An Analysis of Call Admission Problems on Grids

Hans-Joachim Böckenhauer
ETH Zürich
Department of Computer Science
hjb@inf.ethz.ch

Dennis Komm
ETH Zürich
Department of Computer Science
dennis.komm@inf.ethz.ch

Raphael Wegner
ETH Zürich
Department of Computer Science
rwegner@student.ethz.ch

Abstract

This paper analyzes the call admission problem on grids, where a central authority receives requests that two of the computers in the network arranged as a grid structure want to communicate. The central authority can then, for every request, either satisfy it by establishing one of the possible connections in the grid, or reject the request. Thereby, the requests have to be answered in an online fashion, every connection is permanent, and connections have to be edge-disjoint. We are particularly interested to examine how much information about the future the central authority needs in order to compute an optimal solution or a solution of some given quality compared to the optimal solution. Therefore, the central authority can read an arbitrary number of bits from a tape providing the most helpful bit string, called advice.

Our results show that, without advice, the central authority cannot perform satisfactorily well, and we establish a lower bound linear in $|E|$ for the number of advice bits needed for near-optimal solutions, where $|E|$ denotes the number of edges in the grid. Furthermore, concerning optimality, we were able to prove nearly tight bounds of at least $0.94|E|$ and at most $3|E|$ advice bits. In addition, we state another upper bound in the number of requests k and the number of vertices $|V|$ in the grid graph of $\lceil \log_2(5) \cdot k + \log_2(3) \cdot |V| \rceil + \lceil 2 \log_2(k) \rceil$ bits of advice, which is stronger for a small number of requests.

A similar problem concerning a path graph as underlying network served as inspiration for this paper, and has already been studied in great detail.

2012 ACM Subject Classification Online algorithms, Graph algorithms analysis

Keywords and phrases disjoint path allocation, call admission, advice complexity, competitive analysis

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

Imagine you are the administrator of a computer network, where each computer can request a connection to any of the other ones from a central authority, which immediately either satisfies the request or rejects it. Of course the different properties of network topologies are manifold, and the priorities among them are not the same in every case. However, your main concern is to be able to establish connections for the largest possible portion of requests. As conventional in computer science, we examine the worst case, i.e., the case where the

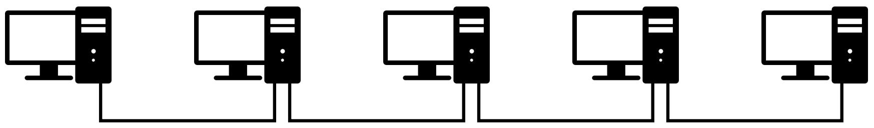

Figure 1 A path network containing five computers.

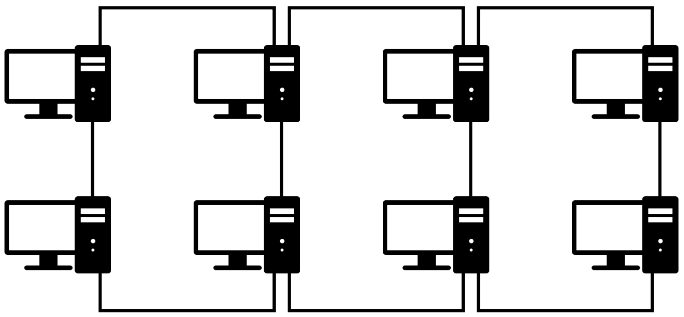

Figure 2 A (2×4)-grid network.

requests are the most unfavorable. Hence, given a specific topology, the question of how good this topology suits our problem arises.

Now, for simplicity let us assume that once a connection between two computers has been established, the connection is fixed, i.e., it does not get terminated or changed in any way. Further, every wire can only be utilized for one connection, so there are no two connections sharing a wire. In the literature, this informally described setting is known as the *disjoint path allocation problem* or as the *call admission problem*.

So far, the disjoint path allocation problem has been predominantly studied on simple paths as network topologies (see Figure 1). Thus, in this paper we use this term when referring to the problem on a path network, although the disjoint path allocation problem actually comprises the complete variety of network topologies. Based on the research on path networks, we aim to analyze the problem on grid networks, and for distinction we specify this as the *call admission problem on grids* (see Figure 2).

Note that some network topologies naturally perform well without the central authority being remarkably powerful, e.g., for the complete graph as underlying network, obviously all requests can be satisfied by just using the direct connection between the computers in every request. Unfortunately, without any further assumptions on the power of the central authority besides being an algorithm, it can be proven that both, paths and grids, as network perform quite badly, so the results are not very satisfactory. Therefore, the concept of *advice* has been introduced. There, the central authority has access to some kind of mighty oracle that knows the future requests and can transmit any information, called advice, to the central authority. Deploying this model, we are not longer restricted to just state that some ratio of satisfied requests cannot be achieved, but can give a precise measure of how much information the central authority has to obtain from the oracle in order to be able to reach exactly this ratio of satisfied requests. Or less formally, if you are not able to utilize your network as desired, at least you want to identify how much hidden information keeps you from having a considerably successful central authority. Let us remark that in general, even knowing the amount of information, still which information this is might remain undiscovered. However, usually upper bounds on the amount of advice are proved in a constructive way, so coming back to practical applications, there may even be a possibility to gather (some of) this information (e.g., some properties about the order of requests or about the position of the computers entailed in the requests might be detectable).

Since the lack of knowledge about the future impeding a reasonably good solution unifies a lot of online problems, suitable complexity models have been invented: The fundamental idea of comparing the solutions of a specific algorithm with the best possible solutions has been introduced by Sleator and Tarjan [12], and the worst case ratio between them is called the *competitive ratio*. Further, the notion of *advice complexity* is due to Dobrev et al. [6]. However, the model has some weaknesses because the advice is given in a piecewise manner, with pieces of a finite length, which results in additional information the algorithm can exploit, but that is not accounted for by the model. Therefore, the model we consider in this paper is a refined version proposed by Hromkovič et al. [9] and Böckenhauer et al. [3]. Using this model, various online problems have already been studied, including the ski rental problem, online vertex cover problem, online independent set problem, as well as the problem motivating this paper, the disjoint path allocation problem [6, 5, 3].

For the latter, without the help of advice, no deterministic online algorithm can achieve a constant competitive ratio. More precisely, measured in the length l of the path of the network, every online algorithm accepts in the worst case at most $\frac{1}{l}$ of the requests that are satisfied by an optimal algorithm [4]. Even in case we allow for a constant number of unsatisfied requests that are ignored for the competitive ratio, Komm [10] proved that every algorithm is at least $\sqrt{l}$-competitive, i.e., accomplishes at most a ratio of $\frac{1}{\sqrt{l}}$ of granted requests. Similarly, with regards to the total number of requests k, Böckenhauer et al. [3] already showed in their initial paper about this model that no algorithm is better than $(k - \mathcal{O}(1))$-competitive, or in case of a randomized online algorithm $(\frac{k}{4} - \mathcal{O}(1))$-competitive.

In the context of deterministic online algorithms with advice, $l - 1$ bits of advice are both necessary and sufficient in order to compute an optimal solution concerning the disjoint path allocaiton problem [1]. Digressing from strict optimality, for an algorithm to obtain a competitive ratio of c, again Böckenhauer et al. [3] established that at least $\frac{k+2}{2c} - 2 \in \Omega(\frac{k}{c})$ advice bits have to be read. Later, Boyar et al. [5] proved this to be asymptotically tight, i.e., $\Theta(\frac{k}{c})$ bits of advice are necessary and sufficient for being c-competitive,. As they were able to prove similar results for various online problems, based on this, they introduced the first online advice complexity class, analogously to those prevailing in time and space complexity.

Now, for the call admission problem on grids, we aim to generalize some of the bounds for the disjoint path allocation problem and to establish further ones. Since, in contrast to a path network, on a grid network there are multiple possible paths (or even walks) that can be deliberated to satisfy a request between two computers, the online algorithm has significantly more freedom to build its solution. At the same time, there are far more possible requests that contradict each other, so considering the worst case, the construction of a solution might be remarkably more complicated. Thus, it is not clear whether more or less advice is needed in order to ensure some competitive ratio in comparison to the disjoint path allocation problem.

However, this paper will present proofs that also on the grid $G = (V, E)$, indeed advice is needed to achieve a competitive ratio constant in the grid size, and that for optimality at least almost the number of edges $|E|$ of advice bits are necessary. In addition, we show a lower bound on the number of advice bits necessary for non-optimal algorithms with a certain competitive ratio.

Concerning upper bounds, we prove that for short, horizontally or vertically aligned requests less than $|E|$ advice bits are sufficient to compute an optimal solution. In general, no more than $3|E|$ bits of advice are needed. Similarly, (partially) measured in the number of requests k, roughly $\log_2(5) \cdot k + \log_2(3) \cdot |V|$ bits of advice suffice.

Therefore, let us continue with formalizing the mentioned concepts and the problems in question.

2 Preliminaries

This chapter serves to introduce our basic conventions concerning mathematical notation, and to explain all definitions necessary to follow the proofs in Section 3.

2.1 Mathematical Notation

Throughout this paper, $\mathbb{N}$ is defined as the set of positive integers, i.e., $\mathbb{N} = \{1, 2, 3, \ldots\}$. For a set as an argument, $|\cdot|$ stands for its cardinality (and otherwise for the absolute value). The exponentiation of a set refers to the cartesian power of the set, e.g., $\mathbb{N}^3 = \mathbb{N} \times \mathbb{N} \times \mathbb{N}$. Following common conventions, $G = (V, E)$ denotes a graph (in most cases a grid), with vertex set V and edges E. If these sets are not specified in the definition of a graph G, $V(G)$ or $E(G)$ are the sets of vertices or edges, respectively. Hence, $|V(G)|$ denotes the number of vertices and $|E(G)|$ denotes the number of edges in a graph G. If G is a path, then $|E(G)|$ is its length. The degree of a vertex v is $d(v)$, the maximum degree of a graph G is $\Delta(G)$, the chromatic number, i.e., the minimum number of colors needed in a proper vertex-coloring of G, is denoted by $\chi(G)$, and $\omega(G)$ is the maximum clique size. Grid graphs will be formally defined later, but let us already note that usually m and n refer to the number of vertices in the vertical or horizontal direction, respectively. Slightly abusing notation, requests are denoted by (u, v) instead of $\{u, v\}$, in order to distinguish them from edges, so in this context exceptionally $(u, v) = (v, u)$. Furthermore, for functions $f\colon X \to Y$ and $g\colon X \to Y$ for some sets $X, Y \subseteq \mathbb{R}$, the Landau notation is used in the following sense:

$$f(x) \in \mathcal{O}(g(x)) \iff \limsup_{x\to\infty} \left|\frac{f(x)}{g(x)}\right| < \infty$$

$$f(x) \in o(g(x)) \iff \lim_{x\to\infty} \left|\frac{f(x)}{g(x)}\right| = 0$$

$$f(x) \in \Omega(g(x)) \iff \liminf_{x\to\infty} \left|\frac{f(x)}{g(x)}\right| > 0$$

$$f(x) \in \Theta(g(x)) \iff f(x) \in \mathcal{O}(g(x)) \wedge f(x) \in \Omega(g(x))$$

2.2 Online Problems and Online Algorithms

We assume that the concept of offline problems and offline algorithms is known to the audience (otherwise see Hromkovič [8]), so we omit their definition and start directly with their online counterparts. Some of the definitions are based on Komm [10], which we recommend for further explanations.

The essential underlying concept of the online setting is that an online algorithm receives a request r_i of an instance of the appropriate online problem, and has to answer this request with an answer a_i before it gets the next one. So viewing the communication, requests and answers strictly alternate. Of course, at any point in time the online algorithm is only aware of the requests revealed so far without any knowledge of the future, so a response a_i may only depend on $r_1, r_2, \ldots, r_i$. In order to formalize this, an online problem has to define the feasible sequences of requests and the corresponding suitable sequences of answers.

▶ **Definition 1** (Online Problem)**.** An online problem Π consists of a set of instances $\mathcal{I}(\Pi)$ and, for every such instance $I \in \mathcal{I}(\Pi)$, a set of feasible solutions $\mathcal{S}(I)$. An instance I is a sequence of requests $(r_1, r_2, \ldots, r_k)$, and likewise a corresponding solution $S \in \mathcal{S}(I)$ is defined to be a sequence of answers $(a_1, a_2, \ldots, a_k)$, for some $k \in \mathbb{N}$.

For reasons of readability and convenience, we often hide the dependency on Π and simply write $\mathcal{I}$ instead of $\mathcal{I}(\Pi)$, since Π should be clear from the context, or we define $\mathcal{I}$ to be a restricted set of instances we consider for a proof.

▶ **Definition 2** (Online Optimization Problem)**.** An online optimization problem Π is an online problem together with a goal function and a measurement function. The goal function is either $\max\{\cdot\}$, in this case Π is an online maximization problem, or $\min\{\cdot\}$, then Π is called an online minimization problem. The measurement function can be any function $f(I, S) \in \mathbb{R}$, where $I \in \mathcal{I}(\Pi)$ and $S \in \mathcal{S}(I)$, and is referred to as $\text{gain}(I, S)$ if the objective is to maximize, and as $\text{cost}(I, S)$ in case of an online minimization problem.

Given some $I \in \mathcal{I}(\Pi)$, $\text{Opt}(I) \in \mathcal{S}(I)$ is defined to be an optimal solution if and only if $\text{gain}(I, \text{Opt}(I)) = max\{\text{gain}(I, S) \mid S \in \mathcal{S}(I)\}$, or $\text{cost}(I, \text{Opt}(I)) = min\{\text{cost}(I, S) \mid S \in \mathcal{S}(I)\}$, respectively.

Consequently, the term *optimal algorithm* will be used for an algorithm that computes an optimal solution on every instance $I \in \mathcal{I}(\Pi)$, and hence we abbreviate it with Opt. Since we mostly argue about the solution $\textsc{Alg}(I)$ of a concrete deterministic algorithm $\textsc{Alg}$ on some instance I, we usually write $\text{gain}(\textsc{Alg}(I))$ or $\text{cost}(\textsc{Alg}(I))$, instead of $\text{gain}(I, \textsc{Alg}(I))$ and $\text{cost}(I, \textsc{Alg}(I))$. Besides, throughout this paper we actually only examine maximization problems in more detail, so we state the definitions concerning minimization problems rather for matter of completeness.

As already mentioned in the introduction, the online problem that motivated the topic of this paper is called the disjoint path allocation problem, and its informal description will be made precise next. For this, note that the length of a path graph already defines the whole path graph (up to isomorphism).

▶ **Definition 3** (Disjoint Path Allocation Problem)**.** The disjoint path allocation problem is an online maximization problem denoted by Π_{DPA}. The set of instances $\mathcal{I}(\Pi_{\text{DPA}})$ consists exactly of all possible $I = (r_1, r_2, \ldots, r_k)$, such that the first request r_1 contains a length $l \in \mathbb{N}$, which defines a path graph $G = (V, E)$, and additionally, every request r_i, $1 \leq i \leq k$, consists of a pair of two vertices $(u_i, v_i) \in V \times V$, $u_i \neq v_i$. Further, the pairs of vertices in the requests in I have to be pairwise distinct.

We say that two requests r and r' of I contradict if and only if the path p in G between the pair of vertices of r, and the path p' in G between the pair of vertices of r', are not edge-disjoint.

A corresponding solution $S = (a_1, a_2, \ldots, a_k) \in \mathcal{S}(I)$ is a bit string, such that, for all i, j, $1 \leq i, j \leq k$, $a_i = 1 \wedge a_j = 1$ implies that r_i and r_j do not contradict.

The measurement function returns the number of satisfied requests, i.e., $f(I, S) = |\{a_i \mid 1 \leq i \leq k \wedge a_i = 1\}|$, where $S = (a_1, a_2, \ldots, a_k) \in \mathcal{S}(I)$.

Sometimes, this problem is called the *DPA problem* or just *DPA*. In the following, we say that the requests of some instance I are *asked*, *demanded* or *requested*, and, if $a_i = 1$ in some solution $S = (a_1, a_2, \ldots, a_k) \in \mathcal{S}(I)$, we say that the solution (or the algorithm that computes the solution) *satisfies*, *grants*, *permits*, *accepts* or *admits* the corresponding request r_i. Otherwise, it gets *rejected* or similar.

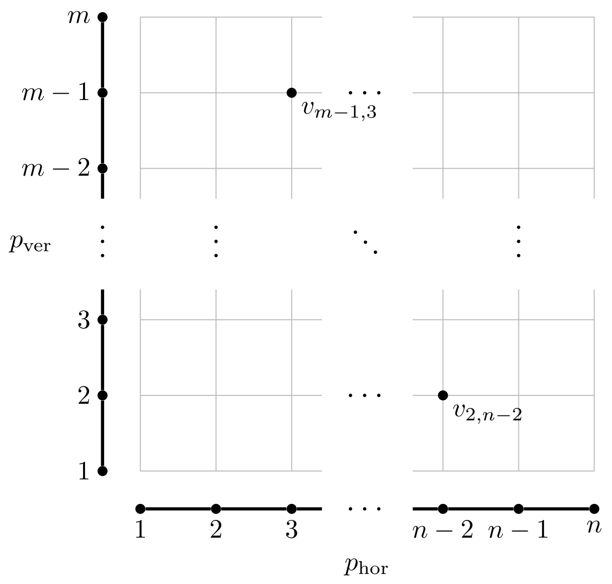

Figure 3 The construction of a grid through the cartesian product of two paths p_{ver} and p_{hor}.

Since the next problem concerns more general graphs, we define the *length of a request* r to be the length of the shortest path between the pair of vertices in r, and conversely, such path which is not of minimal length is referred to as a *detour*.

For a rigorous definition of the call admission problem on grids, we first define the underlying network on which we want to grant connections, the grid.

▶ **Definition 4** (Grid). For $m, n \in \mathbb{N}$, an $(m \times n)$-grid $G = (V, E)$ is the cartesian product of two paths $p_{\text{ver}} = (V_m, E_m)$, and $p_{\text{hor}} = (V_n, E_n)$, where $V_k = \{1, 2, \dots, k\}$ and $E_k = \{\{1,2\}, \{2,3\}, \dots, \{k-1, k\}\}$. Therefore, $V = \{v_{i,j} \mid (i,j) \in V_m \times V_n\}$ and $E = \{\{v_{a,b}, v_{x,y}\} \mid (a = x \wedge \{b,y\} \in E_m) \vee (b = y \wedge \{a,x\} \in E_n)\}$.

Note that p_{ver} is of length $m-1$ and p_{hor} is of length $n-1$. In illustrations and descriptions, $v_{1,1}$ is chosen to be the lower-left corner, whereas $v_{m,n}$ is the upper-right corner, as Figure 3 depicts.

Similar to a path, obviously a grid is completely specified through the lengths m and n, and with this observation we have everything to define our primary object of study.

▶ **Definition 5** (Call Admission Problem on Grids). The call admission problem on grids (short CAPG) is an online maximization problem denoted by Π_{CAPG}. The set of instances $\mathcal{I}(\Pi_{\text{CAPG}})$ consists exactly of all possible $I = (r_1, r_2, \dots, r_k)$, such that the first request r_1 contains two lengths $m, n \in \mathbb{N}$, which define an $(m \times n)$-grid $G = (V, E)$. Additionally, every request r_i, $1 \leq i \leq k$, consists of a pair of vertices $(u_i, v_i) \in V \times V$, $u_i \neq v_i$. Further, the pairs of vertices in the requests in I have to be pairwise distinct.

We say a set of paths is contradicting if and only if the paths are not pairwise edge-disjoint. A set of requests $\{r'_1, r'_2, \dots, r'_t\}$ of I is contradicting if and only if there is no set of paths $\{p_1, p_2, \dots, p_t\}$ in G that is not contradicting, and such that p_j is a path between the pair of vertices of request r'_j, for $1 \leq j \leq t$.

A corresponding solution $S = (a_1, a_2, \dots, a_k) \in \mathcal{S}(I)$ is an element from $(P \cup \{0\})^k$, where P denotes the set of all paths in G, such that for all $a_i \neq 0$, a_i connects the vertices in r_i and $\{a_i \mid 1 \leq i \leq k \wedge a_i \neq 0\}$ is not contradicting.

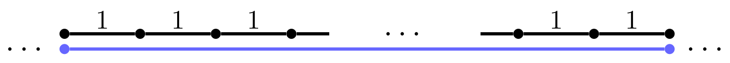

Figure 4 Illustration of a greedy online algorithm for DPA which satisfies the first request (in blue) and has to reject all subsequent small requests of length 1.

The measurement function returns the number of satisfied requests, i.e., $f(I, S) = |\{a_i \mid 1 \le i \le k \wedge a_i \ne 0\}|$, where $S = (a_1, a_2, \ldots, a_k) \in \mathcal{S}(I)$.

While for the DPA problem it has been sufficient to state whether a request should be satisfied or not, because there is only one unique way to satisfy a request on a path, this is not the case for the call admission problem on grids. Here, for each of the respective requests, there may be multiple paths in the grid to grant it. Hence, if a solution would only contain whether a request has been satisfied or not, the algorithm would not need to fix any paths at all, but it only has to ensure the existence of some non-contradicting set of paths connecting the granted requests. However, thinking of the real-world scenario behind CAPG, it seems much more natural that the connection in the network should already be established and therefore be fixed from the time of acceptance.

Besides, in some proofs we consider a (partial) solution in which the paths that satisfy some requests are already fixed and use the term *contradicting* to make clear that these fixed paths are not pairwise edge-disjoint.

Now that we have seen two examples of online problems, we turn to online algorithms, which, given an instance of the respective online problem, compute some suitable solution.

▶ **Definition 6** (Online Algorithm)**.** An online algorithm $\textsc{Alg}$ for an online problem Π is an algorithm that, given an instance $I = (r_1, r_2, \ldots, r_k) \in \mathcal{I}(\Pi)$, computes a feasible solution $\textsc{Alg}(I) = (a_1, a_2, \ldots, a_k) \in \mathcal{S}(I)$, where a_i may only depend on $r_1, r_2, \ldots, r_i$.

For example, a greedy online algorithm $\textsc{Alg}$ for the DPA problem on an instance I would check, for every request it receives, whether it contradicts any previously satisfied request and, if this is not the case, satisfies it. Or more formally, for an instance $I = (r_1, r_2, \ldots, r_k)$, $\textsc{Alg}$ would output the solution $\textsc{Alg}(I) = (a_1, a_2, \ldots, a_k)$, where $a_i = 1$ if and only if for all $j < i$, $a_j = 1$ implies that r_i and r_j do not contradict.

Note that for CAPG there is no unique greedy online algorithm $\textsc{Alg}$, since it is not clear which path should be used in order to satisfy a request. However, using any ordering Q of the paths in the grid, we could still define that $\textsc{Alg}$ accepts a request r_i using a path p_i between its vertices if and only if p_i is the first path in Q where $\{a_j \mid 1 \le j < i \wedge a_j \ne 0\} \cup \{p_i\}$ is not contradicting.

Consider an instance of DPA in which first a long request is asked and then this request gets partitioned into requests each of length 1 which are demanded next. The greedy algorithm $\textsc{Alg}$ from above would satisfy the first request and then all the other requests have to be rejected because they contradict (see Figure 4).

Intuitively, $\textsc{Alg}$ does not seem to perform very well because, on some instances, it obviously satisfies only very few requests, and in our intuitive understanding of “performing well”, we compare it to an optimal solution that might be able to grant many more requests, as in this example all small requests. This notion of performance is captured by the following definition.

▶ **Definition 7** (Competitive Ratio)**.** Let $\textsc{Alg}$ be an online algorithm for an online optimization problem Π. Then, for $c \in \mathbb{R}$, $c \ge 1$, $\textsc{Alg}$ is c-competitive if there is a constant $\alpha \ge 0$, such that for all $I \in \mathcal{I}(\Pi)$

$$\text{gain}\,(\text{Opt}(I)) \le c \cdot \text{gain}\,(\textsc{Alg}(I)) + \alpha$$

in case Π is a maximization problem, and

$$\text{cost}\,(\textsc{Alg}(I)) \le c \cdot \text{cost}\,(\text{Opt}(I)) + \alpha$$

if Π is a minimization problem. In case this holds for $\alpha = 0$, Alg is called strictly c-competitive. The competitive ratio of Alg then is $c_{\textsc{Alg}} = \inf\{c \mid \textsc{Alg} \text{ is } c\text{-competitive}\}$.

The additive constant α only serves to compensate for some constant advantage of the optimal algorithm, because we are interested in the asymptotic correlation, rather than in the ratio in some corner cases.

For the ease of explanation, we will at times slightly abuse terminology and also speak about the competitive ratio as the performance of an algorithm on a single, specific instance. Moreover, if there is no constant c, such that some algorithm is c-competitive, in some literature this is expressed as being “not competitive” or that “there is no competitive ratio”, or similar [10, 4]. However, even if $c = c(\cdot)$ is a function, e.g., of the grid size or the number of requests, the ratio concerning an optimal solution can still be of interest, so in the respective cases we prefer to formulate this as something in the sense of *the algorithm is* $c(\cdot)$*-competitive*.

Especially since not all papers were mindful of this issue, let us remark that in case that the grid size itself is not part of the instance, α can be a constant with respect to the instances, and yet depend on the grid size. Then, for a given grid size there are only finitely many instances, and thus, there always would exist a sufficiently large constant α, such that the algorithm is 1-competitive, independently of what the algorithm actually does. This is the reason for the length of the path or the size of the grid, respectively, to be contained in the first request in the definitions of DPA and CAPG.

2.3 Online Algorithms with Advice

The two considered online problems, and almost all others of interest, seem to be too hard to find any online algorithm that achieves a reasonably decent competitive ratio, let alone optimal solutions. But not the lack of our abilities constitutes the barrier. Instead, there rather simply do not exist satisfactory ones, even independently of the computational power at disposal. Hence, classical measures of complexity do not really capture the hardness of these problems and thus are not as expressive as desired. What is inevitably needed to compute an adequate solution is more information about the upcoming requests of the future when deciding on the current one. Therefore, we introduce the concept of advice, which brings a string into play that provides the most helpful information about the (future) requests to our online algorithm. Basically, our online algorithms have a binary string at hand from which they can read bits and then decide accordingly. If this string, called advice (string), is designed mindful of the particular algorithm as well as the complete instance that will be provided, we obviously can improve tremendously. Thus, very naturally we will measure the hardness of the problem in the number of bits of advice an algorithm has to read, i.e., the amount of needed information about the future, in order to obtain some competitive ratio, instead of time or space demands of the algorithm. Obviously, then, in contrast to resources of time or space, given enough information, an optimal solution is computable, so this alternative measure of complexity is sensible for online problems.

Note that we define this setting only for optimization problems, since clearly the concept of advice is primarily meaningful in the context of some desired quality of the solution.

So far, we defined the competitive ratio as the infimum ratio over all instances $I \in \mathcal{I}$, but since, as we will see, the advice is chosen by some oracle, we now let an adversary choose the instance, so we have "good" and "evil" counterparts which make explanations much more illustrative. If the adversary is powerful enough to choose the worst instance $I \in \mathcal{I}$, this then is the same as quantifying over all instances. The appropriate adversary model is called the *oblivious adversary*.

▶ **Definition 8** (Oblivious Adversary)**.** The oblivious adversary knows the online optimization problem Π, the online algorithm $\textsc{Alg}$ with advice, and the oracle. In particular, it is aware of the given advice and hence of all deterministic actions of $\textsc{Alg}$. Still, the adversary has no knowledge about the values on the random tape from which $\textsc{Alg}$ can read bits in case $\textsc{Alg}$ is a randomized algorithm.[1] The oblivious adversary is assumed to have unbounded computational power. Under these assumptions, the adversary chooses one instance $I \in \mathcal{I}(\Pi)$ such that the competitive ratio of $\textsc{Alg}$ on this instance I is maximized.

Formally, the adversary chooses the complete instance in advance of any asked or served requests. Nevertheless, it knows the algorithm which the instance will be handed to, so in the deterministic case, for a more intuitive description, to a certain extent, this can be seen as an interactive game where the adversary reacts to the actions of the algorithm and constructs the instance alongside. However, in randomized scenarios, or in reduction proofs where an algorithm impersonates the role of the adversary, we have to consult the precise definition.

Next, we specify the capabilities of the good pendant, the oracle.

▶ **Definition 9** (Oracle)**.** The oracle knows the online optimization problem Π, the online algorithm $\textsc{Alg}$ with advice, and the complete instance $I \in \mathcal{I}(\Pi)$ which is given to $\textsc{Alg}$. Still, the oracle has no knowledge about the values on the random tape from which $\textsc{Alg}$ can read bits in case $\textsc{Alg}$ is a randomized algorithm. The oracle is assumed to have infinite computational power. Under these assumptions, the oracle writes an infinite advice string on the advice tape of $\textsc{Alg}$, such that the competitive ratio of $\textsc{Alg}$ on I is minimized.

As already indicated, an online algorithm with advice needs access to an additional tape on which the oracle writes the advice string. Except for being able to read and use its advice, the algorithm is subject to the same conditions as before.

▶ **Definition 10** (Online Algorithm with Advice)**.** An online algorithm $\textsc{Alg}$ with advice for an online optimization problem Π is an algorithm that has access to an infinitely long binary advice tape with advice bits $b_1, b_2, \ldots$ of which $\textsc{Alg}$ can read an arbitrary long, but finite prefix. Given an instance $I = (r_1, r_2, \ldots, r_k) \in \mathcal{I}(\Pi)$, $\textsc{Alg}$ then computes a corresponding solution $\textsc{Alg}(I) = (a_1, a_2, \ldots, a_k) \in \mathcal{S}(I)$, where a_i may only depend on $r_1, r_2, \ldots, r_i$ and the bits of advice $b_1, b_2, \ldots, b_{q_i}$ it read so far. If $\textsc{Alg}$ reads at most the first $q(\cdot)$ bits of advice, $q(\cdot)$ is the advice complexity of $\textsc{Alg}$.

Note that $q(\cdot)$ can be a function of any parameter of the instance (or even the whole instance), but usually, we measure it either in the number of requests of the instance, or in the size of the considered graph G, e.g., in $|V|$ or $|E|$. Of course, we are interested in finding an advice complexity that is as small as possible in the respective parameter.

By defining the advice tape of an algorithm $\textsc{Alg}$ to be infinitely long, $\textsc{Alg}$ cannot extract any information from the length of the advice string, and thus, the number of advice bits

[1] For a formal definition of randomized algorithms see [10]

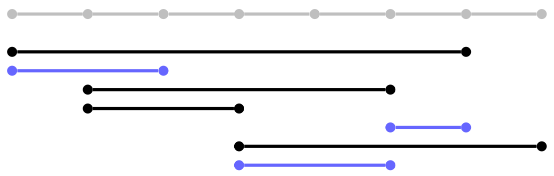

Figure 5 Example instance for DPA on a path of length 7 (in gray). If 0100101 is the prefix on the advice tape, ALG satisfies the blue requests, which constitute one of the optimal solutions.

read by the algorithm measures the whole information that ALG needs about the future requests.

Of course, the ulterior motive is to state results about advice complexity in the context of some achieved competitive ratio, and obviously a good competitive ratio and little amount of advice compete against each other in most cases. Therefore, let us define the competitive ratio for online algorithms with advice.

▶ **Definition 11** (Competitive Ratio with Advice). Let ALG be an online algorithm with advice for an online optimization problem Π. Then, for $c \in \mathbb{R}$, $c \geq 1$, ALG is c-competitive if there is a constant $\alpha \geq 0$, such that, for every instance $I \in \mathcal{I}(\Pi)$ chosen by the oblivious adversary, and an advice string $b_1, b_2, \ldots$ chosen by the oracle,

$$\text{gain}\left(\text{Opt}(I)\right) \leq c \cdot \text{gain}\left(\text{ALG}(I)\right) + \alpha$$

in case Π is a maximization problem, and

$$\text{cost}\left(\text{ALG}(I)\right) \leq c \cdot \text{cost}\left(\text{Opt}(I)\right) + \alpha$$

if Π is a minimization problem. In case this holds for $\alpha = 0$, ALG is called strictly c-competitive. The competitive ratio of ALG then is $c_{\text{ALG}} = \inf\{c \mid \text{ALG is } c\text{-competitive}\}$.

Here the same terminology conventions as for the competitive ratio without advice apply. For a better comprehension, let us examine another example for DPA. Consider an algorithm ALG with advice that, on every request r_i, reads one bit b_i of the advice tape and, if $b_i = 1$, it satisfies r_i (recall that for DPA there is only one unique possibility to grant a request), and else ALG rejects r_i. We know that the oracle provides the best possible string of advice, so we are safe to assume that, according to some optimal solution S, it writes a 1 for every request that is granted in S and a 0 for every non-satisfied request on the advice tape, since then ALG obviously computes the optimal solution S, and thus is 1-competitive (see Figure 5).

If k is the number of requests in an instance, ALG reads exactly k bits of advice and the advice complexity is therefore $q(k) = k$. Although not convenient in this case, we could measure the advice complexity with respect to the length l of the underlying path as well: Since all requests are pairwise distinct, there are at most l requests of length 1, $l-1$ of length 2 and so on, i.e., $\sum_{i=1}^{l} i = \frac{l(l+1)}{2}$ requests in an instance. Hence, with respect to l, the advice complexity is $q(l) = \frac{l(l+1)}{2}$.

2.4 Facts and Remarks

In particular, there are some small gadgets that will repeatedly come to use, so we want to introduce them here in order to safely assume they are known in the following proofs.

▶ Fact 1. In an $(m \times n)$-grid $G = (V, E)$, there are $|V| = mn$ vertices and $|E| = m(n-1) + n(m-1)$ edges, so $|E| = 2|V| - m - n$.

Moreover, for vertex-coloring a clique properly, all vertices in the clique have to be colored using pairwise distinct colors, and conversely, for a vertex v of degree $d(v)$, we can color all adjacent vertices with $d(v)$ pairwise distinct colors and v itself with one more. Therefore, we obtain the subsequent fact.

▶ Fact 2. For every graph G we have $\omega(G) \leq \chi(G) \leq \Delta(G) + 1$.

▶ Remark. If some lower bound only holds for finitely many instances, then clearly the lower bound does not grow in e.g., the grid size or the number of requests. On the other hand, since there are only finitely many instances for every grid size (or number of requests), in case that it holds for infinitely many instances, there is no largest grid size (or number of requests) for which the lower bound applies, so the adversary can choose an arbitrarily large grid size (or number of requests) for which the lower bound holds. Therefore, in order to be able to establish some lower bound as a function of the grid size (or the number of requests), it is sufficient to prove it for any infinite subset of instances. In particular, we can, without loss of generality, make assumptions on the parity of m and n, the ratio between m and n, and we may expect the grid or the number of requests to be sufficiently large. Thereby we avoid a lot of tedious rounding and do not need to consider every corner case.

▶ Remark. Occasionally, we want to pass some value $x \in \mathbb{N}$ to an algorithm Alg using parts of the string of advice. We can encode x in binary with $\lceil \log_2(x) \rceil$ bits. However, just writing this encoding on the advice tape is usually not sufficient, since Alg cannot know where the encoding on the advice tape ends, so it cannot decode the value (recall that the tape is considered to be infinite). There are multiple ways to circumvent this issue, for example, the oracle can use the first $\lceil \log_2(x) \rceil$ odd positions on the advice tape to encode x, but write a 0 at the even positions in between, meaning that the encoding still continues, and a 1 behind the encoding, indicating its end. Then, Alg is able to reconstruct x reading $2\lceil \log_2(x) \rceil$ bits of advice [10].

3 Results

3.1 Lower Bounds

First, we focus on lower bounds and establish some for CAPG that are similar to those for DPA, just for general grid sizes. Of course, since paths are corner cases of grids, DPA is a special case of CAPG, and since the adversary can choose the grid size, the respective lower bounds concerning DPA already apply for CAPG as well. However, we are interested in learning which lower bounds hold for general grids, and how they depend on the grid sizes. In fact, with some amount of additional reasoning, most of the examined lower bounds can be generalized to arbitrary grid sizes.

Komm [10] has already proved that there is no online algorithm with a constant competitive ratio for DPA, i.e., for CAPG on a $(1 \times (l+1))$-grid. The idea of the proof is to first request up to $\sqrt{l}$ many requests of the form $(v_{1,1}, v_{1,\sqrt{l}+1})$, $(v_{1,\sqrt{l}+1}, v_{1,2\sqrt{l}+1})$, …, $(v_{1,(\sqrt{l}-1)\sqrt{l}+1}, v_{1,l+1})$, just until one has been granted.[2] If none at all is granted, the adversary stops and the competitive ratio cannot be constant, since the maximal number of requests grows in l. In case one of these requests has just been satisfied, the adversary partitions this request into $\sqrt{l}$ further requests, each of length 1 (see Figure 6). So, if $(v_{1,j}, v_{1,\sqrt{l}+j})$ has been granted, then the adversary asks for $(v_{1,j}, v_{1,j+1})$, $(v_{1,j+1}, v_{1,j+2})$, …, $(v_{1,\sqrt{l}+j-1}, v_{1,\sqrt{l}+j})$. All these

[2] Technically speaking, the first request additionally contains the size of the grid.

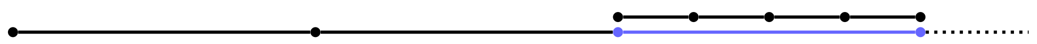

Figure 6 Example of a path of length 16, where the third request has been granted (blue). Black lines depict non-satisfied requests and dotted lines indicate those which have not been asked anymore.

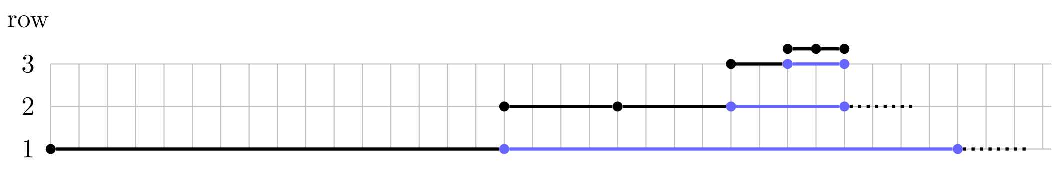

Figure 7 Instance on a grid of size 3×257.

requests cannot be satisfied anymore, because they overlap with the previously satisfied request. Hence, only one request gets satisfied by any given online algorithm. However, given this instance, one could have just permitted the requests of length one (and all non-contradicting requests, if present). Hence, again, the competitive ratio is increasing in l and therefore, in conclusion, there is no constant competitive ratio in this setting.

Now, this idea can be extended to grids very naturally: Assume a grid of size $(k \times (l+1))$ is given, for some constant $k \in \mathbb{N}$. Then, we just play the game recursively on every row of the grid. Therefore, we partition the requests into phases $P_1, P_2, \ldots, P_{k+1}$. First, in phase P_1 the adversary asks at most $\sqrt{l}$ many requests of length $\sqrt{l}$ on the first row, until one of them is accepted, just like before. So explicitly the requests are given by $(v_{1,1}, v_{1,\sqrt{l}+1})$, $(v_{1,\sqrt{l}+1}, v_{1,2\sqrt{l}+1})$, $\ldots$, $(v_{1,(\sqrt{l}-1)\sqrt{l}+1}, v_{1,l+1})$. However, note that in contrast to before, there are various walks of different lengths to satisfy the request, but that of minimal length is unique. Then, for phase P_i, where $1 < i \leq k$, we pretend the grid has diminished in size, in the sense that all rows below do not exist anymore and the length of the grid is only the length of the previous request. More rigorously, let the previously satisfied request of phase P_{i-1} start at $v_{i-1,j}$ and have length l_{i-1}. Then, this corresponds to a grid of height $k-(i-1)$ and length l_{i-1}, so the requests $(v_{1,1}, v_{1,\sqrt{l_{i-1}}+1})$, $(v_{1,\sqrt{l_{i-1}}+1}, v_{1,2\sqrt{l_{i-1}}+1})$, $\ldots$, $(v_{1,(\sqrt{l_{i-1}}-1)\sqrt{l_{i-1}}+1}, v_{1,l_{i-1}+1})$ on this new grid constitute phase P_i (see Figure 7). Since so far in every phase the length has been diminished by a square root, $l_{i-1} = l^{1/2^{i-1}}$, and in addition, all requests of this phase are in row i of the actual grid, this easily translates to the original parameters. In each phase, if one request gets admitted, all further requests of the phase are suspended and the next phase takes over. In case that in one phase none of its requests are satisfied, the whole procedure stops immediately. Note that in phase P_k we reduced the situation to that on the path, so finally, in phase P_{k+1} the adversary demands all $l_k = l^{\frac{1}{2^k}}$ many requests of length one, which cover the same part of the grid as in the previous phase, i.e., if the previously satisfied request has begun at $v_{k,j}$, then all $(v_{k,j}, v_{k,j+1})$, $(v_{k,j+1}, v_{k,j+2})$, $\ldots$, $(v_{k,l_k+j-1}, v_{k,l_k+j})$ are requested.

Assume the procedure stops early, videlicet, in some phase P_i, $i \in \{1, 2, \ldots, k\}$, none of the requests have been granted. Then, obviously fewer than k requests have been permitted altogether, but in phase P_i alone, there have been $l_i = l^{\frac{1}{2^i}}$ requests which could have been accepted in retrospect, so since k is constant but l_i grows in l, no constant competitive ratio can be reached in this case. On the other side, if in every phase P_i, $i \in \{1, 2, \ldots, k\}$, one request has been satisfied, no request of phase P_{k+1} can be permitted. This is because in all of the k rows of the grid, one request, which overlaps the requests of P_{k+1} column-wise, has already been satisfied, so all vertical edges between the endpoints of the requests of P_{k+1} are already occupied. Hence, k requests have been granted, but more than l_{k+1} could have been satisfied, so again, there is no algorithm that achieves a constant competitive ratio.

Before we state a formal proof of a more general statement, note that this argument would not hold for grids that grow arbitrarily in both dimensions, since then, the number of satisfied requests is not bounded by a constant, but instead grows in l as well, and perhaps even fast enough to achieve a constant competitive ratio. Even worse, if $k(l)$ grows too fast, we might not be able to ask a request on every row at all, since the lengths of the requests diminish considerably from row to row. However, heretofore, we did not exploit our freedom of choice to the full extent, regarding how many requests are demanded in each row, and closely tied, of which length each request is. Thus, in the following we will attempt to thoroughly exhaust the underlying concept. Notice, therefore, that the main focus has to lie on meeting two crucial conditions: First, at the end there have to be mutually exclusive requests, such that any online algorithm cannot satisfy future requests if it has already taken some other ones before. Secondly, the number of requests in each row has to be increasing in l, since we have to cover the case that none of the requests in some row gets satisfied. More precisely, we need $k(l)+1$ increasing functions, one for each row, plus one for the last phase. Be aware that from row to row the region in which previous requests have been satisfied gets partitioned further. Therefore, in order to facilitate this for the next rows, the length of each request has to be increasing sufficiently in l as well, because otherwise their number cannot grow in l anymore.

▶ **Theorem 12.** *If $k(l) \in o(\log(l))$ for some $k\colon \mathbb{N} \to \mathbb{N}$, the competitive ratio of every online algorithm without advice for CAPG on a $(k(l) \times (l+1))$-grid is at least $c \in \Omega(l^{\frac{1}{k(l)+1}} \cdot k(l)^{-1})$.*

Proof. First, note that the length of the grid is l, and by assumption $k(l)+1 \in o(\log(l))$. Hence, there is an increasing function $f(l)$ which satisfies $f(l)\cdot(k(l)+1) = \log(l)$, so we obtain $l = \exp(f(l)\cdot(k(l)+1)) = \exp(f(l))^{k(l)+1}$ and define the right-hand side to be $c(l, k+1)$ (for convenience and readability, henceforth we write k instead of $k(l)$ if possible). Furthermore, let Alg be some deterministic online algorithm for CAPG.

Now, the adversary is able to design a set of instances $\mathcal{I}$, such that each instance $I \in \mathcal{I}$ consists of phases $P_1, P_2, \ldots, P_{k+1}$ which are constructed in the following manner: In phase P_1 the adversary asks from the left to the right $c(l,1)$ many consecutive requests each of length $c(l,k)$ on the first row (where the size of the grid is included in the first request), i.e., $(k, l+1, v_{1,1}, v_{1,c(l,k)+1})$, $(v_{1,c(l,k)+1}, v_{1,2c(l,k)+1})$, $\ldots$, $(v_{1,(c(l,1)-1)c(l,k)+1}, v_{1,c(l,k+1)+1})$, as long as Alg does not satisfy any of those requests. As soon as one request has been granted, the adversary aborts the current phase and starts with the next one. Likewise, assuming in P_{i-1} the request starting at $v_{i-1,j}$ has been granted, the adversary partitions the part of the row covered by the previously satisfied request further, such that in phase P_i where $1 < i \leq k$, it demands the $c(l,1)$ many requests of length $c(l,k-i+1)$ of the form $(v_{i,j}, v_{i,c(l,k-i+1)+j})$, $(v_{i,c(l,k-i+1)+j}, v_{i,2c(l,k-i+1)+j})$, $\ldots$, $(v_{i,(c(l,1)-1)c(l,k-i+1)+j}, v_{i,c(l,k-i+2)+j})$. Again, once Alg satisfies one of these requests, P_i is aborted and the procedure continues with phase P_{i+1}. In P_{k+1} the adversary asks for all $c(l,1)$ many requests of length $c(l,0)=1$ on the same row as in phase P_k, that is, $(v_{k,j}, v_{k,j+1})$, $(v_{k,j+1}, v_{k,j+2})$, $\ldots$, $(v_{k,c(l,1)+j-1}, v_{k,c(l,1)+j})$, independent of the behavior of Alg, where again $v_{k,j}$ denotes the vertex at which the request satisfied in phase P_k starts. If in any phase none of its requests are satisfied, the adversary omits all subsequent phases and stops completely immediately after the current phase.

Let us now analyze the best competitive ratio Alg can achieve. We observe that in each of the first k phases Alg can grant at most one request. Hence, in case that Alg does not grant any request in some phase P_i, $1 \leq i \leq k$, then $\text{gain}(\text{Alg}(I)) \leq i-1$ whilst Opt can

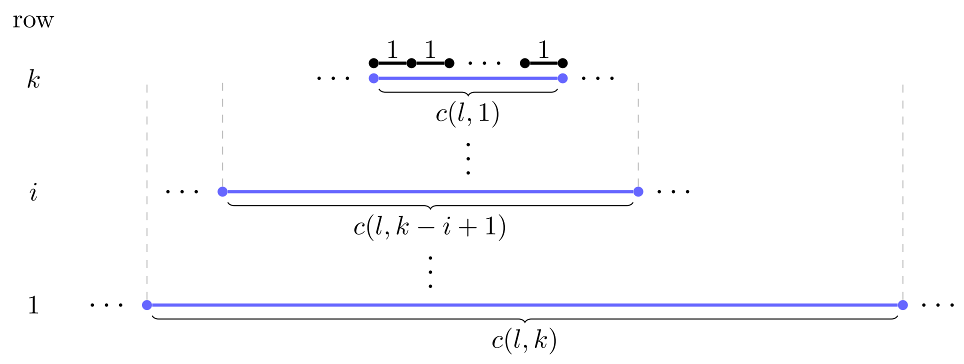

Figure 8 Schematic illustration on a general grid, where in each row of the grid a request has been satisfied. In every row there are up to $c(l,1)$ many requests from the first k phases. Then, in row k there are $c(l,1)$ many additional requests which cannot be accepted anymore.

satisfy (at least[3]) all requests that ALG has granted and additionally all requests of phase P_i, since none of them are mutually exclusive, so $\text{gain}(\text{Opt}(I)) \geq i - 1 + c(l,1)$. If this is not the case, we may assume that ALG satisfies exactly one request in any of the first k phases. However, assume without loss of generality that ALG has satisfied the request $(v_{k,j}, v_{k,j+c(l,1)})$ in phase P_k. Then, in every row all horizontal edges between column j and $j + c(l,1)$ are already taken, since there are k rows and ALG has granted exactly k requests so far, all of which start at the left side of j, i.e., in a column indexed by some $a \leq j$, and end on the right side of $j + c(l,1)$, i.e., in a column indexed by some $b \geq j + c(l,1)$. By the pigeonhole principle, we conclude that none of the $c(l,1)$ requests of phase P_{k+1} can be satisfied, so $\text{gain}(\text{ALG}(I)) = k(l)$, but Opt could have satisfied all requests of phase P_{k+1} instead of granting a request from P_k, and thus $\text{gain}(\text{Opt}(I)) \geq k(l) + c(l,1)$. Hence, the competitive ratio is at least $c \geq 1 + (c(l,1) - \alpha) \cdot k(l)^{-1}$, for some constant $\alpha \geq 0$ in compliance with Theorem 11. Since $f(l)$ is an increasing function, so is $\exp(f(l)) = c(l,1) = l^{\frac{1}{k(l)+1}}$, which concludes $c \in \Omega(l^{\frac{1}{k(l)+1}} \cdot k(l)^{-1})$. ◀

Note that by the considerations stated priorly, that in every row the number of requests has to be increasing in l, we observe that as a result the partitioning of l into the product of $k+1$ increasing functions is actually needed and the bound on the competitive ratio is maximized when these functions are about the same, i.e., $l = c(l,1)^{k+1}$ and thus $c(l,1) = l^{\frac{1}{k(l)+1}}$.

Also, let us remark that this proof generalizes the case of DPA very smoothly, in the sense that for $k = 1$ it actually boils down to $c(l,1) = l^{\frac{1}{k+1}} = \sqrt{l}$, which is precisely what has been used by Komm [10].

▶ **Corollary 13.** *For a constant $k \in \mathbb{N}$, there is no online algorithm without advice for CAPG which achieves a constant competitive ratio on a $(k \times (l+1))$-grid.*

Proof. According to Theorem 12, the bound on the competitive ratio satisfies $c \in \Omega(l^{\frac{1}{k+1}})$, because k is a constant. But clearly, for any constant $k \in \mathbb{N}$, we have $l^{\frac{1}{k+1}} \xrightarrow{l\to\infty} \infty$, and thus c cannot be bounded by a constant, so the statement follows directly. ◀

Returning to the DPA problem, as already mentioned, the partitioning of some former request r into two requests yields requests r'_1, r'_2, which are mutually exclusive to r. Since,

[3] ALG might not have satisfied the first, but the jth request in some phase, so Opt could satisfy all of the first j requests of that phase.

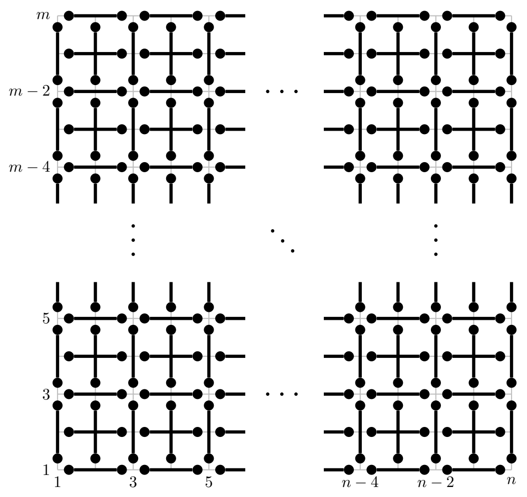

Figure 9 Grid with odd m and n, which is completely covered by requests of length 2.

given an instance containig such requests, every algorithm then needs to decide whether to grant r or any requests from $\{r'_1, r'_2\}$ instead, some advice should be necessary. This basic concept can be successively deployed in order to obtain further bounds: First, most obviously one can attempt to make statements about the necessary amount of advice to acquire an optimal solution, which in case of the DPA problem is proved to be at least half the length of the path [3]. Secondly, by reducing a well known and thoroughly studied problem, referred to as bit guessing, to DPA, statements about the amount of advice which is needed to achieve some competitive ratio can be established [14].

These proofs extend to grids nicely. Even so, there is one obstacle in the not naturally given mutual exclusivity of these kinds of requests on grids. Since in this case there are plenty of possibilities to satisfy a given request, we will pack the requests very densely to cover the grid and provide some reasoning why then any detours would contradict other requests.

▶ **Lemma 14.** *Every $(m \times n)$-grid G, with m and n odd, can be completely covered with requests of length 2, i.e., $\frac{|E(G)|}{2} = mn - \frac{m+n}{2}$ requests of length 2 can be satisfied.*

Proof. The arrangement of the requests is shown in Figure 9. We define all requests to be horizontally and vertically aligned, since in that case the shortest paths between their endpoints are unique, which we will use in subsequent proofs. Note that if m and n are odd, then the length and height (in terms of the number of edges) are even. Since every request has length 2, there are m rows with $\frac{n-1}{2}$ horizontal requests and n columns with $\frac{m-1}{2}$ vertical requests. This yields $m\frac{n-1}{2} + n\frac{m-1}{2} = mn - \frac{m+n}{2}$ requests. ◀

▶ **Theorem 15.** *Every optimal online algorithm with advice for CAPG on a grid G uses at least $\frac{|E(G)|}{2}$ advice bits.*

Proof. By Subsection 2.4 it suffices to only consider $(m \times n)$-grids G with m and n odd. We construct a set $\mathcal{I}$ of instances such that every instance $I \in \mathcal{I}$ consists of two phases P_1 and P_2: First in P_1, $\frac{|E(G)|}{2}$ requests of length 2 are demanded, which is possible due to Theorem 14. Then, in phase P_2 the adversary partitions some of the previous requests into two requests each of length 1. For instance, some request $(u, w) \in P_1$ with shortest path

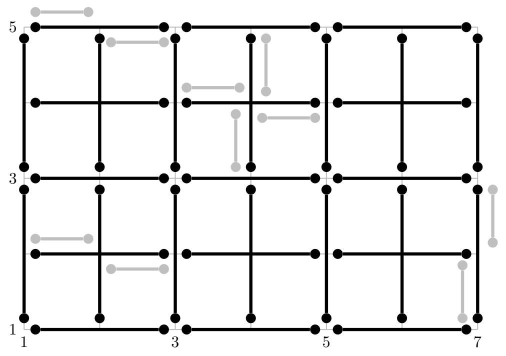

Figure 10 Some instance with ten requests in phase P_2 (depicted in gray).

u, v, w is split up into two requests $(u, v), (v, w) \in P_2$. Since P_1 is immutable for all instances and any request of P_1 can be chosen to be partitioned in P_2, there are $2^{|E(G)|/2}$ instances.

Let Alg be some optimal online algorithm with advice for CAPG. For the sake of contradiction, suppose that Alg uses $b < \frac{|E(G)|}{2}$ advice bits. Then, the set of possible advice strings has cardinality $2^b < 2^{|E(G)|/2}$, so there are strictly less different advice strings than instances in $\mathcal{I}$. Hence, by the pigeonhole principle, there have to exist two instances I_1, I_2 for which Alg receives the same advice, which in particular implies that Alg also admits the same requests in phase P_1 of both instances.

Now, let us determine what these optimal solutions look like. First, note that requests from P_2 have a shorter length than those from P_1. Hence, in case that some algorithm always uses shortest paths to satisfy requests and prefers requests from P_2 over those from P_1 (whenever contradicting), it uses a minimal number of edges of G. Therefore, if then all edges of the grid are used, the number of granted requests is maximal, i.e., the algorithm is optimal. Note that this indeed can be attained, since according to Theorem 14 the requests of P_1 (meaning the shortest paths to satisfy them) already cover G entirely and for every request-pair of P_2, which partitions a previous request r, we can satisfy these two instead of r, by only using the same edges as r did (again with shortest paths).

Note that using detours never makes sense, since any detour would use more edges, i.e., less requests could be accepted. Hence, all optimal solutions use solely shortest paths to satisfy requests. Moreover, although not true in general, still in this setting the shortest paths to satisfy these requests are unique. In conclusion, then, there is only one unique optimal solution for every instance.

Furthermore, observe that the optimal algorithm satisfies exactly $|P_1| + \frac{|P_2|}{2}$ requests, since it just permits as many requests from P_1 as possible under the constraint that it can satisfy all further upcoming requests from P_2, i.e., whenever it does not satisfy a request from P_1 then it satisfies two requests from P_2 instead, and all requests from P_2 are satisfied.

However, I_1 and I_2 differ in at least one request, which has to be from P_2. In addition, in phase P_1 Alg permits the same requests for both instances, because the same advice is given, so for the common prefixes of the instances Alg takes the exact same decisions. Thus, for one instance, a request from P_2 contradicts and cannot be satisfied, although the optimal solution would admit all requests from P_2, so we arrive at a contradiction and Alg cannot be optimal. This concludes the statement. ◀

As mentioned beforehand, this construction can be employed to prove lower bounds beyond strict optimality, i.e., to make statements about the number of advice bits that are needed in order to obtain some competitive ratio (larger than 1). For this, we introduce the problem of string guessing and a lower bound for it. Then, if we are able to reduce this problem to CAPG, we can transform the lower bound to one that is valid for CAPG. In other words, we have to show that, if we can solve CAPG with some competitive ratio and a certain amount of advice, then we could use this solution to achieve some performance for string guessing, contradicting the lower bound on string guessing, so such a solution for CAPG cannot exist. Therefore, we continue by defining the auxiliary problem.

▶ **Definition 16** (String Guessing with Unknown History [2]). The problem of string guessing with unknown history over an alphabet Σ, $|\Sigma| \geq 2$, is a minimization problem Π_{SGU}. Every instance $I \in \mathcal{I}(\Pi_{\text{SGU}})$ consists of some $n \in \mathbb{N}$, followed by $n-1$ requests "?" containing no additional information, and a string $s = s_1 s_2 \ldots s_n \in \Sigma^n$ of length n, i.e., $I = (n, ?, ?, \ldots, ?, s)$. A corresponding solution $S \in \mathcal{S}(I)$ is of the form $S = (a_1, a_2, \ldots, a_n)$, where $a_i \in \Sigma$, and the last request remains unanswered. The measurement function is the Hamming distance[4] between $s_1 s_2 \ldots s_n$ and $a_1 a_2 \ldots a_n$.

So in principle, a corresponding algorithm obtains the length of some yet unknown string s, and has to guess every character of s, without being aware of whether past guesses have been accurate. After all characters have been guessed, s is revealed to the algorithm. In the case that is of primary importance for us, namely $|\Sigma| = 2$, we call this problem *bit guessing*, since every response of a solution is a guess between two values, i.e., we try to find the right string of bits.

The next theorem states the lower bound for bit guessing we will use in our reduction.

▶ **Theorem 17** (Böckenhauer et al. [2]). *If $\frac{1}{2} \leq \gamma \leq 1$ and an online algorithm with advice guesses at least γn bits correctly of every instance of bit guessing with unknown history, then it uses at least $(1 + (1-\gamma)\log_2(1-\gamma) + \gamma \log_2 \gamma)n$ bits of advice.* □

Before we continue, let us elaborate a little on reduction proofs in the context of online algorithms with advice. Suppose we want to reduce an online problem Π_A to an online problem Π_B. Let therefore $\textsc{Alg}_A$ denote an online algorithm with advice for problem Π_A and likewise $\textsc{Alg}_B$ denote some online algorithm with advice for problem Π_B. Then, first the adversary constructs a fixed instance for Π_A and the oracle writes the most helpful advice string for this instance and this algorithm $\textsc{Alg}_A$ to the dedicated tape. Afterwards, $\textsc{Alg}_A$ gets its input in an online manner and has access to the advice tape. Possibly using this, $\textsc{Alg}_A$ now impersonates the role of the adversary for $\textsc{Alg}_B$. Hence, $\textsc{Alg}_A$ is free to construct any suitable input for $\textsc{Alg}_B$, but it also has to provide advice bits if demanded, by forwarding its own advice to $\textsc{Alg}_B$ on demand. Therefore, in case $\textsc{Alg}_A$ does not read additional advice bits, both algorithms use the same amount of advice. This is how advice is handled in the following proof. $\textsc{Alg}_A$ may determine its own output according to $\textsc{Alg}_B$'s output. However, there is a subtle detail: $\textsc{Alg}_A$ is not allowed to construct the input for $\textsc{Alg}_B$ depending on the output of $\textsc{Alg}_B$, since in the advice model the instance has to be fixed in advance, so that the advice can be based on it. Nonetheless, $\textsc{Alg}_A$ only needs to fix the asked requests one by one, possibly receiving new requests itself from the adversary in between. Then, since the oracle knows the instance for $\textsc{Alg}_A$ and the deterministic algorithm $\textsc{Alg}_A$, it can specify the advice beforehand.

[4] The Hamming distance of two strings of the same length is the number of positions at which their characters differ.

▶ **Theorem 18.** *Every online algorithm with advice for CAPG which achieves a competitive ratio of $c \leq \frac{12}{11}$ on a grid G has to read at least*

$$\left(1 + \left(6 - \frac{6}{c}\right)\log_2\left(6 - \frac{6}{c}\right) + \left(\frac{6}{c} - 5\right)\log_2\left(\frac{6}{c} - 5\right)\right)\frac{|E(G)|}{2}$$

bits of advice.

Proof. Let Capg be some online algorithm with advice for CAPG. Then, we can devise another algorithm BGuess for bit guessing with unknown history, which uses Capg and forwards the advice bits on demand, i.e., we reduce the bit guessing problem to the Call Admission Problem on Grids. In conclusion, Theorem 17 then yields a lower bound on the amount of advice which Capg has to use.

Initially, on an instance $I' \in \mathcal{I}'$ for bit guessing, BGuess obtains the number n' of bits it has to guess. As Subsection 2.4 explains, it suffices to consider instances that are defined on an $(m \times n)$-grid G with m and n odd, such that $\frac{|E(G)|}{2} = n'$. Note that any $(m \times n)$-grid has $m \cdot (n-1) + n \cdot (m-1)$ edges, so this is possible. Hence, we can use the same instances $I \in \mathcal{I}$ for CAPG as in Theorem 15, consisting of phases P_1 and P_2.

BGuess also operates in two phases: First, for every "?" received, it demands one of the requests from P_1 and guesses 1 for every request that Capg satisfies, and 0 otherwise, so the ith answer of Capg specifies the ith guess of BGuess. Since hereafter BGuess has guessed all bits, the correct bit string s is revealed to BGuess, and BGuess now knows which of its guesses have been accurate. As discussed before, BGuess is only allowed to exploit the knowledge of s, but not what Capg answered in order to ask further requests. However, note that so far P_2 has not been constructed, so in particular whether the decisions of Capg have been right or wrong is still not revealed and thereby, though BGuess has completed its output, wrong decisions of Capg of the past might be punished by following requests. Since the oracle is conscious of all these decisions, and the advice is only used by Capg, the advice bits then are actually chosen with regard to the final instance for Capg.

Therefore, in P_2, BGuess partitions every request of P_1 which corresponds to a 0 in s into two requests, includes them into P_2 and demands them from Capg. Then, the optimal answer for Capg would have been to grant exactly the requests for which s contains a 1. Note that this construction does not depend on the answers of Capg or BGuess, but only on the last request s for BGuess, which is transformed into possibly multiple requests for Capg. So, this way BGuess penalizes Capg for every guessed bit that deviates from the corresponding bit of s, because then, for every such bit, Capg's solution differs from the optimal solution Opt(I). However, since for multiple erroneously unsatisfied requests of P_1, Capg might be able to grant requests from P_2 which are not contained in the optimal solution, the cost of Capg is not obvious. Therefore, we analyze the consequences in more detail.

Case 1: Suppose Capg has mistakenly not granted k_1 many requests from P_1. Then, this decreases gain(Capg(I)) by k_1 compared to gain(Opt(I)) (with respect to P_1 only, of course, since the unused edges might be used elsewhere).

Case 2: If Capg has satisfied a request $r \in P_1$ that is not satisfied by Opt(I), then there are two possibilities to consider:

Case 2.1: First, Capg might not be able to accept any of the requests $r_1, r_2 \in P_2$ which partition r, so mutual exclusivity actually holds.[5] Consequently, for k_{21} incidents

[5] At least in some sense, since r_1 and r_2 do not necessarily contradict with r itself. In particular, it is

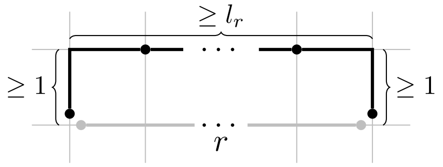

Figure 11 Every detour (black) has to go at least one edge up, at least the length l_r of the request r (gray) to the right and again all the way down, since the requests are horizontally or vertically aligned in this setting. The same holds for the rotated settings.

of such an r, Capg loses k_{21} of the potential gain, since it only satisfies r instead of both r_1 and r_2. Observe that these kinds of mistakes do not leave free additional edges compared to the optimal solution, because the only reason why r_1 and r_2 cannot be satisfied is that the edges are already occupied by some other request, so there is no "advantage" one could make use of when satisfying other requests.

Case 2.2: Eventually, the case remains in which Capg can satisfy at least one of r_1 and r_2 nonetheless and thereby, if both are accepted, is able to compensate for former mistakes. Let there be k_{22} such incidents. Then, either r has been granted using a detour, thus r_1 and r_2 can be satisfied directly, but the detour concerning r has length at least 4 (see Figure 11). Otherwise, r_1 and r_2 need to use detours, each of length at least 3 (again see Figure 11). However, in either scenario it conflicts with more requests than in the optimal solution, which is what we will exploit: Since every conflicting request has length at most 2 and the optimal solution completely covers the grid with requests, the mistakes of case 1 can only leave two edges per request unused in comparison to the optimal solution. In order to compensate a former mistake, both r_1 and r_2 need to be granted. Therefore, by counting the needed amount of edges to satisfy two extra requests of case 2.2 and the number of edges which are unoccupied due to mistakes of case 1, we obtain $\min\{4, 2\cdot 3\}\cdot k_{22} \leq 2k_1$, so no more than $\frac{1}{2}k_1$ requests can be rectified.

Thus, altogether, if BGuess guesses a fraction of γ bits correctly, the $(1-\gamma)n' = k_1 + k_{21} + k_{22}$ mistakes of BGuess result in a decline of at least $k_1 + k_{21} - k_{22}$ for the gain of Capg in contrast to the optimal gain, i.e.,

$$\begin{aligned}
\text{gain}\,(\text{Capg}(I)) &\leq \text{gain}\,(\text{Opt}(I)) - (k_1 + k_{21} - k_{22}) \\
&= \text{gain}\,(\text{Opt}(I)) - \frac{k_1 + k_{21} - k_{22}}{(1-\gamma)n'}(1-\gamma)n' \\
&= \text{gain}\,(\text{Opt}(I)) - \frac{k_1 + k_{21} - k_{22}}{k_1 + k_{21} + k_{22}}(1-\gamma)n'.
\end{aligned}$$

Note that mistakes as in case 2.1 cannot be corrected and decrease the gain of Capg only, therefore without loss of generality we assume $k_{21} = 0$, so $(1-\gamma)n' = k_1 + k_{22}$. Together

possible that even without permitting r both r_1 and r_2 could not be granted.

with $k_{22} \le \frac{1}{2}k_1$ we derive

$$\begin{aligned}
\text{gain}\left(\textsc{Capg}(I)\right) &\le \text{gain}\left(\text{Opt}(I)\right) - \frac{k_1 + k_{21} - k_{22}}{k_1 + k_{21} + k_{22}}(1-\gamma)n' \\
&\le \text{gain}\left(\text{Opt}(I)\right) - \frac{k_1 - k_{22}}{k_1 + k_{22}}(1-\gamma)n' \\
&\le \text{gain}\left(\text{Opt}(I)\right) - \frac{\frac{1}{2}k_1}{\frac{3}{2}k_1}(1-\gamma)n' \\
&= \text{gain}\left(\text{Opt}(I)\right) - \frac{1}{3}(1-\gamma)n'.
\end{aligned}$$

Then, for the competitive ratio c of Capg, clearly

$$\begin{aligned}
c &\ge \frac{\text{gain}(\text{Opt}(I))}{\text{gain}(\textsc{Capg}(I))} \\
&\ge \frac{\text{gain}(\text{Opt}(I))}{\text{gain}(\text{Opt}(I)) - \frac{1}{3}(1-\gamma)n'} \\
&\ge 1 + \frac{\frac{1}{3}(1-\gamma)n'}{\text{gain}(\text{Opt}(I)) - \frac{1}{3}(1-\gamma)n'}.
\end{aligned}$$

Now, gain$(\text{Opt}(I)) \le 2n'$, because there are exactly that many edges in G. Hence,

$$\begin{aligned}
c &\ge 1 + \frac{\frac{1}{3}(1-\gamma)n'}{2n' - \frac{1}{3}(1-\gamma)n'} \\
&= 1 + \frac{\frac{1}{3}(1-\gamma)}{2 - \frac{1}{3}(1-\gamma)} \\
&= 1 + \frac{1-\gamma}{5+\gamma} \\
&= \frac{6}{5+\gamma}.
\end{aligned}$$

Solving for γ results in

$$\gamma \ge \frac{6}{c} - 5.$$

Hence, the condition $\gamma \ge \frac{1}{2}$ yields $c \le \frac{12}{11}$ and thus, by Theorem 17 we obtain that at least

$$\begin{aligned}
&\left(1 + \left(6 - \frac{6}{c}\right)\log_2\left(6 - \frac{6}{c}\right) + \left(\frac{6}{c} - 5\right)\log_2\left(\frac{6}{c} - 5\right)\right)n' \\
=&\left(1 + \left(6 - \frac{6}{c}\right)\log_2\left(6 - \frac{6}{c}\right) + \left(\frac{6}{c} - 5\right)\log_2\left(\frac{6}{c} - 5\right)\right)\frac{|E(G)|}{2}
\end{aligned}$$

bits of advice have to be used by Capg in order to be c-competitive. ◀

Actually, Barhum et al. [1] showed that, on a path G, the just exploited bound of Theorem 15 can be enhanced by a factor of almost 2 to $|E(G)| - 1$ advice bits, which they also proved to be sufficient. Intuitively, in order to achieve this, we have to force any algorithm to take more decisions concerning whether to take a request or not, so that it has to use some advice for every such decision. Therefore, we need to construct a set of instances $\mathcal{I}$ where the requests are possibly densely packed and the satisfaction of a request r ought to be contradicting with many other requests possibly present in the instance, i.e., granting r should exclude the admission of a lot of other requests which may be asked in the future. Since mutually exclusiveness is obtained quite easily on paths, but is rather involved for proper grids, following the employment of this idea for CAPG instead of DPA needs some extra workarounds. However, for either problem a valid construction of such instances requires caution.

To this end, we will introduce some additional findings, which are very helpful for proving lower bounds. In principle, they just formalize the intuitive understanding that, if an online algorithm has to take different decisions for some instances, before it is observable for the algorithm that the instances themselves are mutually distinct, then the algorithm needs advice to distinguish them. Hence, if there are d such instances, then $\log_2(d)$ advice bits are needed. For a more thorough discussion on partition trees, and detailed proofs of the following lemma and theorem, we recommend [15].

▶ **Definition 19** (Partition Tree [1])**.** Let $\mathcal{I}$ be some set of instances for an online problem Π. Then, a partition tree $\mathcal{T}(\mathcal{I})$ is a graph which is defined as follows:

1. Each vertex v of $\mathcal{T}(\mathcal{I})$ is associated with a set of instances $\mathcal{I}_v \subseteq \mathcal{I}$ and the length k_v of a prefix which all $I \subseteq \mathcal{I}_v$ have in common.
2. The sets of the children of an inner vertex v of $\mathcal{T}(\mathcal{I})$ constitute a partition of $\mathcal{I}_v$.
3. The root is associated with the set of all considered instances $\mathcal{I}$.

Slightly abusing notation, we define the prefix of length k of an instance, a solution or a string to be $[\cdot]_k$. Then, $[I]_k$, for an instance I, refers to the first k requests, for a solution S, $[S]_k$ denotes the first k replies, and for a string s we define $[s]_k$ to be the first k characters of s.

As mentioned before, the next lemma will state that, for distinct instances which are indistinguishable for the algorithm at the time it has to take different decisions for achieving an optimal solution, distinct advice strings are necessary. The theorem after that completes our needed prerequisites by concluding how much advice then has to be read.

▶ **Lemma 20** (Barhum et al. [1])**.** *Let $\mathcal{I}$ be some set of instances for an online problem Π and let $\mathcal{T}(\mathcal{I})$ be some corresponding partition tree. Further, consider a vertex v of $\mathcal{T}(\mathcal{I})$ and two vertices v_1 and v_2 in disjoint subtrees rooted at children of v. Let $I_1 \in \mathcal{I}_{v_1}$ and $I_2 \in \mathcal{I}_{v_2}$ be any instances with a common prefix of length k_v. If for all optimal solutions S_1, S_2 for I_1 and I_2, respectively, $[S_1]_{k_v} \neq [S_2]_{k_v}$, then different advice strings have to be used for I_1 and I_2 by every optimal online algorithm with advice.* ◻

▶ **Theorem 21** (Barhum et al. [1])**.** *Let $\mathcal{I}$ be some set of instances for an online problem Π and let $\mathcal{T}(\mathcal{I})$ be some corresponding partition tree such that the conditions of Theorem 20 are satisfied and suppose that $\mathcal{T}(\mathcal{I})$ has d leaves. Then, every optimal online algorithm with advice has to use at least $\log_2(d)$ advice bits.* ◻

Let us now explain the idea of the proof by Barhum et al. [1] for the disjoint path allocation problem on a path of length l, i.e., a $(1 \times (l+1))$-grid. We consider a set of instances $\mathcal{I}$ for which the optimal solutions consist of requests that partition the whole path, in particular there is no edge that is not used, i.e., at every vertex at which some request ends, another request starts (except for the end vertices of course). Thus, each inner vertex v_i, $2 \leq i \leq l$, can be associated with a bit s_{i-1}, which is 1 if in the provably unique optimal solution one request ends and another one starts at v_i, and 0 otherwise. So an instance $I \in \mathcal{I}$ is constructed by taking any bit string s of length $l-1$ and splitting I into l phases in the following manner: Phase P_i contains all i possible requests of length $l+1-i$ on the path,

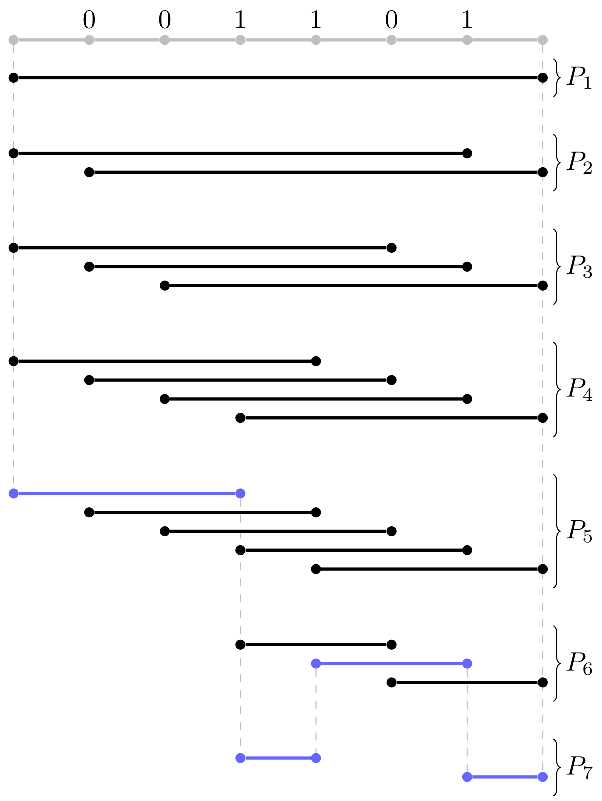

Figure 12 An example instance as used by Barhum et al. [1] on a path of length 7 which depicts the phases and how the path is partitioned by the optimal solution that is indicated by the above bit string.

except those which overlap a request from an earlier phase which has been granted by the (unique) optimal solution, which in turn is indicated by s as described earlier. Within a phase, the requests are given from left to right (see Figure 12).

It can be shown that the optimal solutions for different instances already have to differ before the instances are distinct on their prefixes of requests. Thus, Theorem 20 and Theorem 21 are satisfied. Also, we can describe the optimal solutions by the bit string $s_1 s_2 \dots s_{l-1}$, so in conclusion, there is a partition tree $\mathcal{T}(\mathcal{I})$ which has 2^{l-1} leaves, so at least $\log_2(2^{l-1}) = l - 1$ advice bits are required.

Unfortunately, returning to the case of a general grid, playing this game on every row and column separately does not immediately yield a similar result. This is because, e.g., in case that for some instance $I \in \mathcal{I}$, the solution $S(I)$ intended by the bit strings $s_1, s_2, \dots, s_{m+n}$ (one for each row and each column) grants quite long requests on every second row (or column), some other solution $S'(I)$ could reject the long requests with a cost of only 1 per request and rather use their many edges for detours which satisfy relatively short requests, resulting in an increased overall gain. Figure 13 shows a concrete counterexample to this strategy on a (7×7)-grid (depicting only the requests of interest due to reasons of space and clarity).

Hence, the size of the requests has to be restricted what can be achieved by starting only at phase P_{l-3} on each row (with respect to the above enumeration of phases), or respectively each column, so every request has length at most 4, i.e., the associated bit strings contain at

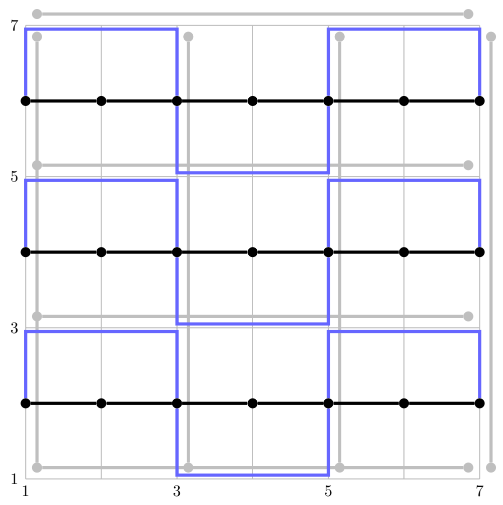

Figure 13 In this figure only the contradicting requests are shown. Those of length 6 drawn in gray are not satisfied, but instead the requests of length 2 in blue on every even row are granted using detours. If all non-contradicting requests are granted or dismissed according to the bit strings $s_1, s_2, \dots, s_{m+n}$, then this solution grants $9 - 8 = 1$ request more than $S(I)$.

most three consecutive 0s. For these strings, we will prove the following lemma and using this, eventually generalize the proof to the case of grids.

▶ **Lemma 22.** *There are t_{n+2} bit strings of length $n \in \mathbb{N}$ which contain at most three consecutive* 0*s, where t_n denotes the nth tetranacci number.*[6]

Proof. We count the number of such bit strings recursively. Note that in case we append a 0 to a given bit string which satisfies the property, we cannot conclude whether the property is violated or not without additional information. Hence, let us count slightly different bit strings.

Let t_k denote the number of bit strings of length $k \geq 1$ which satisfy the property, and both start and end with a 1. Then, $t_1 = 1$, $t_2 = 1$, $t_3 = 2$, $t_4 = 4$. Moreover, observe that a bit string b of length $k \geq 5$ has to end with one of the suffixes 1 1, 1 01, 1 001 or 1 0001 in order to comply with the property. Hence, by considering in each case the first 1 of the suffixes to be the last one of a shorter bit string b' which satisfies the property, we count $t_k = t_{k-1} + t_{k-2} + t_{k-3} + t_{k-4}$. But this is just the definition of the tetranacci numbers for $k \geq 1$.

Finally, by removing the first and last 1 of each such string, we retrieve all bit strings which satisfy the originally imposed condition. Thus, the number of bit strings of length n with at most three consecutive 0s is t_{n+2}. ◀

▶ **Theorem 23.** *Every optimal online algorithm with advice for CAPG on an $(m \times n)$-grid G has to read at least $m \cdot \log_2(t_n) + n \cdot \log_2(t_m)$ advice bits.*

Proof. Basically, we want to ask the last phases of the same instances as in the original proof by Barhum et al. [1] separately on each row and each column. Hence, let $\mathcal{I}$ be the set of instances we use in this proof. Then, for the construction of an instance $I \in \mathcal{I}$, consider the

[6] A000078 in Sloan's "The On-Line Encyclopedia of Integer Sequences." [13]

ordering Q of the rows and columns in which they are sorted from left to right, respectively from top to bottom. Then, successively for each row (column), $q_i \in Q$, we take some bit string s_i of length $n-2$ (or $m-2$, respectively) with at most three consecutive 0s and create phases just as before according to s_i but this time only those containing requests of length at most 4, i.e., phases $P_{n-4}, P_{n-3}, P_{n-2}, P_{n-1}$ ($P_{m-4}, P_{m-3}, P_{m-2}, P_{m-1}$, respectively), where each row (column) is considered as an isolated path. Recall that bit strings with at most three consecutive 0s, like s_i, correspond to a partition of a row (column) with requests of length at most 4, so the bit strings and the phases are consistent.

Eventually, we want to satisfy the conditions of Theorem 20 in order to be able to apply Theorem 21. Therefore, we first want to establish that there are only a few optimal solutions for every instance $I \in \mathcal{I}$, so then, when arguing about all optimal solutions for I, actually only these have to be considered.

Let $S(I)$ be a solution that satisfies requests on every row (column) q_i according to the dedicated bit string s_i, videlicet, partitions q_i such that exactly the edges from q_i are used, and at any inner vertex of q_i one request ends and another one starts if and only if the corresponding bit in s_i is 1. Obviously, there exists such a solution. Observe that $S(I)$ uses all edges and that all requests are satisfied using a shortest path, so there is no possibility to get by with fewer edges or granting additional requests. Furthermore, all these shortest paths are unique, since the requests are horizontally or vertically aligned respectively. Thus, there is only one unique solution $S(I)$ which satisfies exactly the requests suggested by $s_1, s_2, \ldots, s_{m+n}$. We intend to prove that $S(I)$ is one of the optimal solutions. For this purpose, we first conclude that, for all optimal solutions, any kind of deviating from $S(I)$ would provoke a detour and later that these detours can only be of certain forms, such that every of these solutions can reach at most the gain of $S(I)$.

To this end, consider a solution $S'(I)$ that accepts different requests. If a request r_1, which has been granted by $S(I)$, is not permitted in $S'(I)$, then, by construction of I, there are no further requests on these edges. Thus, every solution that does not use these edges is inferior to any optimal solution, since it could have additionally admitted r_1. Every solution that still uses these edges has to use a detour on these edges such that it contradicts r_1.

In case there is a request r_2 which has not been satisfied by $S(I)$, but is accepted by $S'(I)$, I by definition contains shorter requests that partition r_2. Thus, either $S'(I)$ admits r_2 horizontally or vertically aligned, but clearly this would be suboptimal, because then more shorter requests could be satisfied instead using the same edges, or $S'(I)$ uses a detour for r_2 (or the contradicting shorter requests) as well.

Therefore, every optimal solution deviating from $S(I)$ involves detours. Remember that despite being referred to as "detour", in a more general setting, this does not necessarily diminish the gain of the solution as Figure 13 illustrates. However, with these restricted phases, such a solution cannot be superior, as we will see by examining the possible lengths of detours.

First, observe that for these instances every detour has length at least 3 (see Figure 11), because all requests are horizontally or vertically aligned. However, since there are no overlapping (meaning identical) requests of length 1, either such a request r_3 could have been satisfied directly needing less edges (if the edge would remain free otherwise), or the respective edge is occupied by another request r_4 of length at least 2. But then, r_3 can be satisfied directly using only one edge, while the other one uses the former detour of r_3, which would result in an (in some sense) equivalent solution $S''(I)$ that satisfies the same requests, but uses a detour of length at least 4 (see Figure 14). So if there is an optimal solution with a detour of length 3, then there is one satisfying the same requests and using the exact same

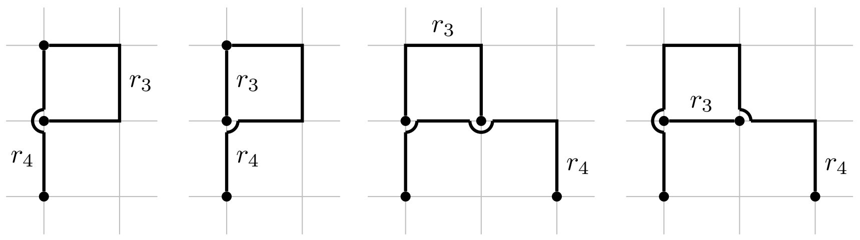

Figure 14 Two examples of a request r_3 and a request r_4, s.t. in each example they are satisfied in two different ways, but both times using the same edges.

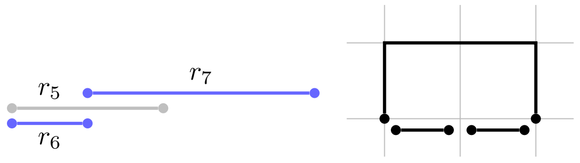

Figure 15 To the left, we see why a request of length 2 (gray) cannot be overlapped by only one request of length 1. On the right side, the unique form of a detour (up to symmetry) used by an optimal solution is depicted.

edges where all detours have length at least 4.

Next, consider the case that all detours in $S'(I)$ have length at least 4 and there exists a detour of length at least 5. Such a this solution cannot be optimal, since every detour is at least as long as the longest request and one is even longer, i.e., deleting the detours would yield some free edges, all of which are covered by $S(I)$ with requests that are satisfied with paths of length at most 4.

Thus, the case of detours of length exactly 4, which replace requests of length 4, remains. Note that such a solution can be at most as good as $S(I)$, since we trade detours of length 4 against requests of length at most 4, and additionally all detours of length 4 satisfy requests of length 2. In conclusion, then, $S(I)$ is optimal, and it remains to prove that optimal solutions of different instances have to be distinct already before their instances differ.

To this end, we need to know the shape of the detours in even greater detail, because we aim at showing that optimal solutions for distinct instances differ at the latest in a detour they actually grant. Therefore, we will disclose that all detours are actually shaped as Figure 15 depicts.

Since we can alter all detours to have length 4 and detours of length 4 satisfy requests of length 2, there has to be a request of length 2 involved in every detour (either as detour itself or as contradicting request). Knowing all this about detours, consider a request r_5 of length 2, which is not contradicted by two, but only by one request r_6 of length 1. Then, by construction, there has to be another request r_7 before r_5, which overlaps it and is intended by the bit string to be granted together with r_6. But then, again by the definition of the phases, r_5 cannot be part of this instance (see Figure 15). Hence, actually all detours are contradictions between one request of length 2 and two requests of length 1, and given a length of 4, there is actually only one possible kind of detour left that can be used by an optimal solution (up to symmetry, see Figure 15).

We proceed by proving that every optimal solution has to grant a detour before it is able to reject a request which is intended to be satisfied by the corresponding bit string. To comprehend why we do so, think of a solution that first omits a request that is intended by the dedicated bit string, and afterwards permits a detour instead. Then, at the time the

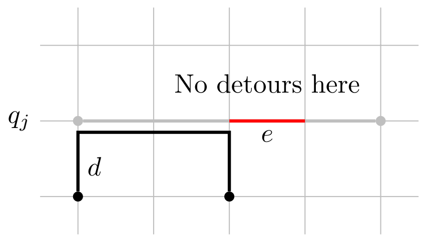

Figure 16 There is no way to get to and back from the single edge e, so it cannot be used to satisfy any request.

request in question has been left unsatisfied, it is still undetermined whether there will be further smaller requests on this part of the grid in the future or not, i.e., whether an optimal solution would need to take a detour later on or not. Yet, already the following request (or rather the existence of a subsequent phase on these edges) might depend on this. Hence, there could be two distinct instances, for which some optimal solutions do not differ earlier in time, as imposed by Theorem 20. However, given detours of a form as in Figure 15, it can be proven that any optimal solution has to satisfy a detour before a request which is suggested by the bit string has not been granted.

For the sake of contradiction, suppose this is not the case. Then, let q_j be the first row (column) where an intended request has not been granted. By the previous reasoning, this request has length 4. Let d be a detour that uses 2 of these free edges. Then, there is some free edge e which is adjacent to two edges of d. Since by assumption there has not been a detour before, there is no detour above (left) of q_j. But then, since by construction there are no shorter requests partitioning the intended request on this part of q_j, there is no possibility to satisfy a request passing through e, and hence the solution is not optimal (see Figure 16). Therefore, some detour has to be accepted first.

Let $I^{(1)}, I^{(2)} \in \mathcal{I}$ be two distinct instances, and $S(I^{(1)})$ and $S(I^{(2)})$ be the optimal solutions suggested by the respective bit strings $s^{(1)} = s_1^{(1)} s_2^{(1)} \dots s_{m+n}^{(1)}$ and $s^{(2)} = s_1^{(2)} s_2^{(2)} \dots s_{m+n}^{(2)}$. Further, $S'(I^{(1)})$ and $S'(I^{(2)})$ denote some optimal solutions for $I^{(1)}$ and $I^{(2)}$ which take detours. Then for P_k being the first phase in which $I^{(1)}$ and $I^{(2)}$ differ, by construction, there has to be a request in the preceding phase that has been suggested by $s^{(1)}$, but not by $s^{(2)}$, or the other way around. Since by assumption $S(I^{(1)})$ and $S(I^{(2)})$ act according to these bit strings, they differ even before their instances differ. Also, either $[S'(I^{(2)})]_j = [S(I^{(2)})]_j$ for j being the number of requests asked so far, or $S'(I^{(2)})$ has used a detour already. Anyway, $S'(I^{(2)})$ and $S(I^{(1)})$ have to be distinct before their instances as well, and so do $S'(I^{(1)})$ and $S(I^{(2)})$. Now, assume that $[S'(I^{(1)})]_j$ and $[S'(I^{(2)})]_j$ are identical. Since $[S(I^{(1)})]_j \neq [S(I^{(2)})]_j$, they have to take a detour. But since there is only one kind of detour it is easy to see that using the same detours would imply that $[s^{(1)}]_j = [s^{(2)}]_j$, and thus, also $S'(I^{(1)})$ and $S'(I^{(2)})$ have to differ earlier than their instances.

Finally, we conclude that there is a partition tree $\mathcal{T}(\mathcal{I})$ which satisfies the prerequisite of Theorem 20 with as many leaves as there are different instances in $\mathcal{I}$, which themselves can be counted by the corresponding bit strings for each row and each column. Hence, there are $t_n^m \cdot t_m^n$ instances and by Theorem 21 there are $\log_2(t_n^m \cdot t_m^n) = m \cdot \log_2(t_n) + n \cdot \log_2(t_m)$ advice bits necessary for every optimal online algorithm with advice. ◀

▶ **Corollary 24.** *Every optimal online algorithm with advice for CAPG on an $(m \times n)$-grid G has to read at least $0.94677 \cdot |E(G)| - m - n$ advice bits.*

Proof. It has been proven that the ratio $\frac{t_i}{t_{i-1}}$ of consecutive tetranacci numbers converges to the so-called tetranacci constant which is given by the unique real root greater than 1 of the characteristic polynomial of the recurrence, i.e., $\lambda > 1$ such that $\chi(\lambda) = \lambda^4 - \lambda^3 - \lambda^2 - \lambda - 1 = 0$, so $\lim_{i\to\infty} \frac{t_i}{t_{i-1}} = \lambda \geq 1.92756$ [11, 7]. According to [7], we thus have

$$\begin{aligned} t_k &\geq \frac{\lambda - 1}{2 + 5(\lambda - 2)}\lambda^{k-1} - 0.5 \\ &\geq 0.29381 \cdot 1.92756^k - 0.5. \end{aligned}$$

Since the logarithm is a strictly increasing function, we conclude

$$\begin{aligned} \log_2(t_k) &\geq \log_2(0.29381 \cdot 1.92756^k - 0.5) \\ &= \log_2\left(0.29381 \cdot 1.92756^k \left(1 - \frac{0.5}{0.29381 \cdot 1.92756^k}\right)\right) \\ &= \log_2(0.29381 \cdot 1.92756^k) + \log_2\left(1 - \frac{0.5}{0.29381 \cdot 1.92756^k}\right). \end{aligned}$$

Then, for sufficiently large k and small enough $\varepsilon \in \mathbb{R}$, $\varepsilon > 0$,

$$\begin{aligned} \log_2(t_k) &\geq \log_2(0.29381 \cdot 1.92756^k) - \varepsilon \\ &= \log_2(0.29381) + k \cdot \log_2(1.92756) - \varepsilon \\ &\geq 0.94677 \cdot k - 1.76705. \end{aligned}$$

Using this, the bound from Theorem 23 simplifies to

$$\begin{aligned} m \cdot \log_2(t_n) + n \cdot \log_2(t_m) &\geq m \cdot (0.94677n - 1.76705) + n \cdot (0.94677m - 1.76705) \\ &= 0.94677 \cdot 2mn - 1.76705m - 1.76705n \\ &= 0.94677 \cdot (2mn - m - n) - 0.82028m - 0.82028n \\ &= 0.94677 \cdot |E(G)| - 0.82028m - 0.82028n \\ &> 0.94677 \cdot |E(G)| - m - n, \end{aligned}$$

which concludes the statement. ◀

Note that if we could implement all phases, i.e., use bit strings of length $n - 2$ or respectively $m - 2$ for every row or column just as on the path, then this would result in $m \cdot (n - 2) + n \cdot (m - 2) = |E(G)| - m - n$ advice bits, so our result is indeed surprisingly close (see Figure 17).

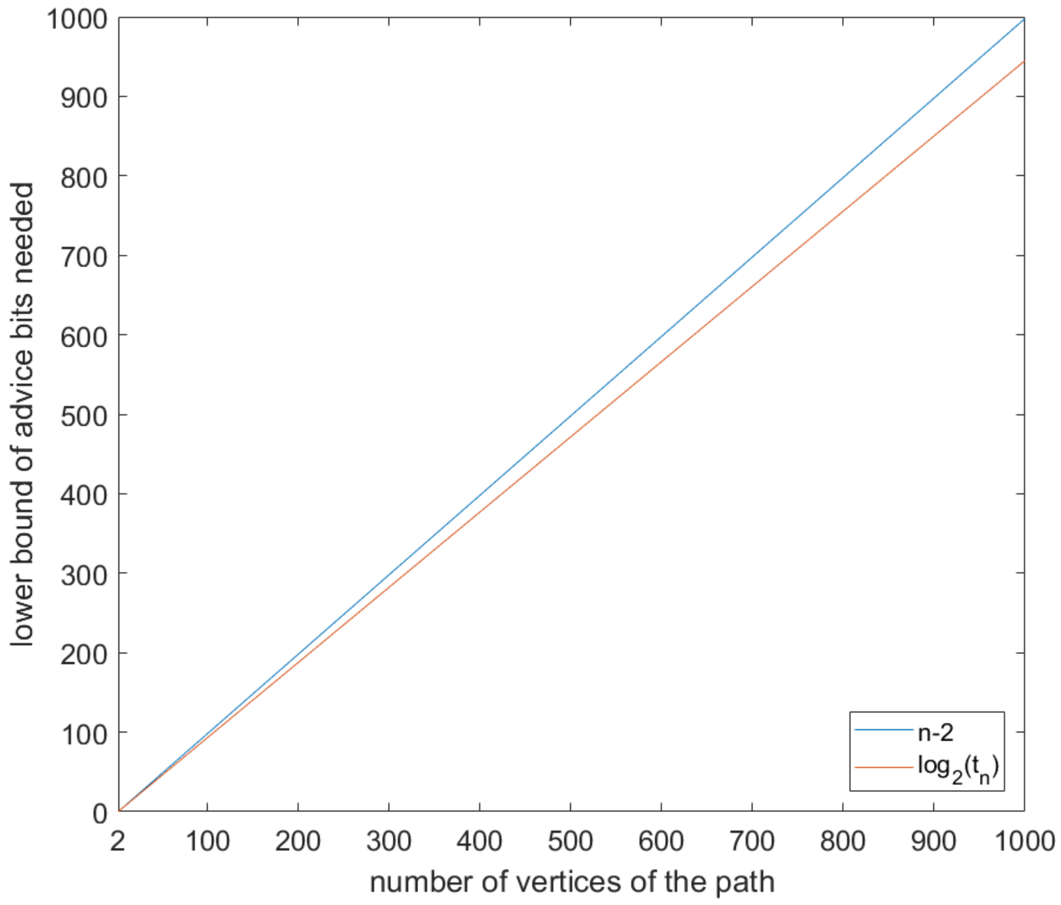

Figure 17 Illustration of the number of advice bits needed on a path of length n in comparison to the slightly worse lower bound which is obtained by using only requests of length 4 as in the proof of Theorem 23.

3.2 Upper Bounds

For the results regarding upper bounds, recall the definitions of $d(v)$ as the degree of a vertex v, $\Delta(G)$ as the maximum degree of a graph G, $\chi(G)$ as its chromatic number, and $\omega(G)$ denoting the maximum clique size.

Note that, unlike for lower bounds, upper bounds for DPA do not inherently carry over to CAPG, videlicet, a method to solve DPA does not necessarily have to be applicable for CAPG as well. Hence, the proofs of these results will be far less guided by ideas of upper bounds for DPA. In fact, upper bounds are usually given by an explicit design of an algorithm that achieves a solution of some quality reading at most a certain amount of advice. Since this has to hold for all instances, general properties concerning the grid and the problem procedure are exploited instead of inspecting specific instances as typical for lower bounds. Let us start with the most obvious upper bound.

▶ **Theorem 25.** *There is an optimal online algorithm with advice for CAPG which reads at most $2|E| \cdot \lceil \log_2(|V|) \rceil \leq 2|E| \cdot \log_2(|E| + m + n)$ bits of advice for every $(m \times n)$-grid $G = (V, E)$.*

Proof. Recall that there are $|V| = mn$ vertices and $|E| = m(n-1) + n(m-1) = 2mn - m - n$ edges in G, so $|V| = \frac{|E|+m+n}{2}$. Hence, for an arbitrary, fixed optimal solution, the oracle can encode the two endpoints of every satisfied request using $2 \cdot \lceil \log_2(|V|) \rceil \leq 2 \cdot \log_2(|E| + m + n)$ bits per request. Since there are at most $|E|$ requests contained in an optimal solution, the oracle needs at most $2|E| \cdot \lceil \log_2(|V|) \rceil$ bits of advice to encode all granted requests of the optimal solution (the rest of the $2|E| \cdot \lceil \log_2(|V|) \rceil$ bits can be filled with repeating the first encoded vertex). Then, some online algorithm Alg with advice can read these $2|E| \cdot \lceil \log_2(|V|) \rceil$ bits at the very beginning and with this additional information it is able to recompute a (possibly different) optimal solution in an offline manner. Alg then simply admits or denies the demanded requests according to this solution. ◀

Actually, a further upper bound for a restricted set of instances is already established by the idea of Theorem 23, since its proof is (nearly) constructive, just as for the inspiring result regarding DPA [1].

▶ **Theorem 26.** *For instances with either horizontally or vertically aligned requests of length at most 4, there is an optimal online algorithm with advice for CAPG that uses at most $\lceil m \cdot \log_2(t_n) + n \cdot \log_2(t_m) \rceil$ advice bits for every $(m \times n)$-grid G, where t_k denotes the kth tetranacci number.*

Proof. Since in Theorem 23 it has been proven that the solution indicated by the bit string $s_1 s_2 \ldots s_{m+n}$ is optimal, it is sufficient to transmit this information. Furthermore, according to Theorem 20, there are exactly $t_n^m \cdot t_m^n$ such strings. Hence, given an ordering (e.g., lexicographical) of all possible such strings, the oracle can specify the bit string corresponding to an optimal solution by writing the position of the bit string in the ordering on the advice tape. For this, the oracle needs $\lceil \log_2(t_n^m \cdot t_m^n) \rceil = \lceil m \cdot \log_2(t_n) + n \cdot \log_2(t_m) \rceil$ advice bits. Then, there is some online algorithm Alg with advice that reads these bits, determines the bit string $s_1 s_2 \ldots s_{m+n}$ by considering the same ordering and then decides in compliance with it, i.e., Alg computes an optimal solution. ◀

Therefore, analogously to DPA, in this special case the bound is tight. Again, we can numerically bound these parameters and obtain the following corollary.

▶ **Corollary 27.** *For instances with either horizontally or vertically aligned requests of length at most* 4*, there is an optimal online algorithm with advice for CAPG that uses at most* $|E(G)| - \frac{m+n}{2} + 1$ *advice bits for every* $(m \times n)$*-grid* G*.*

Proof. Analogously to the proof of Theorem 24, by [11, 7] we can bound the tetranacci constant λ from above by $\lambda \leq 1.92757$, and thus, for the kth tetranacci number we conclude from the bound given in [7] that

$$\begin{aligned} rClt_k &\leq \frac{\lambda - 1}{2 + 5(\lambda - 2)} \lambda^{k-1} + 0.5 \\ &\leq 0.29381 \cdot 1.92757^k + 0.5. \end{aligned}$$

With the same reasoning as in the proof of Theorem 24, this results in

$$rCl \log_2(t_k) \leq 0.94679 \cdot k - 1.76704.$$

Therefore,

$$\begin{aligned} \lceil m \cdot \log_2(t_n) + n \cdot \log_2(t_m) \rceil &\leq 0.94679 \cdot |E(G)| - 0.82025m - 0.82025n + 1 \\ &< |E(G)| - \frac{m+n}{2} + 1, \end{aligned}$$

which by Theorem 26 yields the desired proposition. ◀

As observed easily, knowing which edges of the grid are used to satisfy a request and which remain unused for some optimal solution is not sufficient for an algorithm to reconstruct an optimal solution. For example, consider the instances used in the proof of Theorem 15, in which all edges are used by every optimal solution, and in which some of the requests of length 2 are partitioned into two requests each of length 1 in a phase later in time. Then, obviously, knowing that all edges are used is worthless, since the algorithm still cannot conclude whether to grant a request of length 2 or rather to wait for a potential pair of shorter requests. Therefore, we cannot bound the number of advice bits from above by simply writing one bit per edge indicating whether it is utilized in some specific optimal solution or not. This suggests it might be necessary to transmit the membership to a request as well, in order to establish the contours of the requests somehow. Hence, the question of how many bits per edge are necessary to reconstruct these memberships in an unambiguous way arises. Considering that for this reason "neighboring" paths need to be distinguishable, i.e., they ought to have different identifiers, one might already presume correctly that this leads to the prominent issue of vertex coloring in some transformed auxiliary graph, which will be defined next. Note that the unused edges need to be distinguishable from the used ones, too. However, they do not necessarily form paths, but arbitrary connected components and we are free to split them further up in any way, minimizing the number of needed colors, as long as we can still discriminate them from the used ones.

▶ **Definition 28.** Given any optimal solution S for an instance of CAPG on a grid G, the graph $\widehat{G}(S) = (\widehat{V}, \widehat{E})$ is defined as follows. Every path p used by S to satisfy a request corresponds to a vertex $v_p \in \widehat{V}$. Every connected component of unused edges in G according to S is split up into connected components s.t. each component q corresponds to a vertex $v_q \in \widehat{V}$ and s.t. the chromatic number $\chi(\widehat{G}(S))$ is minimized. There are no further vertices in $\widehat{V}$. The neighborhood of a vertex $v_g \in \widehat{V}$ is defined as the vertices whose corresponding connected components share a vertex with g, i.e., the set of edges $\widehat{E}$ contains all $\{v_g, v_h\}$ such that $v_g, v_h \in \widehat{V}$ and g and h have some vertex in common.

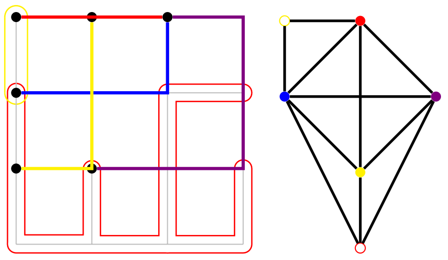

Figure 18 On the left, some optimal solution $S(I)$ for an instance I on a grid G is depicted. The granted requests are drawn as colored paths, whereas the unused edges are surrounded by colored borders indicating how the connected component of unused edges is divided further in order to minimize $\chi(\widehat{G})$. To the right, $\widehat{G}$ is shown, where the filled vertices correspond to the paths colored in the same color which satisfy requests in G, and the unfilled vertices correspond to the connected components of unused edges with the same color. Since $\widehat{G}$ contains a 4-clique even without considering the vertices corresponding to connected components of unused edges, and the depicted coloring is a proper 4-coloring, this is indeed optimal. Moreover, note that the unused edges need to be split up in order to achieve optimality.

For an example of the application of this definition, see Figure 18. For reasons of readability, we will hide the dependency on S and simply write $\widehat{G}$ instead of $\widehat{G}(S)$ whenever suitable. Furthermore, note that there might be multiple ways to split up a connected component according to the definition s.t. $\chi(\widehat{G}(S))$ is minimized, but for our purposes it suffices to examine any of these possibilities, so we venture to retain this inaccuracy.

▶ **Theorem 29.** *Let $\mathcal{I}$ denote all possible instances of CAPG on a grid $G = (V, E)$, and let $\mathcal{S}_{\text{opt}}(I)$ be the set of optimal solutions for an instance $I \in \mathcal{I}$. Then, there is an optimal online algorithm with advice for CAPG using at most*

$$\max_{I\in\mathcal{I}} \min_{S\in\mathcal{S}_{\text{opt}}(I)} \lceil |E| \cdot \log_2(\chi(\widehat{G}))\rceil + 2\lceil \log_2(\chi(\widehat{G}(S)))\rceil$$

advice bits.

Proof. Given an instance $I \in \mathcal{I}$ on a grid G, the oracle knows $\mathcal{S}_{\text{opt}}(I)$ and can compute $\chi(\widehat{G}(S))$ for every $S \in \mathcal{S}_{\text{opt}}(I)$. Hence, the oracle selects $S \in \mathcal{S}_{\text{opt}}(I)$ such that it minimizes $\lceil |E| \cdot \log_2(\chi(\widehat{G}))\rceil + 2\lceil \log_2(\chi(\widehat{G}(S)))\rceil$ for the given instance, colors the corresponding connected components of G according to $\widehat{G}$, and indicates the colors of all edges by enumerating all colorings of the edges (even non-proper colorings) from 0 to $\chi(\widehat{G})^{|E|} - 1$. Then, the oracle writes the appropriate number on the advice tape, where the edges of G are ordered in any arbitrary, but fixed way. For this, $\lceil |E| \cdot \log_2(\chi(\widehat{G}))\rceil$ bits are needed. Since $|E|$ is given as part of the instance, but an online algorithm cannot know $\chi(\widehat{G}(S))$, this value needs to be given in a self-delimiting encoding and prepended to the advice. This can be done using $2\lceil \log_2(\chi(\widehat{G}(S)))\rceil$ additional bits (see Subsection 2.4).[7]

Obviously, there exists an online algorithm Alg with advice for CAPG that first reads and decodes $\chi(\widehat{G}(S))$, calculates $\lceil |E| \cdot \log_2(\chi(\widehat{G}))\rceil$ as the length of the remaining advice, and then reads exactly these numbers from the advice tape and recovers the coloring of the connected components in G by using an identical ordering of the edges. Then, for every incoming request, Alg determines whether it can be satisfied using exactly the edges of one uniformly colored connected component. If this is the case, Alg satisfies the request,

[7] Actually there are better encodings, but since this is only a small additive term, we favor readability.

otherwise it gets rejected. Since, for any of the connected components that are used by the optimal solution S, a suitable request gets asked, and since ALG does not satisfy any request using edges from more than one of these connected components, ALG computes an optimal solution as well (which might be distinct since ALG could possibly interchange a connected component that is not used in S with one that is used in S in order to satisfy a request). Hence, ALG has to read at most $(|E|+2) \cdot \lceil \log_2(\chi(\widehat{G}))\rceil$ advice bits for the worst case of some $I \in \mathcal{I}$. ◀

▶ **Corollary 30.** *If $\max_{I\in\mathcal{I}} \min_{S\in\mathcal{S}_{\mathrm{opt}}(I)} \chi(\widehat{G}(S))$ can be bounded by a number c_χ which is known by an online algorithm with advice for CAPG, then the bound of Theorem 29 adjusts to $\lceil |E|\cdot\log_2(c_\chi)\rceil$ bits of advice.*

Proof. If the color of every edge is encoded by enumerating all possible colorings of the edges from 0 to $c_\chi^{|E|}-1$, and and c_χ is known to the online algorithm with advice, no self-delimiting encoding is necessary and $\lceil |E|\cdot\log_2(c_\chi)\rceil$ advice bits suffice to proceed as in Theorem 29. ◀

Note that this immediately yields a better constant than Theorem 25, since every request has length at least 1, so there are at most $|E|-1$ requests left which can be neighboring in the sense of Theorem 28, i.e., $\chi(\widehat{G}) \le \Delta(\widehat{G})+1 \le |E|$, so already $\lceil |E|\cdot\log_2(|E|)\rceil$ advice bits are sufficient. However, a short request cannot have many incident paths in a grid, thus this bound is only a rather coarse estimate and can be improved easily as the following corollary shows.

▶ **Corollary 31.** *There is an optimal online algorithm with advice for CAPG that reads at most $\lceil |E|\cdot\log_2(\frac{1}{3}(2|E|+7))\rceil$ bits of advice.*

Proof. Let $I \in \mathcal{I}$ be an arbitrary instance on a grid G and let S denote the solution minimizing $\lceil |E|\cdot\log_2(\chi(\widehat{G}))\rceil + 2\lceil \log_2(\chi(\widehat{G}(S)))\rceil$. Note that since, according to the definition of $\widehat{G}$, the connected components of unused edges in S are split up s.t. $\chi(\widehat{G})$ is minimized, we can without loss of generality assume that $\chi(\widehat{G})$ is at most as large as if the respective connected components were split up into paths. Hence, suppose all connected components are paths. Then, for l being the length of such a path p in G, by counting the remaining edges we observe that the number of paths sharing at least one vertex is trivially bounded from above by $|E|-l$. Moreover, at every inner vertex of p, at most 2 edges not contained in p are incident, so at most 2 other paths can share this vertex and for both outer vertices there is one additional such edge. Thus, a second upper bound on the number of neighbors of v_p in $\widehat{G}$ is given by $2(l+1)+2 = 2(l+2)$. Even in case that both bounds are attained, we have

$$|E|-l = 2(l+2) \iff l = \frac{|E|-4}{3},$$

so $d(v_p) \le 2(l+2) \le \frac{2}{3}(|E|+2)$, and since this holds for all vertices of $\widehat{G}$, we conclude that $\frac{2}{3}(|E|+2)+1 \ge \Delta(\widehat{G})+1 \ge \chi(\widehat{G})$. Thus, by Theorem 30 we obtain that there is some online algorithm which uses at most $\lceil |E|\cdot\log_2(\frac{1}{3}(2|E|+7))\rceil$ advice bits to compute an optimal solution. ◀

However, as Figure 19 illustrates, on an $(n\times n)$-grid with even n there is an instance s.t. $\widehat{G}$ contains an n-clique. Since $\chi(\widehat{G}) \ge \omega(\widehat{G})$, this implies that with this approach the upper bound cannot be enhanced to less than $\lceil |E|\cdot\log_2(n)\rceil = \lceil |E|\cdot\log_2(\sqrt{|V|})\rceil = \lceil \frac{1}{2}|E|\cdot\log_2(|E|+2n)-1\rceil \in \mathcal{O}(|E|\cdot\log(|E|))$ advice bits. Hence, either this upper bound

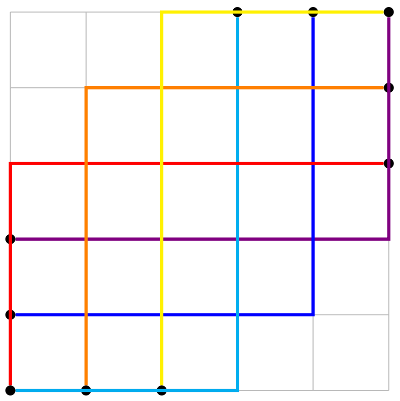

Figure 19 On the empty edges of the grid the instance contains requests of length 1 and other than that there are no further requests. Then, satisfying the shown requests and all requests of length 1 is the unique optimal solution, and every depicted path has a common vertex of the grid with every other shown path. Hence, $\widehat{G}$ has a clique of size at least 6, i.e., in G every shown path needs to be colored in a different color.

is already asymptotically tight, or we need an advanced method to prove a more compact bound.

Nevertheless, for requests of constant maximal length l, also $\Delta(\widehat{G})$ is bounded by some constant c_l, and thus at most $\lceil |E| \cdot \log_2(c_l+1)\rceil \leq c'_l \cdot |E| \in \mathcal{O}(|E|)$ advice bits are necessary, for a constant $c'_l = \lceil \log_2(c_l+1)\rceil$. Due to Theorem 24, this bound is therefore asymptotically tight, i.e., up tight to a constant factor.

▶ **Corollary 32.** *For instances that only consist of requests with constant maximal length,* $\Theta(|E|)$ *advice bits are both necessary and sufficient.* ◻

Furthermore, let us remark that in case of a path instead of a proper grid, we have $\chi(\widehat{G}) = 2$, and thus up to the additive constant, Theorem 30 constitutes another way of proving the tight upper bound for DPA of $|E| - 1$ advice bits.

However, there might be a way we can significantly improve the result of Theorem 31, since we only need be able to distinguish the end vertices of different satisfied requests, i.e., to know the borders between the end vertices, and be able to follow the path that is used to satisfy the request. Since at every inner vertex of such a path there are only three possible directions where the path may continue, it seems plausible that we can reduce the number of advice bits.

Let us try some sanity check of the extreme cases. First, consider some instance containing only requests of length 1. Then, it is sufficient to know where the requests start and end, but there are no edges where one has to decide in which direction a request should be satisfied. Hence, although a simple greedy algorithm would already be optimal in this case, even encoding the solution completely can be done by coloring used and unused edges of the optimal solution with two distinct colors. Therefore, for the complete encoding, $\log_2(2)|E| = |E|$ advice bits would be sufficient.

On the other side, we can look at instances consisting of arbitrarily long requests with pairwise different end vertices. Then, there is no danger to confuse the start or the end of a request, but we might need to know in which direction a path that satisfies a request continues. For this we could for every edge encode the options to continue in 90, 180 and 270 degrees by three different colors, and use one further color to characterize unused edges. Thus,

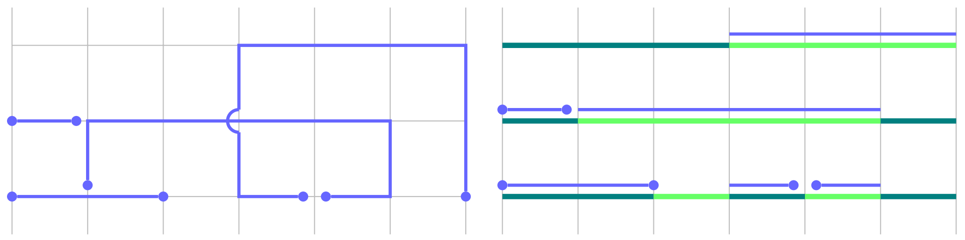

Figure 20 A part of a solution $S(I)$ is depicted on the left. To the right, there is an example of the edge-coloring on the rows, where the components restricted to each row are drawn in blue, and the resulting coloring of the rows is shown in shapes of green. For clarity the coloring of the columns is omitted.

an encoding of a solution for such an instance would need no more than $\log_2(4)|E| = 2|E|$ bits.

This suggests that combining these two variants might even result in a linear, and hence neglecting the multiplicative constant tight, upper bound for CAPG.

▶ **Theorem 33.** *There is an online algorithm with advice for CAPG that computes an optimal solution using at most $3|E|$ advice bits.*

Proof. Let $I \in \mathcal{I}$ be some instance and let $S(I)$ be some optimal solution for I.

First, consider every row and column q_i separately. Now, q_i may consist of three possible sorts of edges: First of all, edges which remain unused in $S(I)$. Then, there are edges that are connected to an end vertex of a satisfied request of $S(I)$ via edges of q_i alone. And finally, there are edges that are used in $S(I)$, but, restricting ourselves to q_i, not connected to an end vertex of a satisfied request of $S(I)$. Now, imagine every component that is connected on q_i is understood as a separate request when q_i gets edge-colored according to $\widehat{q_i}$ with $\chi(\widehat{q_i}) = 2$ colors, where for every row and column the same two colors, e.g., teal and lightgreen, are used (see Figure 20).

Note that for those requests that $S(I)$ actually satisfies in a horizontally or vertically aligned manner, this suffices to reconstruct the solution $S(I)$ concerning these requests, since, under the premise that it is known that no request is satisfied in an non-aligned manner (meaning not horizontally or vertically aligned), it is obvious that edges of different rows and columns belong to different satisfied requests (or are unused anyway), and satisfied requests within one row (or column) can be distinguished by the coloring just as in the preceding proofs (first and foremost in Theorem 29). Also, this coloring already achieves that we can distinguish the beginnings (or endings if you mind) of different requests that are satisfied in a non-aligned way, although we might not know which edges are used for such requests. So on every color transition either a request starts/ends or it is continued orthogonal from the current row or column (or $S(I)$ leaves the edge empty). In order to be able to attach sensible meanings to the colors, such as that shades of green establish borders between the end vertices, we will color some edges in multiple hues and merge them to a single color later on.

Hence, for all edges that are contained in a non-aligned path used to satisfy a request in $S(I)$, we use a further, distinct color, red. In this way we differentiate between requests that are satisfied horizontally or vertically, and such that are satisfied in a non-aligned form.

However, in case multiple red lines cross in some point and their additional green shades do not help to distinguish them, we might not know in which direction the request gets satisfied, i.e., which of the end vertices belong together (e.g., in Figure 21 the crossing

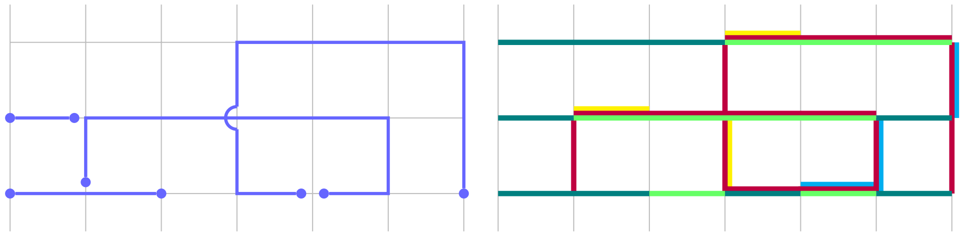

Figure 21 The same instance, solution and coloring as before, supplemented by the red, yellow and cyan colors indicating the paths of non-aligned satisfied requests.

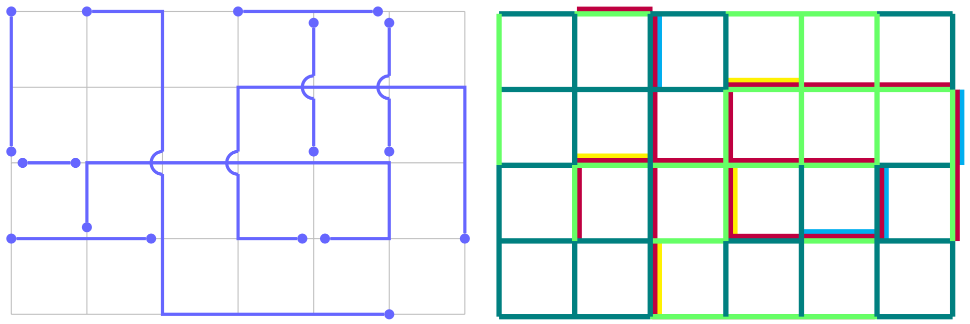

Figure 22 The complete solution $S(I)$, together with the corresponding coloring on the right.

satisfied requests cannot be reconstructed uniquely from their red colored paths). So far, this is only guaranteed to be clear for horizontally and vertically satisfied requests. Therefore, two more colors, yellow and cyan, get introduced. The meaning of a yellow or cyan edge is that the next edge in clockwise direction belongs to the same path which satisfies a request. For yellow edges, the lower, left vertex of the edge is used as pivot, for cyan ones the opposite vertex. Thus, whenever a satisfied request uses two edges orthogonal to each other, exactly one of them is the appropriate one to be colored yellow or cyan, and otherwise, in case the path goes straight through at this vertex, we conclude this implicitly from the missing additional color (see Figure 21).

Now, we are finally able to identify the beginning of a path, and then, to follow the path all the way to the end of it. Note that all edges are colored in one of the two green shapes, perhaps also red, and in case that there is red already possible either yellow or cyan, as well. Therefore, we can view this as an edge-coloring with "colors" lightgreen, teal, (lightgreen, red), (teal, red), (lightgreen, red, yellow), (teal, red, yellow), (lightgreen, red, cyan), (teal, red, cyan), yielding eight colors. Since the number of colors, and thus the number of needed bits per color, is settled, the oracle does not need to encode this number. Hence, considering any arbitrary, but fixed order of the edges, the oracle can write the color of each edge on the advice tape using $\log_2(8)|E| = 3|E|$ bits. Then, there is some online algorithm Alg with advice for CAPG that reads these bits and reconstructs the coloring.

Then, on an incoming request, Alg checks whether the end vertices lie on the border between a lightgreen and a teal edge, and rejects if not. In case that the request can be satisfied with a horizontally or vertically aligned path, Alg examines whether all edges of the respective row or column that lie between the end vertices have the same green shape, and accepts it in that way if so. Otherwise, Alg verifies whether there is a path indicated by the red color, that matches on the end vertices and whose bends comply with the definition of the yellow and cyan colors. If that is the case, Alg grants the request according to this

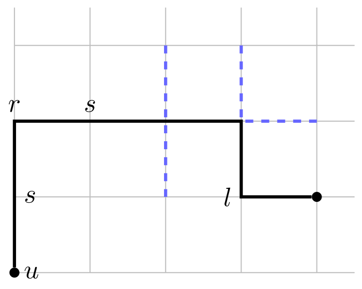

Figure 23 The dashed, blue lines indicate formerly satisfied requests. For a start vertex of a path, N, l, u, r, d are interpreted as "not satisfied", "left", "upwards", "right", or respectively "downwards", and, for an inner vertex v, l, s, r are interpreted as "left", "straight" and "right" with respect to the incoming edge at v. Then, on an incoming request $((1,1),(2,5))$ the string $usrsl$ translates to the depicted path.

established path. Obviously, Alg satisfies exactly the requests that are satisfied in $S(I)$, and hence, Alg is optimal reading only $3|E|$ bits of advice. ◀

As we are now able to realize, the proof is quite close to a simple combination of the principles used to solve the extreme cases and indeed results in the plain addition of the encoding sizes.

One can continue along the same line of reasoning to obtain a slightly different result, bounding the number of advice bits from above in the number of requests: It is sufficient to know for every request whether it has been taken and then to be able to follow the request until its end, in order to reconstruct an optimal solution. Hence, for the first vertex of a request, the oracle can encode whether to reject it or one of the four directions to start, and, for every inner vertex of the path that satisfies the request, the oracle can encode one of the three directions left, straight, or right to continue. Moreover, given that, for a path satisfying a request, the incoming direction for a vertex v is known and already two incident edges of v are used to satisfy another request, there is only one direction left, so there is no additional information needed for v. In other words, for a particular solution, every vertex used as an inner vertex of any path satisfying a request only needs advice once (see Figure 23). The next theorem makes this observation rigorous.

▶ **Theorem 34.** *Let $\mathcal{I}$ denote all possible instances of CAPG on a grid $G = (V, E)$, let $\mathcal{S}_{\text{opt}}(I)$ be the set of optimal solutions for an instance $I \in \mathcal{I}$, and let $V_{\text{inner}}(S) \subseteq V$ be the set of inner vertices of the paths used to satisfy requests in a solution $S \in \mathcal{S}_{\text{opt}}(I)$. Then, there is an optimal online algorithm with advice for CAPG that uses at most*

$$\min_{S \in \mathcal{S}_{\text{opt}}(I)} a(k, |V_{\text{inner}}(S)|)$$

advice bits, where

$$a(k, |V|) = \lceil \log_2(5) \cdot k + \log_2(3) \cdot |V| \rceil + \lceil 2\log_2(k) \rceil + \lceil 2\log_2(|V|) \rceil,$$

and k is the number of requests in I.

Proof. Given an instance $I \in \mathcal{I}$, the oracle can compute all optimal solutions $\mathcal{S}_{\text{opt}}(I)$ and choose one $S \in \mathcal{S}_{\text{opt}}(I)$ that minimizes $a(k, |V_{\text{inner}}(S)|)$. Further, recalling the numbering of the vertices of G (see Theorem 4), we consider an ordering Q of the vertices from lower-left vertices to upper-right ones, i.e., a vertex $v_{a,b} \in V$ precedes another vertex $v_{x,y} \in V$ if and only if $a < x \vee (a = x \wedge b < y)$. Note that this ordering induces a direction for paths, where the path starts at the outer vertex coming first in the ordering and ends at the other outer vertex.

Then, the oracle creates a string t in the following way: It starts from an empty string. For every request r in I, first, an N is appended if the request gets rejected in S, and else, in case r gets granted via a path p, the oracle appends either an l, u, r, or d depending on whether the first edge of it lies to the left, points upwards, to the right, or downwards, relative to the start vertex of p. Secondly, the oracle traverses p and for every inner vertex $v \in V_{\text{inner}}(S)$ it verifies whether v is already an inner vertex of another path p' used to satisfy a preceding request. If so, the oracle continues with the next vertex. Otherwise, it adds either an l, s, or r to t, according to whether p continues to the left, straight, or to the right relative to the incoming edge of p at v.

Hence, overall the oracle constructs a string of length $k + |V_{\text{inner}}(S)|$. Since there are at most $|\{N, l, u, r, d\}|^k \cdot |\{l, s, r\}|^{|V_{\text{inner}}(S)|} = 5^k \cdot 3^{|V_{\text{inner}}(S)|}$ such strings (not all of these combinations are valid constructions), the oracle can enumerate them from 0 to at most $5^k \cdot 3^{|V_{\text{inner}}(S)|}$ in any arbitrary, but fixed order, and write the index of t on the advice tape. For this, $\lceil \log_2(5^k \cdot 3^{|V_{\text{inner}}(S)|}) \rceil = \lceil \log_2(5) \cdot k + \log_2(3) \cdot |V_{\text{inner}}(S)| \rceil$ bits are sufficient. In order to convey k and $|V_{\text{inner}}(S)|$ at the very beginning of the advice tape using a self-delimiting encoding, no more than $\lceil 2\log_2(k) \rceil + \lceil 2\log_2(|V_{\text{inner}}(S)|) \rceil$ bits are used (see Subsection 2.4). Therefore, in total, the oracle writes $a(k, |V_{\text{inner}}(S)|)$ bits of advice on the dedicated tape.

Now, there is an online algorithm Alg with advice for CAPG that reads the advice bits encoding k and $|V_{\text{inner}}(S)|$, calculates $a(k, |V_{\text{inner}}(S)|)$, and reads that many advice bits. By considering the same enumeration of the strings as the oracle, Alg is able to reconstruct t. Then, it proceeds in the following manner: For every incoming request r, Alg reads the next character c from t. If $c = N$, then Alg rejects r, and else, r gets satisfied in S with a path p and $c \in \{l, u, r, d\}$ indicates the direction of the first edge of p. Hence, Alg can traverse this edge arriving at a vertex v. Imagine the current v is not the end vertex of r. If v has been an inner vertex of a path p' used previously by Alg to satisfy another request, then subtracting the traversed edge, there is only one edge left, so the direction in which p continues is clear. Otherwise, Alg obtains the direction from reading the next character $c \in \{l, s, r\}$ of t. Either way, Alg traverses the next edge of p arriving at a new vertex, which it takes as new v and repeats the former steps. If v is the second end vertex of p, Alg has learned the whole path p and satisfies r accordingly.

Clearly, Alg reads the first $a(k, |V_{\text{inner}}(S)|)$ bits of advice and recomputes the optimal solution S. ◀

Let us remark that this proof exploits the order of requests in I, since only at the first time that some vertex v is an inner vertex of some path used to satisfy a request of I, advice has to be given. The following corollary follows immediately, since $V_{\text{inner}}(S) \subseteq V$ and $|V|$ is known by the algorithm, and is more applicable considering how difficult it is to establish the exact cardinality of $V_{\text{inner}}(S)$ as a function of k or $|E|$.

▶ **Corollary 35.** *Let $\mathcal{I}$ denote all possible instances of CAPG on a grid $G = (V, E)$, and let $\mathcal{S}_{\text{opt}}(I)$ be the set of optimal solutions for an instance $I \in \mathcal{I}$. Then, there is an optimal online algorithm with advice for CAPG that uses at most*

$$\lceil \log_2(5) \cdot k + \log_2(3) \cdot |V| \rceil + \lceil 2\log_2(k) \rceil$$

advice bits, where k is the number of requests in I. ☐

Naturally, the question arises for which k Theorem 35 constitutes a better upper bound than Theorem 33. Therefore, by omitting low order terms for simplicity, and since one bound monotonically increases in k, while the other one is constant in k, the threshold is given by the solution of $\log_2(5) \cdot k + \log_2(3) \cdot |V| \overset{!}{=} 3 \cdot |E|$ with respect to k, which results in

$$k = \frac{3|E| - \log_2(3)|V|}{\log_2(5)} = \frac{(6 - \log_2(3))|E| - \log_2(3)(m+n)}{2\log_2(5)}.$$

Hence, for fewer than approximately $0.95|E| - 0.34(m+n)$ requests, Theorem 35 is indeed an improvement, otherwise (e.g., in the worst case of both bounds) Theorem 33 yields the stronger result, since there can be up to $\binom{|V|}{2} = \frac{1}{8}(|E| + m + n - 1)^2 - \frac{1}{8}$ requests in an instance.

4 Conclusion

As desired we showed lower and upper bounds for the number of advice bits concerning optimality, and also examined the case of suboptimal algorithms achieving a certain competitive ratio.

With a lower bound of $m \cdot \log_2(t_n) + n \cdot \log_2(t_m) > 0.94677 \cdot |E| - m - n$ advice bits (see Theorem 23 and Theorem 24) and an upper bound of $3|E|$ advice bits (see Theorem 33) in the general case, the results regarding optimality are already fairly close. However, though this shows that CAPG is roughly as hard as DPA, so far the bounds are not tight and in particular it is still open whether CAPG needs strictly more advice than DPA. Since hitherto, in our best lower bound, we solely consider horizontally or vertically aligned requests, respectively, one natural attempt to prove a stronger result would be to additionally include non-aligned requests as well. For example, we could examine all aligned requests of length at most 2 and then possibly extend some of these requests with an orthogonal edge at one or both of its end vertices. Then this would again result in requests of length at most 4. Note that, at each end vertex, there are two orthogonal choices to extend the request (except for some corner cases), thus this yields a significant growth of the degrees of freedom to construct an instance, i.e., the total number of such instances rises considerably. Unfortunately, already the requests constructed from horizontal requests may cover the whole grid, so, in some cases, the optimal solution would not contain any request constructed from vertical ones. Similarly, the extension of requests on the same row or on neighboring rows can overlap. Hence, it is rather difficult to establish the number of optimal solutions per instance, or the optimal solutions overall. At the same time, it also seems to be hard to find a subset of such instances that can be analyzed with a reasonable effort and still is large enough to improve on our result. Therefore, capturing the essential difference between CAPG and DPA in a lower bound remains unsolved.

Another interesting thought one could pursue in more detail is that most of our upper bounds seem to be easily generalizable to other graphs. For example, a cycle graph as underlying network has $\chi(\widehat{G}) \leq 3$, so by Theorem 30, we instantly obtain an upper bound of $\lceil \log_2(3)|E| \rceil$ advice bits for optimality. On the other hand, without loss of generality, we may assume that a request of length 1 is granted by every optimal solution, and although there are two possibilities to satisfy such a request in a cycle graph, recalling that no request repeats, we may assume the path of length 1 is actually used. Then, granting the request "cuts" the cycle in a path of length $|E| - 1$, so using the tight lower bound for DPA on this subgraph, we obtain that at least $|E| - 2$ advice bits are necessary to compute an optimal solution. Hence, the upper bound is already tight up to a constant factor.

The universal character of most of the upper bounds suggests that similar contemplations are possible for a large number of underlying graphs, especially related ones such as cylinder and torus graphs.

References

1 Kfir Barhum, Hans-Joachim Böckenhauer, Michal Forisek, Heidi Gebauer, Juraj Hromkovič, Sacha Krug, Jasmin Smula, and Björn Steffen. On the power of advice and randomization for the disjoint path allocation problem. In *SOFSEM 2014: Theory and Practice of Computer Science - 40th International Conference on Current Trends in Theory and Practice of Computer Science, Nový Smokovec, Slovakia, January 26–29, 2014, Proceedings*, pages 89–101, 2014.

2 Hans-Joachim Böckenhauer, Juraj Hromkovič, Dennis Komm, Sacha Krug, Jasmin Smula, and Andreas Sprock. The string guessing problem as a method to prove lower bounds on the advice complexity. *Theor. Comput. Sci.*, 554:95–108, 2014.

3 Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královic, Richard Královic, and Tobias Mömke. On the advice complexity of online problems. In *Algorithms and Computation, 20th International Symposium, ISAAC 2009, Honolulu, Hawaii, USA, December 16–18, 2009. Proceedings*, pages 331–340, 2009.

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

5 Joan Boyar, Lene M. Favrholdt, Christian Kudahl, and Jesper W. Mikkelsen. The advice complexity of a class of hard online problems. *CoRR*, abs/1408.7033, 2014.

6 Stefan Dobrev, Rastislav Královic, and Dana Pardubská. How much information about the future is needed? In *SOFSEM 2008: Theory and Practice of Computer Science, 34th Conference on Current Trends in Theory and Practice of Computer Science, Nový Smokovec, Slovakia, January 19–25, 2008, Proceedings*, pages 247–258, 2008.

7 Gregory P. B. Dresden and Zhaohui Du. A simplified binet formula for k-generalized fibonacci numbers. *Journal of Integer Sequences*, 2014.

8 Juraj Hromkovič. *Theoretical Computer Science: Introduction to Automata, Computability, Complexity, Algorithmics, Randomization, Communication, and Cryptography (Texts in Theoretical Computer Science. An EATCS Series)*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2010.

9 Juraj Hromkovič, Rastislav Královic, and Richard Královic. Information complexity of online problems. In *Mathematical Foundations of Computer Science 2010, 35th International Symposium, MFCS 2010, Brno, Czech Republic, August 23–27, 2010. Proceedings*, pages 24–36, 2010.

10 Dennis Komm. *An Introduction to Online Computation – Determinism, Randomization, Advice*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2016.

11 Tony Noe, Tito III Piezas, and Eric W. Weisstein. Fibonacci n-step number. From MathWorld--A Wolfram Web Resource. `http://mathworld.wolfram.com/Fibonaccin-StepNumber.html`. Last visited on 01.06.2017.

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

13 Neil J. A. Sloane. The on-line encyclopedia of integer sequences. `http://oeis.org/A000078`. Last visited on 26.06.2017.

14 Jasmin Smula. *Information content of online problems: Advice versus determinism and randomization*. PhD thesis, ETH Zurich, 2015.

15 Björn Steffen. Advice complexity of online graph problems. Bachelor's thesis, ETH Zurich, 2014.