We claim that the same advice string makes the algorithm optimal also for I_2, as the calls contained in I_2 that are in the optimal solution are also in the optimal solution of I_1, and therefore they will be accepted. That means that instances of type I_2 do not need to be explicitly considered. Finally, suppose an optimal solution contains several (at least two) consecutive calls of length 1. Then, this solution can be implicitly added to the group of solutions that contains an identical solution with the modification that the group of all consecutive calls of length one are replaced by one long call of the appropriate length. Accepting a call of length one instead of any longer call never decreases gain of the solution.

Therefore, for the purpose of the following theorem, it is sufficient to consider only *complete optimal solutions*, *i.e.*, optimal solutions that contain no gaps, and do not contain two consecutive calls of length one. All non-complete optimal solutions are solved according to the previous paragraph.

The following lemma determines which complete optimal solutions can have the same advice string without making the algorithm `A` non-optimal. We call two requests (a,b) and (c,d) *intersecting*, if $a < c < b < d$ or $c < a < d < b$. (We do not consider requests to be intersecting if one is a subset of another.)

Lemma 3.3. *Let O_1 and O_2 be two complete optimal solutions for some corresponding instances for* DPA_1*. If for every request r_1 ($r_1 \in O_1$) and every request r_2 ($r_2 \in O_2$), r_1 does not intersect with r_2, then O_1 and O_2 can be assigned to the same advice string, without leading to non-optimality of the algorithm.*

Proof. By the assumption of the lemma, the optimal solutions O_1 and O_2 do not contain two intersecting requests. Therefore, the only situation the algorithm has to resolve is, when in O_1 there is a request (v_a, v_b) and in O_2 there are requests (v_a, v_{a_1}), (v_{a_1}, v_{a_2}), …, (v_{a_k}, v_b) (or *vice versa*), where $a < a_1 < \ldots < a_k < b$. We claim that the algorithm can take all of these requests while preempting the larger one in favor of smaller ones. Indeed, if the optimal solution of a given instance is O_1, the algorithm does not decrease the gain of its solution

by this preemption. If, on the other hand, the optimal solution is O_2, the solution of the algorithm will contain all the requests $(v_a, v_{a_1}), \ldots, (v_{a_k}, v_b)$ as in the optimal solution.

Theorem 3.4. *To solve preemptive* DPA_1 *optimally,* $\lceil L/8 \cdot \log_2(41) \rceil \approx 0.67 \cdot L$ *bits of advice are sufficient.*

Proof. We shall prove that the algorithm A described above is optimal and uses only the specified number of advice bits. To do this, we start with all complete optimal solutions for a given L. We want to divide them into a minimal number of groups so that the conditions of Lemma 3.3 are preserved. For small L, this can be done using brute-force, and the minimal numbers of groups are summarized in the following table. The respective groups of complete optimal solutions into groups can be found in Appendix 4.

L	0	1	2	3	4	5	6	7	8
$T(L)$	1	1	1	2	3	4	6	9	14

Based on these numbers, we show how many advice strings we need for instances of length $L > 8$. To determine the number of different advice strings, we need to recursively construct groups of solutions of length L having the same advice, from the smaller groups of solutions of length at most $(L-8)$. We can encode each complete solution as a $(L+1)$-bit string with ones at the leftmost and rightmost positions, where a 1 at the ith position means that there is a request from the optimal solution with an endpoint at the ith node.

Let S be a solution of length $L-8$ represented by the string $s_0 s_1 s_2 \ldots s_{L-8}$. We call a solution S' an a-extension of S, (where a is an 8-bit string $a_1 a_2 a_3 \ldots a_8$) if the representation of S' is $s_0 s_1 \ldots s_{L-9} a_1 a_2 \ldots a_8 s_{L-8}$. We create the new groups in the following manner: the a-extension of S and the b-extension of T are in the same group if and only if it holds that

(a) S and T are in the same group; and
(b) there exists i, such that a_i and b_i are the leftmost 1 in a and b, respectively, and the parts $a_i \ldots a_8 s_{L-8}$ and $b_i \ldots b_8 t_{L-8}$ are in the same group according to the partition of instances of the length $9-i$.

Therefore, if $T(L)$ denotes the number of sets of strings constructed as above, we have $T(L) = (\sum_{i=1}^{8} T(i)) \cdot T(L-8)$. From the fact that the sum of $T(i)$ for i from 1 to 8 is 41, we get $T(L) = 41 \cdot T(L-8)$. Since for each of the initial values it holds that $T(i) \leq 2^{i \cdot (\log_2 41)/8}$, we can prove by induction that

$$T(L) = 41 \cdot T(L-8) \leq 41 \cdot 2^{(L-8)\cdot(\log_2 41)/8} \leq 2^{L \cdot (\log_2 41)/8}. \qquad \square$$

3.2. DPA_1 with respect to n

Now we consider the second parameter used to express advice complexity, the number of requests n. One of the differences between the parameters n and L, from the point of view of online algorithms, is that the number of input requests is unknown to an algorithm during computation, while L is known in advance. Even if it can, theoretically, affect effectiveness of an algorithm, it is not as dramatic as it might seem. In the non-preemptive version, an optimal online algorithm needs in both cases as many advice bits as is the size of the considered parameter, up to an additive constant.

3.2.1. DPA_1 without preemption

The best known upper bound on the advice complexity of DPA_1 is $s(n) = n$, due to Komm [8].

Theorem 3.5 (Komm [8])**.** *There exists an optimal online algorithm with advice for the non-preemptive version of* DPA_1 *with advice complexity* $s(n) = n$.

Komm [8] also shows a lower bound of $n/2$, which can be further improved to $n-1$ as the next lemma and the subsequent theorem claim.

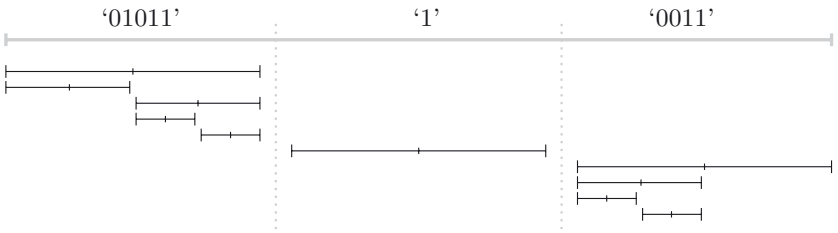


FIGURE 2. Instance corresponding to the string '0101110011', calls arrive in order from top to bottom. A call is in the optimal solution if the corresponding bit in the string is 1.

Lemma 3.6. *To solve* DPA_1 *without preemption optimally,* $(n-2)$ *bits of advice are necessary.*

Proof. Consider all strings of length n that end with the substring '11'. The number of these strings is 2^{n-2}. For every such a string we create an instance of DPA_1 in the following way. Firstly, we divide the string into boxes so that each box is either '1' or it is a shortest substring which starts with '0' and ends with '11'. The calls within a box are formed as follows:

1. If a box contains only the string '1', then we represent such a box by one call of length 2^{n+1}.
2. If a box contains a string '0 ... 11' of length k, then the first zero represents a call of length 2^{n+1}. For all $j \geq 1$, if the jth bit (bits are numbered from 1 to k) in the box is 0, then the $(j+1)$th call has the same starting point as the previous call but with half of its length. For all $j < k-1$, if the jth bit in the box is 1, then the $(j+1)$th call has the same length as the previous one but with the starting point at the same position as the ending position of the previous call.

The boxes are arranged on the interval one after another, that is, if the rightmost endpoint of a call from the kth box is at position ℓ, then the first call of the $(k+1)$th box starts at position $\ell+1$. An example of such an instance can be found in Figure 2.

Based on the construction of the instances from the strings, the following observations can be made.

Observation 1. *Let* s_1 *and* s_2 *be strings that have the same prefix of length* j*, differing at position* $j+1$ *for the first time. Then instances corresponding to these two strings are equal in the first* $j+1$ *calls.*

Observation 2. *Let* c *be a call that corresponds to a zero bit in the string. Then, in the box containing* c*, at least two disjoint calls arrive after* c *that are subsets of call* c*, and they are accepted by the optimal algorithm.*

Observation 3. *Let* c *be a call corresponding to 1 in the string. Every call arriving after* c *will not intersect with* c*.*

Our aim is to prove that any algorithm needs at least $n-2$ advice bits to optimally solve each of these 2^{n-2} instances. To do this we need to prove that every instance in this set needs an advice string that differs from the advice strings of all the other instances. We shall prove this claim by contradiction, that is, we assume that there is an algorithm and a pair of instances such that the algorithm solves both instances optimally with the same advice string.

Let us assume that I_1 and I_2 are instances corresponding to the strings s_1 and s_2, such that an algorithm obtains the same advice string for both of them. Furthermore, let s_1 and s_2 be equal up to the jth bit. Without loss of generality, let the $(j+1)$th bit in s_1 be 0 and the $(j+1)$th bit in s_2 be 1. From Observation 1 and the fact that the instances have the same advice string follows that the algorithm cannot distinguish between them when the $(j+1)$th call arrives.

It does not matter whether the algorithm accepts the jth call or not, and we can assume it decides optimally, as both instances have the same optimal solution up to the jth call. For our analysis, the important part is how the algorithm decides later. Only two cases can happen: either the algorithm accepts the $(j+1)$th call or not.

1. The algorithm accepts the call $j+1$. Then we claim that the algorithm is not optimal for I_1. This follows from Observation 2, as the algorithm accepts only one call (namely the $(j+1)$th call) where the optimal algorithm can accept at least two calls. Hence, the algorithm is not optimal in this case.
2. The algorithm does not accept the call $j+1$. Let us look at the calls of I_2 intersecting with this call. From Observation 3 it follows that no subsequent call can overlap with it. Furthermore, if any previous call c overlaps with it, by Observation 3 it now follows that the call c corresponds to 0 in s_2, which implies that it is not accepted by the optimal algorithm (see Observation 2). Therefore, not taking the $(j+1)$th call means that the solution of the algorithm can be improved (by accepting the $(j+1)$th call), leading to non-optimality of the algorithm. □

This lower bound can be further improved by considering instances of different lenghts, as shown in the next theorem.

Theorem 3.7. *To solve* DPA_1 *optimally,* $n-1$ *bits of advice are necessary.*

Proof. The theorem claims that any algorithm A with advice complexity $n-2$ or less (except for finitely many inputs) cannot be optimal. We prove this by contradiction, that is, we assume that for every input I of length n (except for finitely many input lengths) there exists some advice string ρ such that the algorithm A is optimal using the advice string ρ on instance I and it uses at most $n-2$ bits of advice on I.

Let us consider some fixed n_0 such that the algorithm has advice complexity at most $n-2$ for all lengths $n \geq n_0$. Furthermore, consider all strings of length n_0 ending with '11' and one string s' of length n_0+1, with only one zero at position n_0-1. Denote by $\mathcal{I}$ the set of instances obtained from all these strings using the process described in the proof of Lemma 3.6, and by I' the instance corresponding to the string s'. (Note that I' is contained in $\mathcal{I}$.) All these instances will be presented on a path with the same length L, so the algorithm cannot distinguish between them according to the length of a path. The algorithm is allowed to read one additional bit of advice for the instance I', since it contains one extra request. Since $\mathcal{I} \setminus I'$ has size 2^{n_0-2} and the instances use an advice string of length n_0-2, every string with length n_0-2 is an advice for some instance as follows from the proof of Lemma 3.6.

Let us look at the advice string corresponding to the last instance I', particularly at the first n_0-2 bits of it. We know that these n_0-2 bits are the same as some advice bits for an instance from $\mathcal{I} \setminus I'$, let I_j denote that instance. Instances I_j and I' have some common prefix, that is, up to some point, they contain the same requests. Let us denote the length of the longest common prefix by k, $k < n_0$. Note that the optimal solutions differ at this point, where for I_j the kth call should be rejected, and for I' it should be accepted or *vice versa*.

After the kth call arrives, the algorithm has to decide whether to accept it. However, the algorithm cannot distinguish between the two instances, since up to this point, it can use at most n_0-2 bits of advice and instances look the same so far. The algorithm is deterministic, therefore it can either read another bit of advice, or deterministically accept/reject the kth call. In the first case, it uses n_0-1 bits of advice on instance I_j of length n_0, which is a contradiction. In the second case, it will not be optimal either on the instance I_j or I'. □

Remark 3.8. The method used in the previous proof is not restricted to only this case. In fact, it can be used for other problems as well, and for other parameters. The only restriction is that the considered parameter (in this case it is the number of requests) must be unknown to the algorithm, even during the computation (at least until it is too late, and the algorithm already made a mistake).

3.2.2. DPA_1 with preemption

For the lower bound of algorithms with preemption, we use the same idea as used initially for the non-preemptive version of DPA_1, where the obtained bound was $n/2$. A similar bound can be proven for the preemptive version as well, but as usual, the set of instances has to be slightly modified since for the original instances there exists an optimal preemptive algorithm with advice complexity $s(n) = 0$.

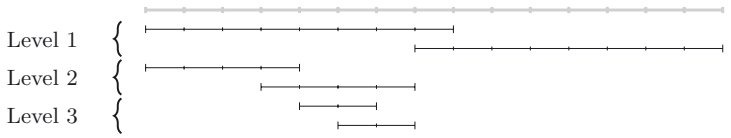


FIGURE 3. An example of input instance in Theorem 3.9.

Theorem 3.9. *To solve* DPA_1 *with preemption optimally,* $n/2-1$ *bits of advice are necessary.*

Proof. Let the length of the path be $L = 2^{k+1} - 1$. We create a set of instances where each instance consists of k levels $L_1, \ldots, L_k$, and every level consists of two calls. (Therefore, the number of requests n is equal to $2k$.) The first level L_1 contains calls $(v_0, v_{\lfloor L/2 \rfloor + 1})$ and $(v_{\lfloor L/2 \rfloor}, v_L)$. If the ith level contains calls (v_a, v_{b+1}) and (v_b, v_c) (of the same length ℓ), then there are two possibilities for the $(i+1)$th level: it either contains the calls $(v_a, v_{a+\ell/2})$ and $(v_{b-\ell/2}, v_b)$, or the calls $(v_{b+1}, v_{b+1+\ell/2})$ and $(v_{c-\ell/2}, v_c)$. An example of such an input instance can be seen in Figure 3.

As we have two choices in every level except level L_1, the total number of instances is 2^{k-1}. The gain of the optimal solution is k for each instance – in every level L_i, there exists a call that does not intersect with the following levels. Furthermore, both calls from the same level cannot be in a feasible solution, hence the gain of any feasible solution is at most k. For the sake of clarity, we denote by $L_i^{(I)}$ the ith level of the instance I.

To complete the proof, it suffices to show that any optimal algorithm with advice has to obtain different advice strings for every such instance. Towards contradiction, assume the opposite, that is, there is an optimal algorithm with advice such that for two distinct instances I_1 and I_2 it provides an optimal solution using the same advice string ϕ. From $I_1 \neq I_2$ it follows that there exists $i > 1$ such that $L_i^{(I_1)} \neq L_i^{(I_2)}$, but $L_j^{(I_1)} = L_j^{(I_2)}$ for every $j < i$. The algorithm has the same advice string for both I_1 and I_2 and these instances are equal up to the level L_{i-1}, hence the algorithm chooses the same call from level L_{i-1} on both instances. However, for one of the instances, level L_i consists of calls that intersect with the call accepted at level L_{i-1}. Therefore, the algorithm cannot be optimal on this input instance.

From the above it follows that any optimal online algorithm with advice needs at least $2^{k-1} = 2^{n/2-1}$ different advice strings, and its advice complexity is therefore $s(n) \geq \log(2^{n/2-1}) = n/2 - 1$. □

For the upper bound we use an auxiliary lemma by Böckenhauer *et al.* [3].

Lemma 3.10 (Böckenhauer *et al.* [3], Komm [8])**.** *Consider an online algorithm* A *with advice complexity* $s_{\mathtt{A}}(n) = n$ *and competitive ratio* r*. Moreover, assume that, for every input instance,* A *achieves a competitive ratio of* r *while using some* n*-bit advice string that contains at most* n/t *zeros or at least* $n - n/t$ *zeros, where* $t > 2$ *is a fixed constant. Then, it is possible to design an improved online algorithm* B *with the same competitive ratio* r *that knows the parameter* t *and has an advice complexity of*

$$s_{\mathtt{B}}(n) \leq \min\left\{ n \log\left(\frac{t}{(t-1)^{\frac{t-1}{t}}} \right), \frac{n \log n}{t} \right\} + 3 \log n + \mathcal{O}(1). \tag{1}$$

Theorem 3.11. *To solve preemptive* DPA_1 *optimally,* $7n/9 + \mathcal{O}(1) \approx 0.78n + \mathcal{O}(1)$ *bits of advice are sufficient.*

Proof. Let A be an algorithm that takes any incoming call that does not conflict with anything accepted before. Furthermore, when there is a conflict between the most recent call and one, two, or more of the previously accepted calls, the algorithm asks for advice. In the case that only one previously accepted call c_1 intersects with the most recent call c, the advice is interpreted as an answer to the question: is c_1 in the optimal solution? If yes, the algorithm rejects c, otherwise it accepts c and preempts c_1. In the second case, let c_1 be the left and c_2 be the right call in conflict with c. The algorithm first asks if c_1 is in the optimal solution. If yes, the algorithm

rejects c. If not, it asks whether c_2 is in the optimal solution. If yes, the algorithm preempts c_1 and rejects c, otherwise it preempts both c_1 and c_2 and accepts c. If there are more than two calls conflicting with c, then c can be rejected as there exists a call which is a subset of c with the same value, so there is no advantage to have c in the optimal solution.

Such an algorithm can ask many questions but, for us it is only important whether the number of questions is at least $7n/9$. If not, then the algorithm reads at most $7n/9$ bits of advice. Otherwise, we can continue in the following manner. We show that the optimal solution consists of a small number of calls, which means that the advice can be compressed using Lemma 3.10. To do that, let us introduce some new notation.

O_{wa} – a set of the calls from the optimal solution that the algorithm takes without asking for advice about them, and even after accepting these calls, these calls never conflict with any other incoming call.

O_a – a set of the calls from the optimal solution, for which the algorithm reads a bit of advice.

N_o – a set of the calls that are not in the optimal solution, and also the algorithm never asks about them, because when these calls arrived the algorithm asked about a conflicting call from O_a.

N_{wa} – a set of the calls that are not in the optimal, solution and the algorithm also never asks about them, because these calls conflict with calls that are known to be in O_a, or because they are intersecting with more than two currently accepted calls (one of them is a subset of such a call).

N_a – a set of the calls for which the algorithm reads one bit of advice and decides to preempt them.

The following observations can be made relative to the above-defined sets.

Observation 4. *If the algorithm is asking at least $7n/9$ questions, then there exist at most $2n/9$ calls for which the algorithm does not ask for an advice.*

Observation 5. *The algorithm asks about a call at most once.*

Observation 6. *For every call x from O_a, there exists a call y not in the optimal solution* (i.e., *in N_o*) *such that the algorithm asks whether to take x because of y.*

Observation 7. *For every call x from O_a and every conflicting call y, the algorithm will not ask about y.*

Based on the previous observations, we can conclude these simple equations:

- The number of calls for which the algorithm is asking for an advice bit can be bounded by
$$O_a + N_a \geq 7n/9.$$
- The number of calls for which the algorithm never asks for an advice bit can be bounded by
$$N_o + N_{wa} + O_{wa} < 2n/9.$$
- From the Observation 6 it follows that
$$N_o = O_a.$$

This can be combined to

$$O_a + N_{wa} + O_{wa} < 2n/9 \Rightarrow O_a + O_{wa} < 2n/9.$$

This result states that the number of calls from the optimal solution is less than $2n/9$. Now we use the above-described algorithm with advice complexity $s(n) = n$ that is asking a bit of advice for every request. Due to the size of the optimal solution, we can conclude that in this advice there are at least $n - 2n/9 = n - n/4.5$ zeros, and by applying Lemma 3.10 with $t = 4.5$, we can conclude that the advice complexity of this case is

$$\min\left\{n\log\frac{4.5}{3.5^{\frac{3.5}{4.5}}}, \frac{n\log n}{4.5}\right\} + 3\log n + \mathcal{O}(1) = 0.76n + 3\log n + \mathcal{O}(1) < \frac{7n}{9} + \mathcal{O}(1). \tag{2}$$

The final algorithm works as follows: it reads the first bit of advice. If it is 1, then it asks for an advice bit every time a conflicting call arrives (the first algorithm). If it is 0, then the algorithm decodes the advice (using Lem. 3.10), to obtain n bits of advice that tell the algorithm which calls are from the optimal solution and which are not. □

3.3. DPA$_\ell$ with respect to L

For the rest of this section, we will investigate another version of DPA called DPA$_\ell$, for which calls are valued according to their length, *i.e.*, a call of length i has value i. This version substantially differs from DPA$_1$ in the following way. Whereas in DPA$_1$ shorter calls are more beneficial for an algorithm than longer calls as their value is the same but they use less space, in DPA$_\ell$ this works the other way around. Longer calls are preferable in this case, because long calls provide safety of obtaining a significant gain, when the arrival of all the necessary short calls is uncertain, even if the gain could be the same. The known results about the deterministic algorithms for preemptive versions of DPA$_1$ and DPA$_\ell$ indicate that DPA$_\ell$ might be easier to solve than DPA$_1$.

3.3.1. DPA$_\ell$ without Preemption

At first, we look at the easier case of DPA$_\ell$, that is, where preemption is not allowed. The upper bound from DPA$_1$ (see Thm. 3.1) can be directly applied for this case, and a matching lower bound can be proven using the traditional approach, *i.e.*, by constructing a set of instances that need different advice strings.

Theorem 3.12. *To solve* DPA$_\ell$ *without preemption optimally,* $L-1$ *bits of advice are necessary.*

Proof. Let us take some arbitrary string s of length $L+1$ with symbol '1' at both ends. We create an instance corresponding to s in such a way that the optimal solution of the constructed instance will be described by s. More precisely, every substring of the string s that starts at the ith and ends at the jth position, with the only two ones at its endpoints, will correspond to a call (v_i, v_j) from the optimal solution. The instance will consist of several blocks (columns) of calls which can be described recursively. The first column is associated with the edge in the middle of the path of length L (we will refer to this path as the currently processed interval) – the $\lceil L/2 \rceil$th and $(\lceil L/2 \rceil + 1)$th position in the string s. (The bits in the string and the vertices on the path are indexed in the same way, from 1 to $L+1$.) The edge between these positions on the path is contained in some call c of length ℓ in the optimal solution. All calls of lengths from 1 to ℓ containing the edge $(v_{\lceil L/2 \rceil}, v_{\lceil L/2 \rceil + 1})$ arrive in a specific order as illustrated in Figure 4 (If some call is out of bounds of the currently processed interval then this call is omitted).

To create the rest of the instance, we continue recursively – firstly, the currently processed interval is divided into three parts, the part L_1 corresponds to the interval on the left side of c, L_2 corresponds to c, and the part L_3 corresponds to the interval on the right side of c. On the parts L_1 and L_3 we proceed recursively, finding their middle edges and building columns associated with them. Based on the construction of the instance, we can deduce that the only calls overlapping with c are in the column containing it.

From all the strings of length $L+1$ with ones at both ends we can create 2^{L-1} different instances using the above construction. The optimal solution of every instance is uniquely determined by the represented string, *i.e.*, the gain of any solution not described by the string is strictly less than the optimal gain. In the case that an algorithm accepts a non-optimal call B instead of A from the optimal solution, the part labeled C in Figure 5 will never be covered by any other call, because only one call from a column can be accepted, and, after the column containing A, no new call will overlap with A. (From the construction of the instance it follows that the length of B is smaller or equal to A.)

We prove by contradiction that every optimal algorithm needs a different advice for every such instance. Let I_1 and I_2 be two different instances that are processed using the same advice. Their optimal solutions differ, therefore we can find the first calls c_1 and c_2 from the optimal solutions (with respect to the construction of an instance) of the respective instances I_1 and I_2 such that these calls differ. Since every optimal call preceding c_1 and c_2 is the same in both instances, all columns preceding the column containing c_1 or c_2 are the same in both instances. Furthermore, c_1 and c_2 are in the same column (The columns start in the same manner in both instances, possibly differing at the end). We distinguish two cases.

1. In the case that the length of c_1 is the same as the one of c_2, the column containing c_1 and c_2 is the same in both instances. The algorithm needs to accept c_1 in I_1 and c_2 in I_2, which is not possible since

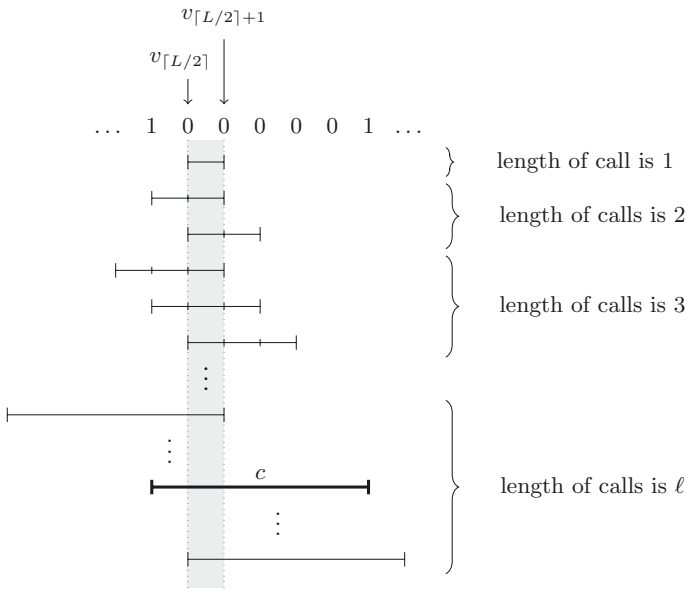


FIGURE 4. A building block (column) of an instance. The bold call is the one from the optimal solution.

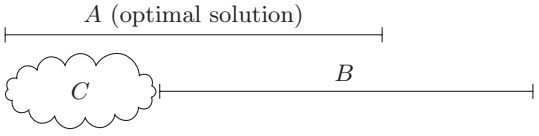


FIGURE 5. Uniqueness of the optimal solution.

it is deterministic, the instances are indistinguishable at this point, it has the same advice string for both of them, and c_1 overlaps with c_2. Therefore, the algorithm is either not optimal or needs different advice strings for I_1 and I_2.

2. The second case is when the lengths of c_1 and c_2 differ. Without loss of generality, assume that c_1 is shorter than c_2. By the construction of the instances, the column containing c_1 in I_1 is a prefix of the column containing c_2. Therefore, at the point c_1 arrives, the two instances are undistinguishable by the algorithm, and the algorithm needs to accept c_1 in the first instance and reject it in the other one. This leads to a contradiction similarly as in the previous case. □

The upper bound can be easily obtained similarly as in Theorem 3.1, by using an algorithm that interprets advice as a description of optimal solution.

Theorem 3.13. *To solve* DPA_ℓ *without preemption optimally,* $L-1$ *bits of advice are sufficient.*

3.3.2. DPA$_\ell$ with preemption

Based on the previous sections, the reader might already suspect that preemption is useful in any setting, and this case is not an exception. We shall start with the lower bound, where there is a proving method that is based on splitting the whole instance into boxes, such that an optimal algorithm needs separate advice for every box.

Theorem 3.14. *To solve* DPA_ℓ *with preemption optimally,* $L/3$ *bits of advice are necessary.*

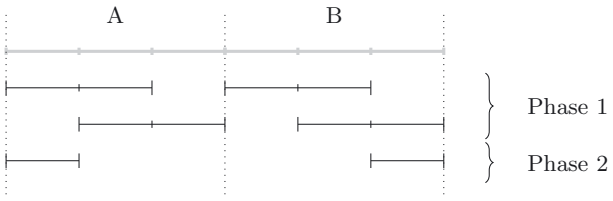


FIGURE 6. Two types of boxes.

Proof. Consider $2^{L/3}$ instances constructed in the following manner. The input path of length L is split into boxes of length 3, in every box one of the two types of calls as illustrated in Figure 6 arrive.

An algorithm, to be optimal, needs to know what is the type of each and every box. Otherwise, it would choose an incorrect call from the first phase, which would lead to a non-optimality. Since every box is independent from the other boxes, the optimal algorithm needs to obtain $L/3$ bits of advice. □

Also in the case of the upper bound we can use the same algorithm for DPA_ℓ as for DPA_1, since advice for this algorithm encodes an optimal solution, and the value of a call only influences the way the optimal solution looks like. The algorithm uses preemption in a way that one advice string is used for several instances. These sets of instances are selected in such a way that the algorithm is optimal for all of them, and, at the same time, the number of different sets is minimized. The algorithm for DPA_ℓ will be slightly different than the one for DPA_1 – it will prefer longer calls instead of shorter ones. Naturally, the proof has to be modified as well. However, the only part that has to be reviewed lies in the proof of Lemma 3.3. Recall that a complete solution does not contain any gaps between the accepted calls, and does not contain two consecutive calls of length one.

Lemma 3.15 (Alternative to Lem. 3.3)**.** *Let I_1 and I_2 be two different instances for* DPA_ℓ*, and let O_1 and O_2 be the corresponding complete optimal solutions. If, for every request r_1 ($r_1 \in O_1$) and every request r_2 ($r_2 \in O_2$), r_1 does not intersect with r_2, then I_1 and I_2 can be assigned to the same advice string, without leading to non-optimality of the algorithm.*

Proof. By the assumption of the lemma, the optimal solutions O_1 and O_2 do not contain two intersecting requests. Therefore, the only situation the algorithm has to resolve is, when there is a request (v_a, v_b) in O_1 and there are requests (v_a, v_{a_1}), (v_{a_1}, v_{a_2}), …, (v_{a_k}, v_b) in O_2 (or *vice versa*), where $a < a_1 < a_2 < \ldots < a_k < b$. We claim that the algorithm can take all of these smaller requests, and when the call (v_a, v_b) is revealed, the algorithm preempts the smaller calls in favor of it. Indeed, if the optimal solution of a given instance is O_2, the algorithm does not decrease the gain of its solution by this preemption. If, on the other hand, the optimal solution is O_1, the solution of the algorithm will contain the request (v_a, v_b) as the optimal solution does. □

Theorem 3.16. *To solve* DPA_ℓ *with preemption optimally, $\lceil L/8 \cdot \log_2(41) \rceil$ bits of advice are sufficient.*

3.4. DPA_ℓ with respect to n

In this case, the majority of the results can be adopted from the previous sections, and we will therefore only briefly discuss the main differences in the proofs.

3.4.1. DPA_ℓ without preemption

The lower bound for the non-preemptive version of DPA_ℓ is the only one that requires new ideas. We proceed in two steps. Firstly, we prove a lower bound of $n-1$ using the standard technique. Then we use the technique described in Theorem 3.7 for improving the lower bound to n.

Lemma 3.17. *To solve* DPA_ℓ *without preemption optimally, $n-1$ bits of advice are necessary.*

FIGURE 7. Examples of instances for strings '001', '011', '101', '111', respectively.

Proof. Let us consider all binary strings of length n having the last bit 1. The number of these strings is 2^{n-1}. For every such string we create an instance in the following way. The first call of every instance has length 1 and starts at position 0. If the ith bit in the string is 0, then the $(i+1)$th call has the same starting position as the ith call and the length is 1 plus the length of the ith call. If the ith bit in the string is 1, then the $(i+1)$th call has length 1 and the starting position of this call is shifted by 1 to the right from the endpoint of the ith call.

Next we need to prove that if an algorithm wants to be optimal, then for any two instances it has to read different advice. Assume the contrary, that is, there exist two different instances with the same advice, such that some algorithm is optimal on both of them. Since the strings corresponding to the instances are different, let us denote by i the length of the longest common prefix of both strings. According to the construction of the instances, we can conclude that these two instances have the same calls up to the call at position $(i+1)$ but in one instance it is optimal to take the $(i+1)$th call, while in the other one it is optimal to reject it. Since the algorithm has the same advice for these two instances, it cannot distinguish between them, and it makes the same decision when the $(i+1)$th call arrives. If the algorithm rejects the $(i+1)$th call, it will not be optimal in the case that the $(i+1)$th bit in the string is 1, as all the following calls will not intersect with the $(i+1)$th call, and it was the longest call among the calls with the same starting position as this call. Otherwise, if the algorithm accepts the $(i+1)$th call, it will not be optimal in the case that the $(i+1)$th bit in the string is 0, as there will arrive some longer call (incurring a larger gain) with the same starting position, which will not intersect with the following calls, and therefore it will be accepted by the optimal algorithm.

As shown above, any optimal algorithm needs a different advice string for each of the 2^{n-1} instances, which leads to a lower bound on the advice complexity of $n-1$. □

The lower bound from Lemma 3.17 can be further improved to n, which matches the upper bound from Theorem 3.19.

Theorem 3.18. *To solve* DPA_ℓ *without preemption optimally, n bits of advice are necessary.*

Proof. The proof of this theorem is analogous to the proof of Theorem 3.7 that improves the lower bound by one. Based on Lemma 3.17, we consider instances corresponding to all strings of length n_0 ending with 1 and one string of length n_0+1 with only one zero at position n_0. □

In the case of an upper bound, as we already mentioned, the same algorithm (from Thm. 3.5) as for DPA_1 can be used. This algorithm reads for every request one bit of advice that tells the algorithm whether to take or reject the current request. Since this algorithm does not depend on the value of a call, we can also use it for DPA_ℓ.

Theorem 3.19. *To solve* DPA_ℓ *without preemption optimally, n bits of advice are sufficient.*

3.4.2. DPA$_\ell$ with preemption

For the lower bound of DPA_ℓ with preemption, the same statement as for DPA_1 holds, as stated in Theorem 3.9. In this case, the optimal solutions for the considered instances have gain $L-1$, instead of k as in the original proof. However, these optimal solutions are the same as before, and each one of them is unique for the respective instance. Therefore, apart from the total gain, the values of the calls do not influence the proof, and every wrong choice leads to a non-optimal solution. This implies the validity of the following theorem.

Theorem 3.20. *To solve* DPA_ℓ *with preemption optimally,* $n/2-1$ *bits of advice are necessary.*

For the upper bound, we can use the same algorithm as used in Theorem 3.11. The only difference is that in the case that a call c conflicts with more than two other calls, it cannot be automatically rejected. The algorithm will therefore need to ask about all the intersecting calls $c_1, \ldots, c_k$, from left to right, until it finds one for which the advice indicates that it belongs to a fixed optimal solution. If such a call c_i is found, the algorithm preempts all the previous calls $c_1, \ldots, c_{i-1}$, for which the answer was negative, and the call c is rejected. Otherwise, all the calls $c_1, \ldots, c_k$ are preempted, and c is accepted. This change can influence the sizes of the defined sets, but the necessary observations used in the proof are still valid. Therefore, all the arguments and equations based on these observations hold too, and the proof is correct also for DPA_ℓ.

Theorem 3.21. *To solve* DPA_ℓ *with preemption optimally,* $7n/9+\mathcal{O}(1)$ *bits of advice are sufficient.*

4. Advice trade-offs For DPA

So far, we have investigated only the advice complexity of optimal algorithms. In this section, we consider c-competitive algorithms, and study the trade-offs between the achieved competitive ratio and used number of advice bits.

Some trade-off results for the non-preemptive case are already known. In particular, when the advice complexity is measured with respect to the length of the input sequence, Böckenhauer *et al.* [3] provide an algorithm, which obtains imprecise description of the optimal solution that can be compressed according to Lemma 3.10.

Theorem 4.1 (Böckenhauer *et al.* [3])**.** *For every* c*, there exists an* c*-competitive online algorithm for* DPA_1 *with advice complexity*

$$s(n) \leq \left(\frac{n}{c}+3\right)\log n+\mathcal{O}(1),$$

or

$$s(n) \leq n\log\left(\frac{c}{(c-1)^{\frac{c-1}{c}}}\right)+3\log n+\mathcal{O}(1),$$

whichever is smaller.

We improve this bound in the following manner. In the previous section, we repeatedly used modifications of an algorithm that receives an advice string of length L. The advice string consisted of a description of the optimal solution, where each bit of advice is associated with a position in the input path. If, on a particular position, there is an endpoint of some call from the optimal solution, the corresponding bit in the string is set to one, otherwise it is zero. The algorithm accepts only the calls whose endpoints correspond to the ones in the advice string, with zeros between them. Such an algorithm with advice works regardless of the used values of the calls or ability to preempt.

A modification of the algorithm is also used in the following theorem, using an idea that if an algorithm is supposed to be c-competitive, it is sufficient for it to be optimal only on a part of the input path. Therefore, knowing only a part of the above mentioned advice string is enough for the algorithm.

Theorem 4.2. *For any integer* $c \geq 2$*, there exists a* c*-competitive algorithm for* DPA_1 *with or without preemption using at most* $L/c+\log_2 c+2$ *bits of advice.*

Proof. For a given c, let us start with the description of a c-competitive algorithm A_c. The algorithm divides the input path of length L into c boxes, with lengths $\lfloor L/c \rfloor$ or $\lfloor L/c \rfloor+1$. In particular, the first $(c-(L \bmod c))$ leftmost boxes have length $\lfloor L/c \rfloor$, and the last $(L \bmod c)$ boxes have length $\lfloor L/c \rfloor+1$. The algorithm reads an advice string of length $\lceil \log_2 c \rceil+\lfloor L/c \rfloor+1$ that is interpreted as follows. The first $\lceil \log_2 c \rceil$ bits of advice form

a binary representation of the position of a chosen box. The box is chosen in such a way that the number of calls from the optimal solution contained (or partially contained) in the box is maximal. The rest of the advice represents the optimal solution in the chosen box, interpreted as usual, with bits 1 on the endpoints and 0 on the other positions. The algorithm then solves the described box optimally, and rejects any call outside of the chosen box. It is clear that the algorithm is able to accept all requests from the chosen box, because if a call has both endpoints in the box then it is directly described in the advice. If only one endpoint is in the box then the algorithm is able to accept a call with the same endpoint and with possibly different length, and therefore accepts the same number of calls from the given part as the optimal solution.

It remains to show that the described algorithm achieves the required competitive ratio. Assume the optimal solution contains k requests. If L is divided into c boxes, then we claim that there exists a box containing at least $\lceil k/c \rceil$ requests from the optimal solution. Then, the competitive ratio can be computed as

$$\frac{\text{gain}(\texttt{OPT})}{\text{gain}(\texttt{A}_c)} = \frac{k}{\lceil \frac{k}{c} \rceil} \leq \frac{k}{\frac{k}{c}} = c,$$

which concludes the proof. □

The algorithm from the previous theorem works, only with a slight modification in the advice string, also for DPA_ℓ. In this case the chosen box has to contain calls with maximal total gain. The algorithm is still c-competitive, as the gain of calls in the box described by the advice string is $\lceil \text{gain}(\texttt{OPT})/c \rceil$.

Theorem 4.3. *For any integer $c \geq 2$ there exists a c-competitive algorithm for DPA_ℓ with or without preemption using at most $L/c + \log_2 c + 2$ bits of advice.*

Remark 4.4. As stated in the theorem, the algorithm works for competitive ratio c that is an integer number. In the case the desired competitive ratio c is a real number, we can do the following. We let $c' = \lfloor c \rfloor$, and apply the algorithm for c'. This way, the achieved competitive ratio is strictly less than c, and therefore no advice bits are saved in comparison to c'.

The idea of the theorem can be also used for the competitive ratio $c \in [1, 2]$, with another modification of the algorithm. In this version, the input path is divided into boxes as well, but a box with *minimal* number of calls from the optimal solution is chosen, and determined by the advice string. The rest of the advice string consists of optimal solution for all the other boxes (ignoring the chosen box). However, the advice complexity of such an algorithm is more than the advice complexity of the optimal algorithm with advice described in the previous section.

Theorem 4.2 concerns an algorithm for which the competitive ratio is measured with respect to the length of the input path L. The same idea can also be used for the competitive ratio measured with respect to the number of input requests n as stated in the following theorem. In this case, however, the parameter n is not known to the algorithm, which is the reason for employing the following observation, which tells us how many advice bits are sufficient to pass an unknown value to the algorithm in a self-delimited form.

Observation 8. *At most $2\lceil \log_2 \lceil \log_2 n \rceil \rceil + \lceil \log_2 n \rceil$ bits of advice are sufficient to be communicated the value n to the algorithm.*

Theorem 4.5. *For any real $c \geq 2$ there exists a c-competitive algorithm for DPA_1 (or DPA_ℓ) with or without preemption using*

$$\lceil n/\lfloor c \rfloor \rceil + \lceil \log_2 \lfloor c \rfloor \rceil + \lceil \log_2 n \rceil + 2\lceil \log_2(\lceil \log_2 n \rceil) \rceil$$

bits of advice.

Proof. Let us first discuss the version of the problem where the value of calls is 1. The algorithm works similarly as the algorithm from Theorem 4.2, there are only two differences: the first is that in this case we consider a real c instead of integer, and the second is that algorithm does not know the value n, while the previous algorithm knew the value L, therefore we need extra advice bits to code the value n. All in all, the following advice bits are needed: to determine which box is chosen – $\lceil \log_2 \lfloor c \rfloor \rceil$; to code the optimal solution for the chosen box – $\lceil n/\lfloor c \rfloor \rceil$; to represent n, which can be coded in the self-delimiting way according to Observation 8 – $\lceil \log_2 n \rceil + 2\lceil \log_2(\lceil \log_2 n \rceil) \rceil$. Adding all the parts together, the algorithm is using

$$\lceil n/\lfloor c \rfloor \rceil + \lceil \log_2 \lfloor c \rfloor \rceil + \lceil \log_2 n \rceil + 2\lceil \log_2(\lceil \log_2 n \rceil) \rceil,$$

bits of advice.

The compute the competitive ratio of the algorithm, assume the optimal solution is composed of k calls. Then, the competitive ratio is

$$\frac{\textsc{Opt}}{A} = \frac{k}{\left\lceil \frac{k}{\lfloor c \rfloor} \right\rceil} \leq \frac{k}{\frac{k}{\lfloor c \rfloor}} \leq \frac{k}{\frac{k}{c}} = c,$$

which concludes the proof for DPA_1.

The reasoning has to be slightly altered for DPA_ℓ. In particular, the box is chosen in such a way that the gain of the optimal solution in the box is maximized. The computation of the competitive ratio is then identical to the case of DPA_1 – if k denotes the total gain of the optimal solution, the algorithm achieves the gain $\lceil k/\lfloor c \rfloor \rceil$ from the chosen box, as in DPA_1. □

Again, there is a known bound for the non-preemptive version of DPA_1, when the advice complexity is measured with respect to the length of the input sequence. The full proof of the theorem can be found in the paper by Böckenhauer *et al.* [3].

Theorem 4.6 (Böckenhauer *et al.*, [3])**.** *For any non-preemptive online algorithm with advice for* DPA_1*, at least* $\frac{n+2}{2c} - 2$ *bits of advice are required to achieve a strict competitive ratio of* c*.*

Before we turn our attention to the lower bound for the preemptive case, we state the theorem by Sprock [12], to which we will refer in the proof. The theorem concerns an online maximization problem called the string guessing problem. An algorithm has to guess the binary string, symbol by symbol, and its gain is defined as the number of correctly guessed symbols.

Theorem 4.7 (Sprock [12])**.** *Consider an input string of length* $n \in \mathbb{N}$ *for string guessing problem. The minimum number of advice bits that can guarantee some online algorithm to be correct in more than* αn *characters, for* $1/2 \leq \alpha < 1$*, is*

$$(1 + (1-\alpha)\log_2(1-\alpha) + \alpha \log_2 \alpha)n.$$

This result can be used for proving a lower bound in the following manner.

Theorem 4.8. *To achieve a competitive ratio* $c \in (1, 4/3]$ *for preemptive* DPA_1*,*

$$(1 + (1-\alpha)\log_2(1-\alpha) + \alpha \log_2 \alpha)\frac{L}{3},$$

bits of advice are necessary, where $\alpha = \frac{2-c}{c}$*.*

Proof. Suppose that $L = 3\ell$, where ℓ is an integer. We create a set of instances that consist of ℓ boxes of size 3. Each box is either of type A or B, as depicted in Figure 8. During the first phase, an algorithm has to correctly guess the type of the box, and the corresponding call of this phase has to be accepted. Theorem 4.7 states how much advice does an algorithm need if it is supposed to correctly guess α-fraction of the number of all boxes.

The gain for the optimal algorithm is in every box 2, while if the algorithm makes a mistake, its gain for the box is at most one. Therefore, the gain of the optimal algorithm on L is 2ℓ. If algorithm correctly guesses $\alpha\ell$

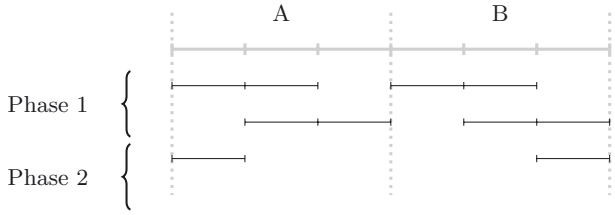


FIGURE 8. Two type of instances for lower bound of trade-off.

boxes, then gain for the algorithm is $\ell + \alpha\ell$. Thus, the competitive ratio of the algorithm with α substituted according to the theorem is

$$\frac{\text{gain}(\texttt{OPT})}{\cos(\texttt{A})} = \frac{2\ell}{\ell + \alpha\ell} = \frac{2}{1+\alpha} = \frac{2}{1 + \frac{2-c}{c}} = \frac{2c}{c+2-c} = c.$$

As follows from Theorem 4.7, the algorithm needs

$$(1 + (1-\alpha)\log_2(1-\alpha) + \alpha\log_2\alpha)\frac{L}{3},$$

advice bits to be c competitive. □

Remark 4.9. With small modifications in calculations we obtain a lower bound for DPA_ℓ. The types of boxes are the same as for DPA_1, but the gain of optimal algorithm is 3 for every box, and the gain of algorithm on incorrectly guessed box is 2. Hence, if $\alpha = (3-2c)/c$, then the algorithm achieves the following competitive ratio

$$\frac{\text{gain}(\texttt{OPT})}{\cos(\texttt{A})} = \frac{3\ell}{2\ell + \alpha\ell} = \frac{3}{2+\alpha} = \frac{3}{2 + \frac{3-2c}{c}} = \frac{3c}{2c+3-2c} = c.$$

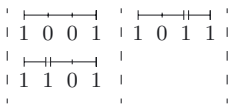
From the requirements of Theorem 4.7, the range of feasible competitive ratios is in this case $(1, 6/5]$.

The proof works equally good even in the case that advice complexity is measured with respect to n, as every box contains exactly three calls (and, therefore, n equals to L).

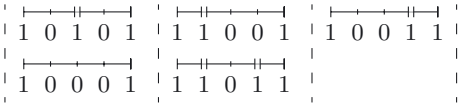
APPENDIX A. PARTITIONS OF OPT. SOLUTIONS FOR THEOREM 3.4

Here is the list of partitions of complete optimal solutions for L from 3 to 8. The optimal solutions in the same column can have the same advice (see Lem. 3.3 for reasoning). The list was obtained by implementing a brute-force algorithm to provide groups of solutions that fulfil the conditions of Lemma 3.3.

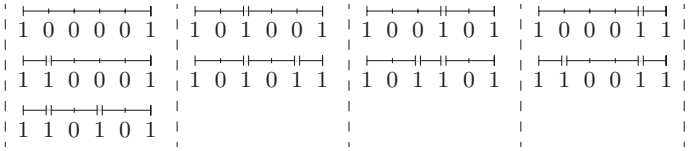
For $L = 3$, there are 2 groups.



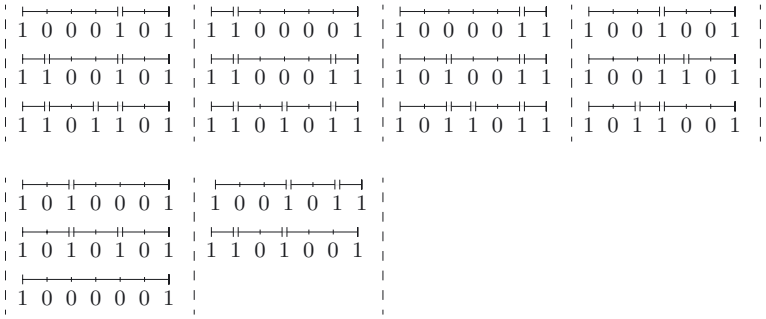
For $L = 4$, there are 3 groups.



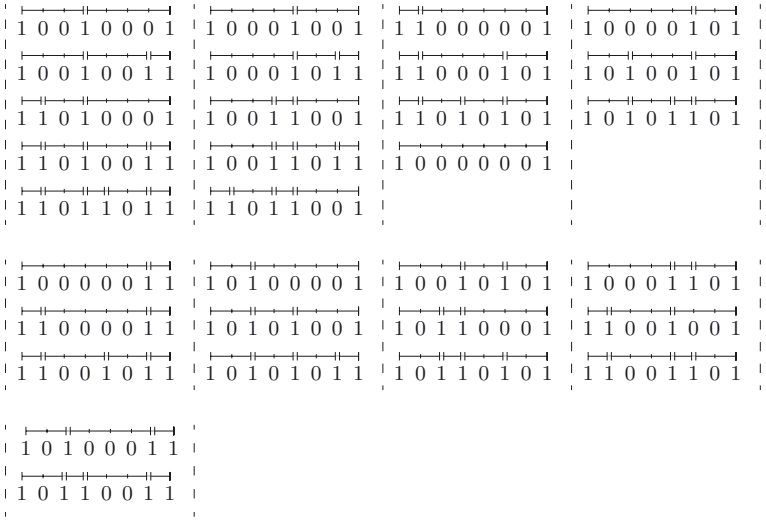
For $L = 5$, there are 4 groups.



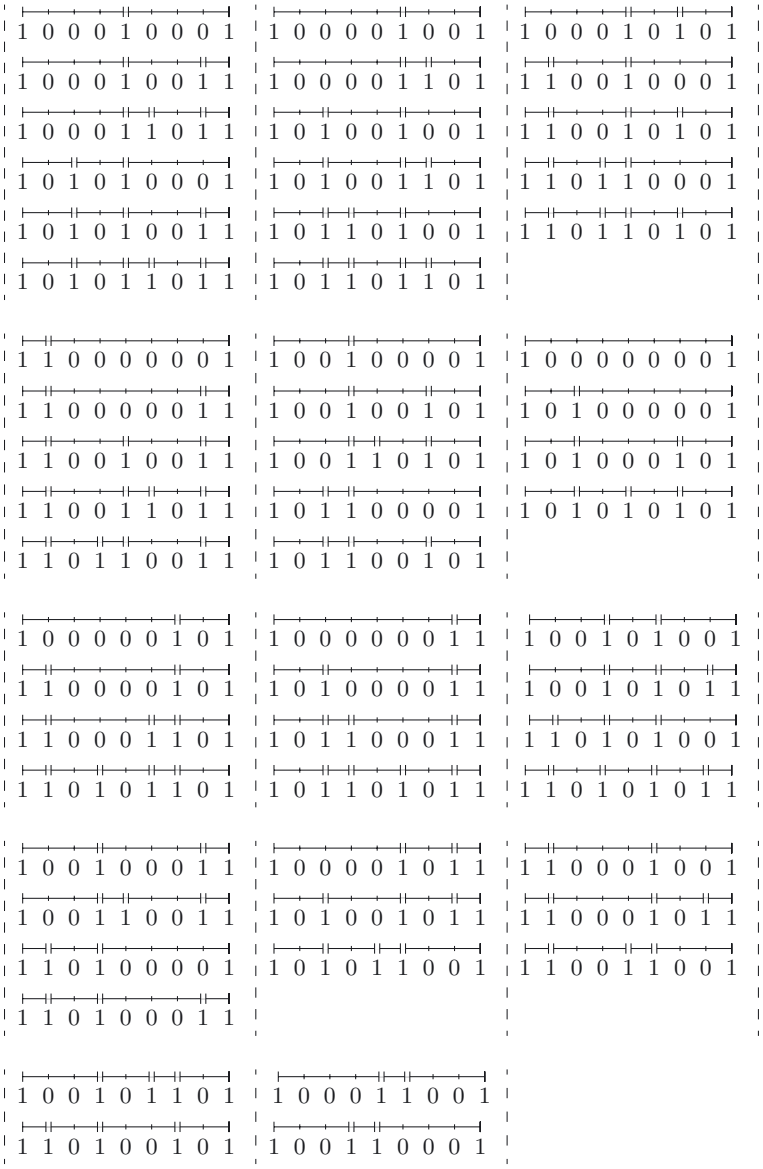
For $L = 6$, there are 6 groups.



For $L = 7$, there are 9 groups.



For $L = 8$, there are 14 groups.



References

[1] S. Albers, Online algorithms: a survey. *Math. Program.* **97** (2003) 3–26.

[2] K. Barhum, H.-J. Böckenhauer, M. Forišek, H. Gebauer, J. Hromkovič, S. Krug, J. Smula and B. Steffen, On the power of advice and randomization for the disjoint path allocation problem. In *Proc. of the 40th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2014)*. Vol. 8327 of *Lect. Notes Comput. Sci.* Springer (2014) 89–101.

[3] H.-J. Böckenhauer, D. Komm, R. Královič, R. Královič and T. Mömke, On the advice complexity of online problems. In Proc. of *Algorithms and Computation, 20th International Symposium, ISAAC 2009, Honolulu, Hawaii, USA, December 16–18, 2009*. Edited by Y. Dong, D.-Z. Du and O.H. Ibarra. Vol. 5878 of *Lect. Notes Comput. Sci.* Springer (2009) 331–340.

[4] A. Borodin and R. El-Yaniv, *Online computation and competitive analysis*. Cambridge University Press (1998).

[5] S. Dobrev, R. Královič and D. Pardubská, Measuring the problem-relevant information in input. *RAIRO: ITA* **43** (2009) 585–613.
[6] Y. Emek, P. Fraigniaud, A. Korman and A. Rosén, Online computation with advice. Special issue of ICALP'09. *Theoret. Comput. Sci.* **412** (2011) 2642–2656.
[7] R.L. Graham, Bounds for certain multiprocessing anomalies. *Bell System Technical Journal* **45** (1966) 1563–1581.
[8] D. Komm, *Advice and randomization in online computation*. Ph.D. thesis, ETH Zürich (2011).
[9] T.III Piezas, F. van Lamoen and E.W. Weisstein, "Plastic Constant" From MathWorld – A Wolfram Web Resource. Available at http://mathworld.wolfram.com/PlasticConstant.html (2015).
[10] D.D. Sleator and R.E. Tarjan, Amortized efficiency of list update and paging rules. *Commun. ACM* **28** (1985) 202–208.
[11] N.J.A. Sloane, "Padovan sequence(A000931)" From The On-Line Encyclopedia of Integer Sequences. Available at https://oeis.org/A000931 (2015).
[12] A. Sprock, *Analysis of hard problems in reoptimization and online computation*. Ph.D. thesis, ETH Zürich (2013).
[13] E.W. Weisstein, "Padovan Sequence" From MathWorld – A Wolfram Web Resource. Available at http://mathworld.wolfram.com/PadovanSequence.html (2015).

Communicated by B. Doerr.
Received November 10, 2015. Accepted September 13, 2016.