Theoretical Computer Science 862 (2021) 81–96

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

On the advice complexity of the online dominating set problem

Hans-Joachim Böckenhauer [a,*], Juraj Hromkovič [a], Sacha Krug [a,1], Walter Unger [b]

[a] Department of Computer Science, ETH Zurich, Switzerland
[b] Department of Computer Science, RWTH Aachen University, Germany

ARTICLE INFO

Article history:
Received 19 June 2020
Received in revised form 26 November 2020
Accepted 14 January 2021
Available online 19 January 2021

Keywords:
Advice complexity
Online computation
Competitive analysis
Dominating set

ABSTRACT

A dominating set S of a graph is a set of vertices such that each vertex is in S or has a neighbor in S. The goal of the dominating set problem is to find such a set of minimum cardinality. In the online setting, the graph is revealed vertex by vertex, together with edges to all previously revealed vertices.

Advice complexity is a framework to measure the amount of information an online algorithm is lacking. Here, an online algorithm reads advice bits from an infinite binary tape prepared beforehand by an all-knowing oracle. The advice complexity is the total number of advice bits read during the computation. Besides giving some insight into what makes an online problem hard, advice complexity can also be used as a means for proving lower bounds on the competitive ratio achievable by randomized online algorithms.

We analyze the advice complexity of the online dominating set problem. For general graphs, we show tight upper and lower bounds for optimality. Then, we use a result for c-competitiveness to prove that no randomized online algorithm can be better than $n^{1-\varepsilon}$-competitive, for any $\varepsilon > 0$. Finally, we analyze the advice complexity of various graph classes for optimality.

1. Introduction

In classic offline computation, an algorithm gets an input and produces an output. In online computation, however, the scenario is different. Here, the input does not arrive all at once, but in discrete time steps. An online algorithm has to produce a partial output in every time step without knowing anything about future parts of the input. Moreover, the algorithm may not revoke its decision later on. The performance of such an algorithm is usually measured by comparing its output to that of an optimal offline algorithm. A detailed introduction to online computation can be found, e. g., in the books by Borodin and El-Yaniv [12] or by Komm [28].

Clearly, an online algorithm has a significant inherent disadvantage compared to an offline algorithm. There is a huge gap between knowing nothing about the future and knowing everything about it. To make this transition smoother, *online computation with advice* has been introduced as a new framework to analyze the lack of information of an online algorithm compared to an offline algorithm [8,9,18,22,26]. In this model, an online algorithm has access to an infinite binary *advice*

* Corresponding author.
E-mail addresses: hjb@inf.ethz.ch (H.-J. Böckenhauer), juraj.hromkovic@inf.ethz.ch (J. Hromkovič), quax@algo.rwth-aachen.de (W. Unger).
[1] Deceased 20 November 2015.

https://doi.org/10.1016/j.tcs.2021.01.022

Table 1
Upper and lower bounds on the advice complexity for optimality.

Graph class	Lower bound	Upper bound
General graphs	$n-1$	n
Paths	$0.5283n-\Theta(1)$	$0.5283n+\Theta(\log n)$
Cycles	$0.5283n-\Theta(1)$	$0.5283n+\Theta(\log n)$
Tori	$0.3610n-\Theta(\log n)$	$0.6490n+\Theta(\log n)$
Maximal outerplanar graphs	$0.5n-\Theta(1)$	$0.9183n+\Theta(\log n)$
Graphs of maximum degree d	$(1-2/(d+1))n$	n

tape during the computation. This tape is prepared before the computation by an all-knowing oracle. The *advice complexity* is the number of advice bits an online algorithm needs to read from the tape during the computation to achieve a certain solution quality. In some sense, this captures the amount of knowledge an online algorithm is lacking about the future. Various online problems have been studied in this new setting, amongst them k-server [7,23,31], bin packing [17], buffer management [19], paging [8,9], graph coloring [4,33], string guessing [5], and call admission problems [2,3,11]. Moreover, the advice complexity of a class of several hard optimization problems, including the dominating set problem, was investigated in both their unweighted [15] and weighted versions [16]. For an overview, see also the survey by Boyar et al. [14].

There is a straightforward relationship between advice and randomization. Since an advice string of a certain length may be seen as the best choice among all possible random strings of the same length, upper bounds on the number of used random bits directly carry over to advice complexity, whereas lower bounds on the advice complexity also hold for the number of random bits. Interestingly enough, however, Böckenhauer et al. [7] showed that a randomized online algorithm using an arbitrary number of random decisions can be used to construct a deterministic online algorithm reading only logarithmically (in the input size) many advice bits, as long as the set of admissible inputs is not too dense. On the other hand, algorithms with advice can, under certain circumstances, be used to design randomized algorithms [30].

In this paper, we consider the dominating set problem, which consists of finding a subset S of vertices in a graph such that every vertex is in S or has a neighbor in S. This is a well-known graph-theoretic problem that has been studied since the 1950s. Applications can be found, e. g., in the famous queens problem, communications network design, placement of radio antennae, or social theory. Haynes et al. [24] provide a good overview.

We analyze the advice complexity of the online dominating set problem (DominatingSet) in the following model: A graph is revealed vertex by vertex; together with a vertex, all edges to previously revealed vertices are revealed.

The online dominating set problem belongs to a class of online problems that bear a certain asymmetry in the following sense. A single wrong decision during the online computation may increase the cost of the computed solution by a factor linear in the input size. This is also true, e. g., for the vertex cover problem [34] or the knapsack problem [10], but is not the case with paging, graph coloring, or string guessing. The advice complexity of the class of online problems with this asymmetric behavior has also been studied by Boyar et al. [15]. In some sense, the asymmetry is even stronger in the case of the dominating set problem on general graphs. If the resulting graph is allowed to have isolated vertices, then every online algorithm without advice (even when randomized) is forced to take every vertex into the dominating vertex that is isolated in the partial graph at the time it is presented; otherwise the algorithm risks to compute an infeasible solution. To the best of our knowledge, this is the first analysis of the advice complexity of a problem with this asymmetric property.

The paper is organized as follows. In Section 2, we give an overview of previous work and introduce all important concepts and definitions. Section 3 contains our results for general graphs. Section 4 is devoted to randomized algorithms. In Sections 5 to 8, we present optimality results for specific graph classes. These results are summarized in Table 1.

2. Preliminaries and related work

King and Tzeng [27] show with a simple worst-case example that the upper bound of $n-1$ on the competitive ratio, i. e., the ratio between the cost of the computed solution and the cost of an optimal solution, (obtained by an algorithm that just accepts every not yet dominated vertex) is tight. Consider the instance where every revealed vertex is connected to all previously revealed ones until the algorithm rejects some vertex v. (If this does not happen, then the instance at hand is K_n, whose optimal solutions have size 1. Yet the algorithm has accepted all n vertices and is thus even only n-competitive.) Then, all remaining vertices are connected only to v and thus must be accepted; the computed solution has therefore cost $n-1$. The optimal solution, however, contains only v and has cost 1. (Note that the authors also use another online setting, where, together with a vertex, *all* its adjacent edges are revealed, also those to not yet revealed vertices.)

Eidenbenz [20] refines these results by investigating DominatingSet on specific graph classes. He also considers two online models. In the *insertion model,* the adversary reveals (or inserts) a vertex in one time step. In the *insertion/deletion model,* the adversary inserts or removes a vertex from the graph in one time step. In both models, an algorithm may not reject a previously accepted vertex. Moreover, the accepted vertices must form a dominating set *in every time step.* For the insertion model, the following results are presented. For trees, the competitive ratio is bounded from above by $3-\varepsilon$, for an arbitrarily small $\varepsilon > 0$, and by 2 from below. Also, there are online algorithms with competitive ratio $n-1$ for forests as well as for bounded-treewidth, bipartite, planar, and variable-size disk graphs, and this bound is tight for all these graph classes.

These models, where a dominating set has to be maintained in every step of the computation, have also been studied in depth in a recent work by Boyar et al. [13].

Instead, we use King and Tzeng's model, where a dominating set has to be chosen by the algorithm only after the whole graph has been presented. Formally, it is defined as follows.

Definition 1. The *online dominating set problem (*DominatingSet*)* is the following online minimization problem on a graph G with n vertices. In time step i, for $1 \leq i \leq n$, a vertex v from G is revealed together with all edges to previously revealed vertices. An online algorithm for DominatingSet has to decide immediately whether to accept or reject v, i. e., whether v is part of the solution or not. The goal is to obtain a dominating set of minimal size for G.

Definition 1 allows the computed solution to be invalid in intermediate time steps. This is one reasonable way to avoid that isolated vertices always have to be accepted.

Definition 2. The *online dominating set problem on the graph class C* is DominatingSet restricted to input graphs from C.

Note that Definition 2 says nothing about the graphs in intermediate time steps. In particular, they do not have to be from C. For instance, if we consider trees, the intermediate graphs do not need to be connected.

These two definitions are consistent in the following sense. On the one hand, we give a certain freedom to an online algorithm for DominatingSet: Only its final output has to be a valid solution. On the other hand, we also give a certain freedom to the adversary: Only its final input has to be a valid graph.

Let us now formally define the advice complexity framework we are using.

Definition 3. Consider an input sequence $I = (x_1, \ldots, x_n)$. An *online algorithm* $\mathtt{A}$ *with advice* computes the output sequence $\mathtt{A}^{\phi}(I) = (y_1, \ldots, y_n)$ such that y_i is computed from $\phi, x_1, \ldots, x_i$, where ϕ is the content of the advice tape, i. e., an infinite bit string. The algorithm $\mathtt{A}$ is *c-competitive with advice complexity $s(n)$* if, for every n and every input sequence I of length at most n, there is some ϕ such that $\mathrm{cost}(\mathtt{A}^{\phi}(I)) \leq c \cdot \mathrm{cost}(\mathcal{O}pt(I)) + \alpha$, for some non-negative constant α, and at most the first $s(n)$ bits of ϕ have been accessed during the computation of $\mathtt{A}$ on I. If $\alpha = 0$, then we say that $\mathtt{A}$ is *strictly* c-competitive.

In what follows, $\mathbb{N} := \{0, 1, \ldots\}$ and $\mathbb{N}^{\geq k} := \{n \in \mathbb{N} \mid n \geq k\}$, for any $k \in \mathbb{N}$. Moreover, $\mathbb{N}^{+} := \mathbb{N}^{\geq 1}$. We denote the binary logarithm of x by $\log x$. The *size* of a graph is its number of vertices.

Note that we can encode a positive integer n on the advice tape in a self-delimiting way using

$$\mathrm{Log}\, n := \max\{2, \lceil \log n \rceil + 2\lceil \log \lceil \log n \rceil \rceil\}$$

advice bits [21], see also [28].

Before we start, we briefly consider the encoding of n q-ary decisions on the (binary) advice tape, for some $q \in \mathbb{N}^{\geq 3}$. Essentially, they can be encoded in such a way that the amount of wasted advice bits tends to 0 asymptotically.

If n is known in advance, then an algorithm simply reads $\lceil n \log q \rceil$ advice bits from the tape. Since $2^{\lceil n \log q \rceil} \geq q^n$, every possible sequence of q-ary decisions of length n can be encoded with a different advice string on the tape. Moreover, one easily verifies that this bound is tight, i. e., that $\lceil n \log q \rceil$ advice bits are necessary and sufficient.

If n is not known in advance, then the first $\mathrm{Log}\, n$ advice bits encode the number n in a self-delimiting way. After having read these advice bits and thus knowing n, an algorithm proceeds as above. The additional advice bits are asymptotically irrelevant since $\lim_{n \to \infty} \frac{\mathrm{Log}\, n + \lceil n \log q \rceil}{\lceil n \log q \rceil} = 1$.

We often consider the following very simple greedy algorithm.

Definition 4. The *greedy algorithm* $\mathtt{G}$ accepts a newly revealed vertex if and only if it is not dominated by an already accepted vertex.

3. General graphs and trees

The greedy algorithm $\mathtt{G}$ rejects at least one vertex whenever the graph contains an edge. Otherwise, the graph consists only of isolated vertices, and $\mathtt{G}$ is optimal. Moreover, it is known that there is no better deterministic online algorithm for DominatingSet on general graphs [27].

Observation 1. *The greedy algorithm $\mathtt{G}$ is strictly $(n-1)$-competitive, for $n \geq 2$, and this bound is tight for deterministic online algorithms without advice.* □

Considering optimal algorithms, we get the following almost matching bounds.

Theorem 1. *For* DominatingSet*, n advice bits are sufficient and $n - 1$ advice bits are necessary to be optimal, for $n \geq 5$.*

Proof. For the upper bound, an algorithm simply reads one advice bit per revealed vertex and accepts it if and only if the bit is 1.

For the lower bound, the adversary reveals $n-1$ isolated vertices $v_1, \ldots, v_{n-1}$. Afterwards, it reveals a last vertex v_n that is connected to a subset $V' \subseteq \{v_1, v_2, \ldots, v_{n-1}\}$ of size at least 2. This implies that the unique optimal solution is $V \setminus V'$, because any optimal solution must contain all isolated vertices, i. e., all vertices in $\{v_1, v_2, \ldots, v_{n-1}\} \setminus V'$, plus at least one vertex from the component C induced by $V' \cup \{v_n\}$. Because $|C| \geq 3$, the only vertex that dominates all vertices in C is v_n.

Every choice of V', i. e., every possible input instance, requires a different output of the algorithm on the first $n-1$ vertices, and any two instances are indistinguishable until the very last vertex is revealed. There are $2^{n-1} - n$ possible subsets V', and we have

$$\lceil \log(2^{n-1} - n) \rceil > \lceil \log(2^{n-1} - 2^{n-2}) \rceil = \lceil \log(2^{n-2}) \rceil = n - 2,$$

where the inequality holds for all $n \geq 5$. Thus, at least $n-1$ advice bits are necessary. □

The following tradeoff is known about algorithms with advice.

Theorem 2 *(Boyar et al. [15]). To achieve a strict competitive ratio of $c > 1$, where c may be a function of n,*

$$\log\left(1 + \frac{(c-1)^{c-1}}{c^c}\right) n + d \log n$$

advice bits are sufficient and

$$\log\left(1 + \frac{(c-1)^{c-1}}{c^c}\right) n - d' \log n$$

advice bits are necessary, for non-negative constants d, d'. □

While the proof of Theorem 2 makes heavy use of isolated vertices, we can prove a slightly weaker lower bound for connected graphs, even for trees, for any strict competitive ratio smaller than 2. To this end, we use a reduction to the bit guessing problem with unknown history as defined by Böckenhauer et al. [5,6].

Definition 5 *(Böckenhauer et al. [5,6]).* The *bit guessing problem with unknown history* (BGUH) is the following online minimization problem. The input $I = (n, ?, ?, \ldots, ?, d)$ consists of the input size n in the first request, $n-1$ subsequent requests "?" carrying no extra information, and the correct string $d = d_1 \ldots d_n \in \{0,1\}^n$. In each of the first n time steps, the online algorithm A is required to output one character from Σ, forming the output sequence $\mathrm{A}(I) = y_1 y_2 \ldots y_n$. The algorithm is not required to respond with any output in the last time step, where the string d is revealed. The cost of a solution $\mathrm{A}(I)$ is the Hamming distance between d and $\mathrm{A}(I)$.

Theorem 3 *(Böckenhauer et al. [5,6]). Consider an input string of length n for* BGUH*, for some $n \in \mathbb{N}$. Any online algorithm that is correct in more than γn bits, for $1/2 \leq \gamma < 1$, needs to read at least*

$$(1 + (1-\gamma)\log_2(1-\gamma) + \gamma \log_2 \gamma) n = (1 - H_2(1-\gamma)) n$$

advice bits, where $H_2(p) = -p \log_2 p - (1-p)\log_2(1-p)$ denotes the binary entropy function.

Using Theorem 3, we can now prove the following lower bound on the advice complexity of DominatingSet on trees.

Theorem 4. *To achieve a strict competitive ratio of $1 + \alpha$ where $0 < \alpha \leq 1/2$ for* DominatingSet *on trees, $\frac{1}{4} \cdot (1 - H_2(\alpha))n$ advice bits are necessary.*

Proof. We prove the claim by a so-called advice-preserving reduction, see [28], from BGUH. We reduce BGUH to DominatingSet on a tree in the following way. Let BBDS be some online algorithm with advice for DominatingSet on trees, which we will use as a black box for constructing an advice algorithm BitGuess for BGUH. Whenever BBDS asks for advice, BitGuess provides it from its own advice tape.

Let $I = (n, ?, ?, \ldots, ?, d)$ be an instance of BGUH, where $d = d_1 d_2 \ldots d_n$. Let r_i denote the ith request of I. For $1 \leq i \leq n$, BGuess answers r_i in the following way. It sends two requests to BBDS, consisting of two vertices u_i and v_i that are connected to each other by an edge, but are not yet connected to any other vertices.

BBDS answers these two requests with taking either u_i or v_i or both into the dominating set. Since taking none of the two vertices into the dominating set would result in an infeasible solution and we assumed BBDS to be a correct algorithm for DominatingSet on trees, this output is not possible. BGuess interprets this output as follows: If u_i (or both vertices) are taken, it guesses the bit 0; otherwise, it guesses the bit 1. With the request r_{n+1}, BGuess learns which of its guesses were

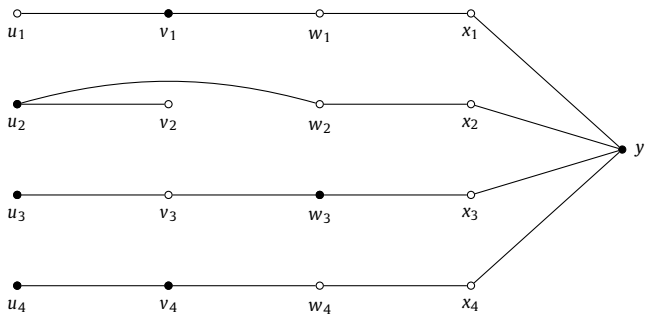

Fig. 1. An example of the construction in the proof of Theorem 4 for $d = 1011$ and guessed bit string 1000. The filled vertices belong to the dominating set.

correct. It then connects the components $\{u_i, v_i\}$ by a tree as shown in Fig. 1. In particular, the vertex w_i is connected to u_i if $d_i = 0$, and to v_i if $d_i = 1$.

Whenever BBDS chose exactly one of the vertices u_i and v_i and this choice led to the correct guess of the ith bit, the vertex w_i is already dominated. If the choice of BBDS led to the wrong guess for the respective bit or both u_i and v_i were chosen, this leads to two vertices from $\{u_i, v_i, w_i\}$ being in the dominating set, i.e., BBDS puts one more vertex into the dominating set than it would have been necessary.

Whenever the guess of BGUESS on request r_i was correct and BBDS did not choose both u_i and v_i for inclusion into the dominating set, only one vertex is necessary to dominate $\{u_i, v_i, w_i\}$. In all other cases, BBDS needs two vertices to dominate this set.

An optimal dominating set for the constructed tree has a size of $n+1$; it contains the vertex y and, for all $i \in \{1, \ldots, n\}$, one of the vertices u_i and v_i.

Assume that BBDS achieves a strict competitive ratio of less than $1+\alpha$, for some α satisfying $0 < \alpha \leq 1/2$. This implies that it computes a dominating set of size less than $(1+\alpha)(n+1)$. Thus, at least a fraction of $\gamma = 1-\alpha$ of all sets $\{u_i, v_i, w_i\}$ are dominated by a single vertex. Hence, BGUESS guesses at least a fraction of γ bits correctly.

According to Theorem 3, this implies that BGUESS uses at least $(1 - H_2(1-\gamma))n$ advice bits on the n requests of I. Thus, BBDS uses at least $(1 - H_2(1-\gamma))n = (1 - H_2(\alpha))n$ advice bits on $4n+1$ requests, i.e., at least $\frac{1}{4} \cdot (1 - H_2(\alpha))n$ advice bits on n requests. □

4. Randomization

In this section, we prove lower bounds on the competitive ratio of any randomized online algorithm. More precisely, we analyze the expected competitive ratio of online algorithms that have access to a source of random bits. Here, we are again interested in the worst-case behavior for all instances of a certain length; the expected value is taken over all possible computations for an instance.

In the general case, where instances containing isolated vertices are allowed, it is easy to see that also for randomized algorithms the lower bound on the competitive ratio from Theorem 1 remains valid. Indeed, on the first $n-1$ requests, a randomized algorithm R cannot distinguish between the following two instances: I_1 consisting of n isolated vertices and I_2 consisting of a star in which the first $n-1$ vertices are connected to a center vertex presented with the last request. To compute a valid solution for I_1, it has to take each of the first $n-1$ vertices with probability 1, but then it achieves a strict competitive ratio of $n-1$ on I_2.

In the following, we show that also in the case of connected graphs, or even in the case of trees, we can prove a strong lower bound. To this end, we use Theorem 4 and the following result.

Lemma 1 *(Böckenhauer et al. [7]). Consider an online minimization problem U, and let $I(n)$ be the number of all possible inputs of length n. Furthermore, suppose that there is a randomized online algorithm for U with worst-case expected competitive ratio at most E. Then, for any fixed $\varepsilon > 0$, it is possible to construct a deterministic online algorithm that uses at most*

$$\log n + 2\log\log n + \log\left(\frac{\log I(n)}{\log(1+\varepsilon)}\right) + d$$

advice bits, for a constant d, and achieves a competitive ratio of $(1+\varepsilon)E$. □

Together with Theorem 2, we obtain the following lower bound.

Theorem 5. *Every randomized online algorithm for* DOMINATINGSET *on trees has a worst-case expected competitive ratio of at least $\frac{3}{2} \cdot (1-\delta)$ on sufficiently large instances, for any arbitrarily small $\delta > 0$.*

Proof. We prove the claim for DominatingSet restricted to the set of instances $\mathcal{I}(n)$ as constructed in the proof of Theorem 4. For every $n \in \mathbb{N}$, this set contains 2^n instances of length $n' = 4n+1$ each and thus $I(n') = 2^{(n'-1)/4}$ instances of size n'.

From Theorem 4, we know that

$$\frac{n}{4} \cdot (1 - H_2(\alpha)) = \frac{n}{4} \cdot (1 + \alpha \log \alpha + (1-\alpha)\log(1-\alpha)) \tag{1}$$

advice bits are not sufficient to achieve a strict competitive ratio of $1+\alpha$, for any $0 < \alpha < 1/2$.

We consider $\alpha = 1/2 - \varepsilon$, for some $\varepsilon > 0$. Assume that there exists a randomized algorithm R for DominatingSet on $\mathcal{I}(n)$ that achieves a strict competitive ratio of $(3/2-\varepsilon)/(1+\varepsilon)$. According to Lemma 1, this implies the existence of a deterministic algorithm A for $\mathcal{I}(n)$ that achieves a strict competitive ratio of $(3/2-\varepsilon)$ using at most

$$\log n + 2\log\log n + \log\left(\frac{\log I(n)}{\log(1+\varepsilon)}\right) + d < 2\log n + 2\log\log n - \log\log(1+\varepsilon) + d \tag{2}$$

advice bits, for some constant d. Comparing (2) with (1) yields that there exists some n_ε such that, for all $n > n_\varepsilon$,

$$\begin{aligned} &2\log n + 2\log\log n - \log\log(1+\varepsilon) + d \\ &\quad < \frac{n}{4} \cdot (1 + (\tfrac{1}{2}-\varepsilon)\log(\tfrac{1}{2}-\varepsilon) + (\tfrac{1}{2}+\varepsilon)\log(\tfrac{1}{2}+\varepsilon)), \end{aligned}$$

since the left-hand side of this inequality grows only logarithmically with n, whereas the right-hand side grows linearly with n, although the left-hand side contains a large additive constant $-\log\log(1+\varepsilon)$ and the entropy function on the right-hand side contributes a rather small multiplicative constant, for any small constant ε. This is a contradiction; thus, the assumption was false and there does not exist any randomized algorithm for DominatingSet on $\mathcal{I}(n)$ that achieves a strict competitive ratio of $(3/2-\varepsilon)/(1+\varepsilon)$. Substituting $1+\varepsilon = 1/(1-\frac{3}{5}\delta)$ yields the claimed result since

$$\begin{aligned} \left(\tfrac{3}{2}-\varepsilon\right)/(1+\varepsilon) &= \left(\frac{3}{2} - \frac{1}{(1-\frac{3}{5}\delta)} + 1\right) \Big/ \left(1/(1-\tfrac{3}{5}\delta)\right) \\ &= \left(\frac{5}{2} - \frac{1}{1-\frac{3}{5}\delta}\right) \cdot \left(1 - \frac{3}{5}\delta\right) \\ &= \tfrac{3}{2} - \tfrac{3}{2}\delta = \tfrac{3}{2}(1-\delta). \quad \square \end{aligned}$$

5. Paths and cycles

We have seen above that DominatingSet is very hard on general graphs, even with advice. Thus, we analyze some classes of restricted graphs in this and the subsequent sections.

First, we consider paths and cycles. As we shall see, already these simple graphs require a linear number of advice bits to achieve optimality. This is certainly a surprising result and motivates our study of this graph class.

In the following, we assume that a path has an orientation and its vertices are numbered from left to right $v_1, \ldots, v_n$.

Theorem 6. *No deterministic online algorithm for* DominatingSet *on paths can be better than strictly* $(1.5-\varepsilon)$*-competitive, for an arbitrarily small* $\varepsilon > 0$.

Proof. Consider as input instance I any path that is revealed from left to right, i. e., in the order of the indices. Clearly, v_1 or v_2 has to be accepted. After that, at least every second vertex v has to be accepted, because an algorithm $\mathtt{A}$ does not know the input size and thus whether $v = v_n$. Therefore, the computed solution has size at least $\lfloor n/2 \rfloor$.

The optimal solution $\mathcal{O}pt(I)$ consists of the vertices v_2, v_5, etc. If $n \not\equiv 0 \pmod 3$, then, without loss of generality, also v_n is part of $\mathcal{O}pt(I)$. Thus, $\mathcal{O}pt(I)$ has always size $\lceil n/3 \rceil$. Hence, for an arbitrarily small $\varepsilon > 0$, the competitive ratio is

$$\frac{\mathrm{cost}(\mathtt{A}(I)) - \alpha}{\mathrm{cost}(\mathtt{Opt}(I))} \geq \frac{\lfloor n/2 \rfloor}{\lceil n/3 \rceil} - \frac{\alpha}{\lceil n/3 \rceil} \geq \frac{\lfloor n/2 \rfloor}{\lceil n/3 \rceil} - \frac{3\alpha}{n} \geq 1.5 - \varepsilon,$$

where the last inequality holds for all sufficiently large n that are multiples of 6. $\square$

The greedy algorithm $\mathtt{G}$ provides an asymptotically matching upper bound.

Theorem 7. *On paths of size n, the greedy algorithm for* DominatingSet *is*

- *strictly* $(1.5(1+1/n))$*-competitive if* $n \equiv 3 \pmod 6$ *and*
- *strictly* 1.5*-competitive otherwise.*

Table 2
Case distinction according to n mod 6.

n mod 6	$\lceil n/2 \rceil$	$\lceil n/3 \rceil$	Competitive ratio $\lceil n/2 \rceil / \lceil n/3 \rceil$
0	$n/2$	$n/3$	$\frac{3}{2}$
1	$(n+1)/2$	$(n+2)/3$	$\leq \frac{3}{2}$
2	$n/2$	$(n+1)/3$	$\leq \frac{3}{2}$
3	$(n+1)/2$	$n/3$	$\frac{3(n+1)}{2n}$
4	$n/2$	$(n+2)/3$	$\leq \frac{3}{2}$
5	$(n+1)/2$	$(n+1)/3$	$\frac{3}{2}$

Proof. The greedy algorithm accepts at most $\lceil n/2 \rceil$ vertices in total, because it never accepts two neighboring vertices.

We know from the previous proof that the optimal solution always has cost $\lceil n/3 \rceil$. A simple case distinction (see Table 2) shows that the competitive ratio is at most 1.5 unless $n \equiv 3 \pmod 6$. In that case, the competitive ratio is $1.5(1 + 1/n)$, i.e., it increases towards 1.5 for increasing n. □

With one advice bit, we can construct an algorithm that is always 1.5-competitive.

Theorem 8. *There is a strictly* 1.5*-competitive online algorithm* A *for* DominatingSet *on paths that reads one advice bit b.*

Proof. The only problematic case in the Theorem 7 is when $n \equiv 3 \pmod 6$, and the only problematic solution there is $v_1, v_3, \ldots, v_n$. Let us therefore call these vertices *bad.* Conversely, the vertices $v_2, v_4, \ldots, v_{n-1}$ are *good.*

If $n \not\equiv 3 \pmod 6$, then $b = 1$. If $n \equiv 3 \pmod 6$, then $b = 1$ if at least one good vertex is revealed isolated, and $b = 0$ otherwise.

The algorithm A reads b at the beginning and behaves greedily if $b = 1$. If $b = 0$, it behaves greedily as well, but with the exception that it rejects all isolated vertices.

We now show that the competitive ratio of A is bounded from above by 1.5. If $n \not\equiv 3 \pmod 6$, we know from the previous proof that A is 1.5-competitive. If $n \equiv 3 \pmod 6$, there are two possibilities.

First, at least one good vertex is revealed isolated. Then, $b = 1$, i.e., A behaves greedily and thus accepts this vertex. This, however, immediately implies that the solution has size at most $\lfloor n/2 \rfloor$, as the only solution of size $\lceil n/2 \rceil$ without neighboring vertices consists of all bad vertices.

Second, no good vertex is revealed isolated. Then, $b = 0$, i.e., A accepts none of the isolated vertices. We show now that this means A accepts all good vertices and rejects all bad vertices, i.e., that, as above, the solution has size at most $\lfloor n/2 \rfloor$. We prove this statement for each component separately by induction on the component size.

Induction base. An isolated vertex is always bad and is never accepted by A.

Induction hypothesis. The statement holds for components of size $n - 1$.

Induction step. Consider a newly revealed vertex v at one end of a component. (It does not matter whether v merges two components, as its "parity", i.e., whether it is good or bad, is the same in both components.) If v is bad, then its neighbor in the component is good and was thus accepted. Hence, v is rejected. If v is good, then its neighbor in the component is bad and was rejected. Hence, v is accepted, because A behaves greedily on vertices that are not revealed isolated. □

We now consider optimality and prove that even the very restricted graph class of paths has a linear advice complexity. We start with the upper bound, using a technique that we will then also use for the lower bound. The upper bound matches the lower bound; this shows that the bounds are tight.

Theorem 9. *For* DominatingSet *on paths,* $(\log 3)n/3 + \Theta(\log n) \approx 0.5283n + \Theta(\log n)$ *advice bits are sufficient to be optimal.*

Proof. As before, we fix the optimal solution $\mathcal{O}pt$ to contain every third vertex, starting with the second from the left: v_2, v_5 etc. Additionally, $\mathcal{O}pt$ contains v_n if $n \not\equiv 0 \pmod 3$. In this case, we call the vertices $v_{3\lfloor n/3 \rfloor + 1}$ and possibly $v_{3\lfloor n/3 \rfloor + 2}$ *dangling.*

First, A reads two advice bits to determine the number of dangling vertices. Then, it reads the indices (with respect to revelation order) of the dangling vertices, if any, from the tape. If $n \equiv 1 \pmod 3$, then A will accept $v_{3\lfloor n/3 \rfloor + 1}$. If $n \equiv 2 \pmod 3$, then A will reject $v_{3\lfloor n/3 \rfloor + 1}$ and accept $v_{3\lfloor n/3 \rfloor + 2}$. Therefore, we can safely ignore the dangling vertices from now on. That is, we assume n to be a multiple of 3 in the remainder of the proof. A reads the number n from the advice tape. Encoding the indices of the dangling vertices and n requires only logarithmic advice [21].

Table 3
Cases when it might be unclear whether to take a vertex. The new vertex is indicated by the ␣; the | indicates the end of the partial path revealed so far. The other vertices are denoted 1 if they were selected.

Arrangement	Clear?	Take vertex?
␣	no	bad loner: use advice
␣-0|	no	bad neighbor: use advice
␣-0-0	yes	yes
␣-0-1	yes	no (yes only in case new vertex is dangling)
␣-1	yes	no
0-␣-0	yes	yes
0-␣-1	yes	no (yes only in case left neighbor 0 is dangling)
1-␣-1	yes	no (can only appear if one neighbor is dangling)

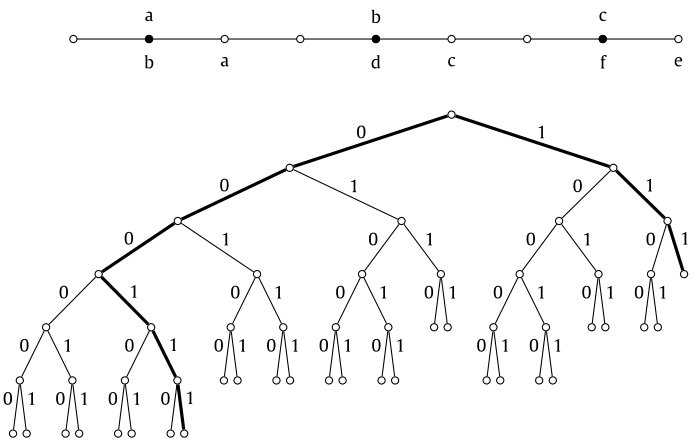

Fig. 2. The letters at the path on top indicate two different revelation orders on the path of size 9. The bold edges in the tree below indicate the respective paths to the leaf that the oracle chooses. Thus, the pseudo-advice strings are 111 and 000111, respectively. Note that no more pseudo-advice is necessary for the remaining vertices.

For the rest of the proof, we need some more terminology. A *bad loner* is a vertex that is revealed isolated. A *bad neighbor* is a vertex v that is revealed as the neighbor of a rejected bad loner u such that the other neighbors of v and u have not yet been revealed. A *bad vertex* is simply a bad loner or a bad neighbor. Let m_1 and m_2 be the number of bad loners and bad neighbors, respectively. A reads the numbers m_1 and m_2 from the tape; this needs again logarithmic advice. We are interested in bad vertices because a simple case distinction (see Table 3) shows that it is only unclear what to do for A if a bad vertex is revealed.

Consider now the following binary edge-labeled tree T_n. The upper part of T_n is a complete balanced tree of depth $n/3$. We label an edge connecting a vertex to its left child with 0 and all other edges with 1. Additionally, we make every leaf whose path to the root contains exactly i zeroes, for $0 \le i \le n/3$, the root of a new complete balanced binary tree of depth i. The edges of these subtrees are labeled analogously. (Fig. 2 shows an example for $n = 9$.) Since there are $\binom{n/3}{i}$ subtrees of depth i, for $0 \le i \le n/3$, the number of leaves in the final tree T_n is

$$\sum_{i=0}^{n/3} \binom{n/3}{i} \cdot 2^i = 3^{n/3},$$

where we have used the binomial theorem. We say that a vertex v in T_n *corresponds* to a string s if the path from the root of T_n to v is labeled s.

Clearly, $\lceil \log(3^{n/3}) \rceil = \lceil (\log 3) n/3 \rceil$ advice bits are sufficient to encode the index of a leaf in T on the advice tape. A reads this index from the tape and computes the corresponding string $s = s_1 s_2 \dots s_{|s|}$, which we shall call *pseudo-advice string,* because A uses s to determine whether to accept a bad vertex or not. We always have $n/3 \le |s| \le 2n/3$, but the length of s varies for different leaves of T. It remains to show that s is always sufficiently long to read pseudo-advice for each bad loner and each bad neighbor, i. e., that $|s| \ge m_1 + m_2$.

We now describe how the oracle chooses the leaf l. To this end, we distinguish two cases. In the following, let m_1' and m_1'' be the number of accepted and rejected bad loners in $\mathcal{O}pt$, respectively.

Case 1: $\mathbf{m_1 \le n/3}$. The oracle sets $s_i := 1$ if and only if the i-th bad loner is in $\mathcal{O}pt$. Conversely, A accepts the i-th bad loner if and only if $s_i = 1$.

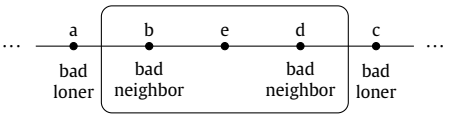

Fig. 3. There are two bad neighbors in this block if the vertices are revealed according to the characters they are labeled with.

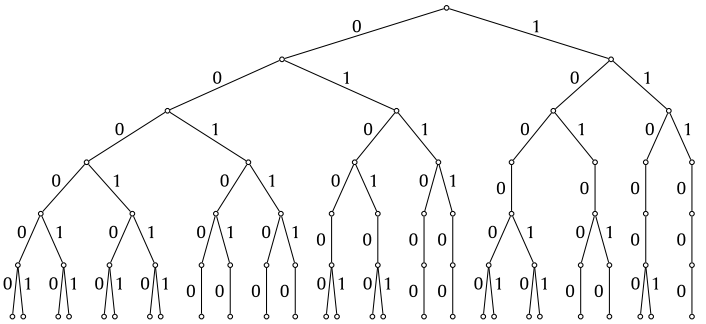

Fig. 4. The tree from the proof of Theorem 10 for the path of size 12. Note the similarity to the tree from Fig. 2.

Since every bad neighbor requires a rejected bad loner, there are at most $m_2 \leq m_1'' \leq n/3 - m_1'$ bad neighbors. Therefore, at most $n/3 - m_1'$ additional pseudo-advice bits are necessary.

Consider the vertex v in T_n corresponding to the string $s_1 s_2 \dots s_{n/3}$. Since this string contains m_1' ones, the subtree rooted at v has, by construction, depth $n/3 - m_1'$. Thus, the oracle can choose the leaf l such that A can reconstruct $s_{n/3+1} s_{n/3+2} \dots s_{m_1+m_2}$ from the path from v to l. A then accepts the i-th bad neighbor if and only if $s_{n/3+i} = 1$.

Case 2: $\mathbf{m_1 > n/3}$. The oracle sets $s_i := 1$ if and only if the i-th bad loner is in $\mathcal{O}pt$. Conversely, A accepts the i-th bad loner if and only if $s_i = 1$. We again consider the vertex v in T_n corresponding to the string $s_1 s_2 \dots s_{n/3}$. This string contains at most m_1' ones, so the subtree rooted at v has depth at least $n/3 - m_1'$.

Therefore, we only need to show that the total number of bad vertices $m_1' + m_1'' + m_2$ is at most $2n/3 - m_1'$ or, equivalently, that $m_2 \leq 2n/3 - 2m_1' - m_1''$.

To this end, we partition the path into blocks $(v_{3j+1}, v_{3j+2}, v_{3j+3})$, for $j \in \{0, 1, \dots, n/3 - 1\}$. At most two vertices per block can be bad neighbors (see Fig. 3). Therefore, we have $m_2 \leq 2n/3$. If v_{3j+2} is a bad loner, it is accepted. (Recall that $\mathcal{O}pt$ contains the middle vertex of each block.) There are thus no bad neighbors in this block, i. e., m_2 is reduced by 2. If either v_{3j+1} or v_{3j+3} is a bad loner, then there can be at most one bad neighbor in this block, i. e., m_2 is reduced by 1. Putting it all together, we obtain the desired result $m_2 \leq 2n/3 - 2m_1' - m_1''$.

Thus, also in this case, the oracle can find a leaf l such that A can reconstruct $s_{n/3+1} s_{n/3+2} \dots s_{m_1+m_2}$ from the path from v to l. A accepts the i-th bad neighbor if and only if $s_{m_1+i} = 1$.

Fig. 2 shows two examples on a path of size 9. □

Interestingly, we can also use this technique to establish a matching lower bound.

Theorem 10. *For* DominatingSet *on paths,* $(\log 3)n/3 - \Theta(1) \approx 0.5283n - \Theta(1)$ *advice bits are necessary to be optimal.*

Proof. We consider the path P_n of size $n = 3k$, for some $k \in \mathbb{N}$. In a nutshell, the proof works as follows. We construct a set $\mathcal{I}$ of input instances that are indistinguishable for any online algorithm on the first $2n/3 - 2$ vertices, but all have unique different optimal solutions. This implies that $\log|\mathcal{I}|$ advice bits are necessary to be optimal [34].

To construct the set $\mathcal{I}$, we reconsider the tree T_{n-3} from the proof of Theorem 9 for the path of size $n - 3$. The upper part, i. e., the subtree from depth 0 to depth $n/3 - 1$, is identical to that of T_{n-3}, i. e., a complete balanced tree of depth $n/3 - 1$. The remaining part of T_{n-3}, however, is modified as follows. For $n/3 - 1 \leq d < 2n/3 - 2$, a vertex at depth d has two children if it is in the left subtree of its predecessor at depth $d - (n/3 - 1)$, and one child otherwise. The edge labeling is identical to that of T_{n-3} except that, additionally, all edges connecting a parent and its only child are labeled 0. Fig. 4 shows the tree for the path of size 12. Note that the number of leaves in this tree is equal to that of T_{n-3}, i. e., $3^{(n-3)/3}$.[2]

From this tree, we now construct $\mathcal{I}$. As in the proof of Theorem 9, we say a vertex v *corresponds* to a string s if the path from the root to v is labeled s. For each leaf corresponding to a string s, we fix a unique revelation order on the vertices of P_n (and thus a unique instance) as follows. Let a denote the number of positions i, $1 \leq i \leq (n-3)/3$, such that $s_i = 0$ and $s_{i+(n-3)/3} = 1$. Analogously, let b denote the number of positions i, $1 \leq i \leq (n-3)/3$, such that $s_i = 0$ and $s_{i+(n-3)/3} = 0$.

[2] Intuitively, we fill up the tree T_{n-3} with dummy vertices such that instances corresponding to different leaves in $T(\mathcal{I})$ are indistinguishable for an algorithm.

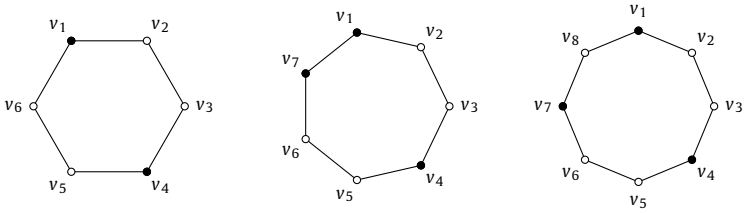

Fig. 5. Cycles of size 6, 7, and 8. The filled vertices indicate $\mathcal{O}pt$ in the proof of Theorem 12.

Finally, let c denote the number of positions i, $1 \le i \le (n-3)/3$, such that $s_i = 1$ (and thus $s_{i+(n-3)/3} = 0$). Clearly, $a + b + c = (n-3)/3$. Consider, moreover, the subsets $A \coloneqq \{ v_{3i} \mid 1 \le i \le a \}$, $B \coloneqq \{ v_{3i} \mid a < i \le a + b \}$, $C \coloneqq \{ v_{n+2-3i} \mid 1 \le i \le c \}$ of $V(P_n)$.

For $1 \le i \le (n-3)/3$, the i-th revealed vertex is

(i) the leftmost previously unrevealed vertex in A if $s_i = 0$ and $s_{i+(n-3)/3} = 1$,
(ii) the leftmost previously unrevealed vertex in B if $s_i = 0$ and $s_{i+(n-3)/3} = 0$, and
(iii) the leftmost previously unrevealed vertex in C if $s_i = 1$.

For $(n-3)/3 < i \le 2(n-3)/3$, the i-th revealed vertex v is

(i) the right neighbor of the vertex revealed in time step $i - (n-3)/3$, if $s_i = 0$, and
(ii) the left neighbor of the vertex revealed in time step $i - (n-3)/3$, if $s_i = 1$.

The remaining vertices are simply revealed from left to right. The instances described in this way form the set $\mathcal{I}$.

All that needs to be done is to verify that any two instances from $\mathcal{I}$ (a) are indistinguishable on the first $2n/3-2$ vertices and (b) all have different unique optimal solutions. Condition (a) is obviously true: By construction, first $n/3-1$ isolated vertices $v_1, v_2, \ldots, v_{n/3-1}$ are revealed, and then some neighbor of $v_1, v_2, \ldots, v_{n/3-1}$ is revealed (in this order) such that the resulting graph consists of components of size 2.[3] Condition (b) follows from the tree we constructed: Consider some instance I and its corresponding leaf l and string s. Then, the optimal solution for I is to accept the i-th revealed vertex if and only if $s_i = 1$, for $1 \le i \le 2n/3 - 2$. □

Similar techniques can be used to obtain the following bounds for cycles. Again, we label the vertices $v_1, \ldots, v_n$, and the edges are $\{v_i, v_{(i \bmod n)+1}\}$, for $1 \le i \le n$. Recall that G denotes the greedy algorithm.

Theorem 11. *For* DominatingSet *on cycles,* G *is strictly* 1.5*-competitive.*

Proof. The proof is analogous to the one of Theorem 7. The only difference is that the greedy algorithm accepts at most $\lfloor n/2 \rfloor$ vertices, because a cycle contains at most that many isolated vertices. Thus, the competitive ratio is 1.5 for every n. □

Theorem 12. *For* DominatingSet *on cycles,* $(\log 3)n/3 + \Theta(\log n) \approx 0.5283n + \Theta(\log n)$ *advice bits are sufficient and* $(\log 3)n/3 - \Theta(1) \approx 0.5283n - \Theta(1)$ *advice bits are necessary to be optimal.*

Proof. The proof of the upper bound is similar to the one of Theorem 9.

If $n \equiv 0 \pmod 3$, the vertices of some fixed optimal solution are evenly distributed, i. e., it contains exactly every third vertex. We only consider the solution $\mathcal{O}pt \coloneqq \{v_1, v_4, \ldots, v_{3\lfloor n/3 \rfloor - 2}\}$. If $n \not\equiv 0 \pmod 3$, $\mathcal{O}pt$ additionally contains $v_{3\lfloor n/3 \rfloor + 1}$. (See Fig. 5 for examples with $n \in \{6, 7, 8\}$.)

Let

$$V' \coloneqq \begin{cases} \{v_1, v_2, v_n\} & \text{if } n \equiv 0 \pmod 3, \\ \{v_1, v_2, v_{n-1}, v_n\} & \text{if } n \equiv 1 \pmod 3, \\ \{v_1, v_2\} & \text{if } n \equiv 2 \pmod 3. \end{cases}$$

An algorithm A first reads the number n from the advice tape as well as the revelation indices of all vertices in V'. This needs $\Theta(\log n)$ advice bits. Note that the graph $G[V \setminus V']$ is simply a path of size $3\lfloor (n-2)/3 \rfloor$. For this path, we can use the technique from the proof of Theorem 9. This uses $(\log 3)\lfloor (n-2)/3 \rfloor$ additional advice bits.

[3] The components do not "touch" each other.

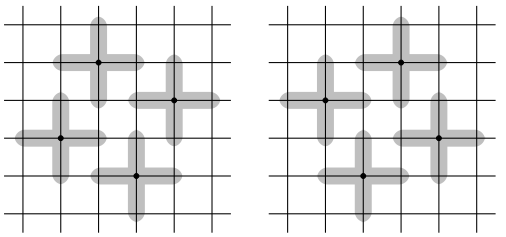

Fig. 6. All optimal solutions on the torus consist of repeated patterns of one of these types. The gray area indicates the vertices dominated by the respective center vertex, i.e., its closed neighborhood.

The proof of the lower bound is completely analogous to the one in Theorem 10, except that, at the very end, three additional vertices are inserted that connect v_1 to v_n. (These three additional vertices can be hidden in the additive constant $-\Theta(1)$.) □

6. Tori

In a sense, paths are a "one-dimensional" graph class. Extending them to two dimensions by considering the cross product of two paths results in the class of grids. Grid graphs arise in various contexts, e.g., in automated circuit design, as idealized city street networks or to model string editing problems [25,29,32].

Investigating dominating sets on grids, however, is tedious and requires a case-by-case analysis since the dominating sets are often distributed very irregularly (see, e.g., [1]). This is because a grid has boundaries, and the top/left vertices are not directly connected to the bottom/right vertices or, in other words, the graph is not vertex-symmetric. One can avoid this problem by simply removing the boundaries or, more precisely, by connecting vertices at opposite boundaries of the grid; this results in a torus.

We consider the torus $T_{m,m}$ of size $m \times m$, for $m \in \mathbb{N}^+$, i.e., the graph (V, E) with $V = \{ v_{i,j} \mid 1 \leq i, j \leq m \}$ and

$$E = \{ \{v_{i,j}, v_{i,(j \bmod m)+1}\}, \{v_{i,j}, v_{(i \bmod m)+1,j}\} \mid 1 \leq i, j \leq m \}.$$

If m is a multiple of 5, there are exactly ten optimal solutions, which all consist of repetitions of the patterns shown in Fig. 6, i.e., in every row and column, every fifth vertex belongs to an optimal solution. The ten solutions result from two degrees of freedom. First, we can mirror the pattern and second, we can choose the horizontal offset with respect to $v_{1,1}$ (from 0 to 4). The solutions are optimal because the closed neighborhoods of the vertices are disjoint. It can be easily seen that there are no other possibilities of covering the torus with non-overlapping closed neighborhoods of single vertices; thus, these ten solutions are indeed the only optimal ones.

For the next results, we will use the bounds

$$\frac{2^{nH(k/n)}}{n+1} \leq \binom{n}{k} \leq 2^{nH(k/n)}, \tag{3}$$

where H denotes the binary entropy function, i.e., $H(0) = H(1) := 0$ and $H(\alpha) = -\alpha \log \alpha - (1-\alpha)\log(1-\alpha)$, for $0 < \alpha < 1$.

Theorem 13. *For* DominatingSet *on the $(m \times m)$-torus, $0.8H(0.25)n + \Theta(\log n) \approx 0.6490n + \Theta(\log n)$ advice bits are sufficient to be optimal, where $n = m^2$ denotes the number of vertices.*

Proof. We use a similar terminology as in the proof of Theorem 9. We consider any fixed optimal solution and try to rebuild this solution. Note that, for any size of the torus, there exists an optimal solution that does not contain any two adjacent vertices. We consider any fixed such optimal solution and try to rebuild this solution. If a newly revealed vertex v has 4 neighbors that were not taken into the dominating set, it is clear that it has to be accepted. Otherwise, some neighbor of v is revealed after v and thus will not be accepted because it is adjacent to v. Thus, the algorithm needs advice for at most four vertices in the closed neighborhood of an accepted vertex v; we call these vertices *bad*.

Therefore, we can encode the optimal solution as a bit string of length $0.8n$ with at most one fourth, i.e., $0.2n$, ones. There are $\sum_{i=0}^{0.2n} \binom{0.8n}{i}$ such bit strings, and thus

$$\log\left(\sum_{i=0}^{0.2n} \binom{0.8n}{i}\right) \leq \log\left(0.2n \binom{0.8n}{0.2n}\right) = \log 0.2 + \log n + \log \binom{0.8n}{0.2n}$$
$$\overset{(3)}{\leq} \log 0.2 + \log n + 0.8nH(0.25) \approx 0.6491n + \Theta(\log n)$$

advice bits suffice to be optimal. □

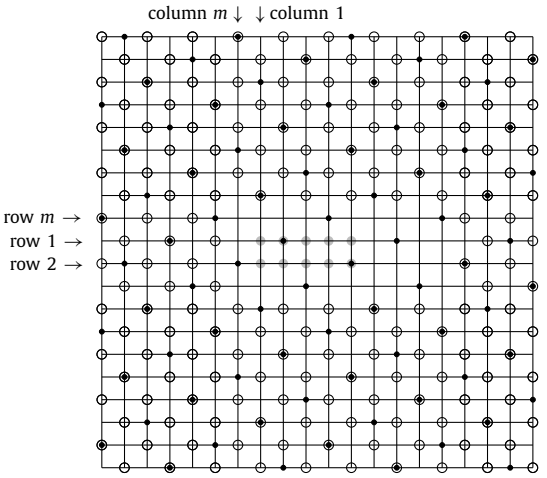

Fig. 7. The torus $T_{20,20}$. The gray vertices indicate the start vertices. (The indices increase (modulo m) from top to bottom and from left to right.) The circled vertices are those the adversary reveals in the first phase. The black dotted vertices indicate the optimal solution.

Theorem 14. *For* DominatingSet *on the* $(m \times m)$*-torus,* $0.5\,H(0.2)\,n - \Theta(\log n) \approx 0.3610n - \Theta(\log n)$ *advice bits are necessary to be optimal, where* $n = m^2$ *is the number of vertices.*

Proof. We restrict ourselves to the torus $T_{m,m}$, with $m = 20k$, for some $k \in \mathbb{N}^+$.

First, the adversary reveals the *start vertices* $v_{i,j}$, for $1 \le i \le 2$ and $1 \le j \le 5$, in some fixed order. Two of them must be accepted, otherwise the solution is not optimal. The choice of the accepted vertices makes the optimal solution unique. We denote it by $\mathcal{O}pt$. Recall that there are exactly ten optimal dominating sets on this torus and the two degrees of freedom according to which an optimal solution can be chosen are used up by fixing the two start vertices.

The adversary reveals the vertices from the set

$$A := (\{\, v_{2i-1,2j-1} \mid 1 \le i, j \le m/2 \,\} \cup \{\, v_{2i,2j} \mid 1 \le i, j \le m/2 \,\}) \setminus \{\, v_{i,j} \mid i \in \{1,2,3,m\},\, j \in \{1,\dots,9,m\} \,\}.$$

See Fig. 7 for an example with $k = 1$.

This set contains exactly $n/2 - 20$ vertices. Moreover, one easily verifies that exactly one fifth, i. e., $n/10 - 4$, of all these vertices are in $\mathcal{O}pt$.

Recall that the choice of the two accepted vertices among the start vertices makes $\mathcal{O}pt$ unique. Moreover, every possible input instance looks identical to the algorithm until the first $10 + n/2 - 20 = n/2 - 10$ vertices have been revealed, because no two vertices from A are connected to each other nor is any of them connected to a start vertex.

We can model these instances as bit strings of length $n/2 - 20$ with $n/10 - 4$ ones, and any two different such bit strings require a different output to be optimal. There are $\binom{n/2-20}{n/10-4}$ of them, thus the necessary number of advice bits is

$$\begin{aligned}\left\lceil \log \binom{n/2-20}{n/10-4} \right\rceil &\overset{(3)}{\ge} \lceil (n/2-20)H(0.2) - \log(n/2-19) \rceil \\ &= 0.5\,H(0.2)\,n - \Theta(\log n) \\ &\approx 0.3610n - \Theta(\log n). \qquad \square\end{aligned}$$

7. Maximal outerplanar graphs

An important family of graphs are planar graphs. For general planar graphs, one easily sees that the lower bound from Theorem 1 holds, since the graph constructed in the proof is planar. In fact, the graph is even outerplanar, and the lower bound also holds for this more restricted graph class. Therefore, we have to restrict ourselves even further to obtain meaningful new results. To this end, we consider the family of maximal outerplanar graphs. For this class, the lower bound of Theorem 1 does not apply anymore; indeed, we provide a better upper bound for optimality.

Formally, the graph class we are interested in is defined as follows.

Definition 6. A graph $G = (V, E)$ is *maximal outerplanar* if there is a planar embedding of G such that all vertices are adjacent to the unbounded face and there is no such planar embedding for the graph $(V, E \cup \{e\})$, for any edge $e \notin E$.

We present both an upper and a lower bound on the advice complexity.

Theorem 15. *For* DominatingSet *on maximal outerplanar graphs,* $H(1/3)n + \Theta(\log n) \approx 0.9183n + \Theta(\log n)$ *advice bits are sufficient.*

Proof. Observe that every maximal outerplanar graph contains a cycle C with all vertices, namely the one whose edges are all adjacent to the unbounded face. Clearly, any optimal solution $\mathcal{O}pt$ contains at most every third vertex from C, i.e., at most $\lceil n/3 \rceil$ vertices. Let $v_1, v_2, \ldots, v_n$ be the vertices of C in revelation order. We can represent $\mathcal{O}pt$ by a bit string $s = s_1 s_2 \ldots s_n$ of length n such that $s_i = 1$ if and only if $v_i \in \mathcal{O}pt$, for $1 \leq i \leq n$.

First, A reads $\mathrm{Log}n$ advice bits to learn n and then another $\lceil \log n \rceil$ advice bits to learn the time step t in which the first vertex is revealed that should be accepted. Consider now the bit string $s' := s_1 s_2 \ldots s_{t-1} s_{t+1} s_{t+2} \ldots s_n$ of length $n-1$. Since s contains at most $\lceil n/3 \rceil$ ones, s' contains at most $\lfloor n/3 \rfloor$ ones. The oracle now simply writes the index of s' with respect to the canonical order over all possible bit strings of length $n-1$ with at most $\lfloor n/3 \rfloor$ ones. There are

$$\sum_{i=0}^{\lfloor n/3 \rfloor} \binom{n-1}{i}$$

such bit strings, and thus

$$\begin{aligned}
\left\lceil \log\left(\sum_{i=0}^{\lfloor n/3 \rfloor} \binom{n-1}{i} \right) \right\rceil &= \left\lceil \log\left(\binom{n-1}{0} + \sum_{i=1}^{\lfloor n/3 \rfloor} \binom{n-1}{i} \right) \right\rceil \\
&= \left\lceil \log\left(1 + \sum_{i=1}^{\lfloor n/3 \rfloor} \binom{n-1}{i} \right) \right\rceil \\
&\leq \left\lceil 1 + \log\left(\frac{n}{3} \binom{n}{\lfloor n/3 \rfloor} \right) \right\rceil \\
&= \left\lceil \log n + 1 - \log 3 + \log \binom{n}{\lfloor n/3 \rfloor} \right\rceil \\
&\overset{(3)}{\leq} \left\lceil \log n + 1 - \log 3 + n\, H\left(\frac{\lfloor n/3 \rfloor}{n} \right) \right\rceil \\
&\leq \lceil \log n + 1 - \log 3 + n\, H(1/3) \rceil
\end{aligned}$$

advice bits are sufficient to encode s'. Since A already knows t, it can compute s and accepts vertex v_i if and only if $s_i = 1$, for $1 \leq i \leq n$. □

Theorem 16. *For* DominatingSet *on maximal outerplanar graphs,* $0.5n - \Theta(1)$ *advice bits are necessary to be optimal.*

Proof. Consider the graph $G_k = (V, E)$, for $k \in \mathbb{N}^+$, with $V = \{v_1, \ldots, v_{4k}, v\}$ and

$$\begin{aligned}
E = &\{ \{v_i, v_{i+1}\} \mid 1 \leq i < 4k \} \\
&\cup \{ \{v_{4i-3}, v_{4i-1}\} \mid 1 \leq i \leq k \} \\
&\cup \{ \{v_{4i-3}, v\}, \{v_{4i-1}, v\}, \{v_{4i}, v\} \mid 1 \leq i \leq k \}.
\end{aligned}$$

Fig. 8 shows the graph G_6.

We first show that the unique optimal solution is $\mathcal{O}pt = \{ v_{4i-1} \mid 1 \leq i \leq k \}$. To this end, observe that any optimal solution must contain *at least* one of the vertices $v_{4i-3}, v_{4i-2}, v_{4i-1}$, for $1 \leq i \leq k$, as otherwise v_{4i-2} is not dominated, and it must contain *at most* one of these vertices, as otherwise the optimal solution is bigger than $\mathcal{O}pt$. Hence, no optimal solution contains v or any v_{4i}. Because of this, any optimal solution has to contain v_{4k-1}, as otherwise v_{4k} is not dominated. This argument can be applied inductively for every v_{4i-1} and v_{4i}, for $1 \leq i < k$.

The adversary chooses a string s of length $k-1$ over the alphabet $\{0, 1, 2, 3\}$.

Consider the sets

$$\begin{aligned}
S_0 &:= \{ (v_{4i}, v_{4i+1}, v_{4i+2}) \mid |s|_1 + |s|_3 < i \leq |s|_0 + |s|_1 + |s|_3 \}, \\
S_1 &:= \{ (v_{4i-1}, v_{4i}, v_{4i+1}) \mid 1 \leq i \leq |s|_1 \}, \\
S_2 &:= \{ (v_{4i-2}, v_{4i-1}, v_{4i}) \mid |s|_0 + |s|_1 + |s|_3 + 1 < i \leq k \}, \\
S_3 &:= \{ (v_{4i-1}, v_{4i}, v_{4i+1}) \mid |s_1| < i \leq |s|_1 + |s|_3 \}.
\end{aligned}$$

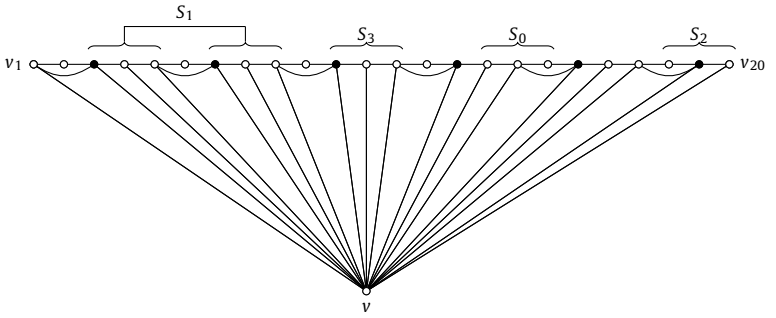

Fig. 8. The graph G_6 from the proof of Theorem 16 for the string 01123. The filled vertices indicate the unique optimal solution.

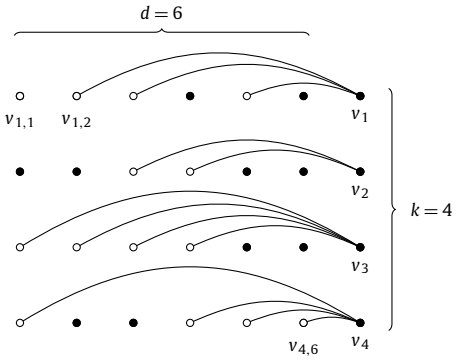

Fig. 9. One possible graph from the proof of Theorem 17, for $d = 6$ and $k = 4$. The filled vertices indicate the unique optimal solution.

For $1 \leq i < k$, the adversary inspects the i-th digit s_i of s and proceeds as follows. If $s_i = j$, for $j \in \{0, 1, 2, 3\}$, it selects some not yet selected triple from the set S_j and reveals its three vertices either from left to right or from right to left such that the j-th revealed vertex is in $\mathcal{O}pt$. (For $j = 0$, no vertex from the triple is in $\mathcal{O}pt$.)

Afterwards, it reveals all remaining vertices in an arbitrary order.

After the first $3(k-1)$ vertices have been revealed, all instances are still indistinguishable for the algorithm, since no two of the triples are connected to each other and in no triple, the left and the right vertex are connected, i. e., there are no induced triangles. Since the optimal solution is unique, any two different strings s require a different behavior to be optimal. There are 4^{k-1} such strings, and thus at least

$$\lceil \log(4^{k-1}) \rceil = \lceil 2(k-1) \rceil = \left\lceil 2 \cdot \frac{n-5}{4} \right\rceil = \lceil 0.5n - 2.5 \rceil$$

advice bits are necessary. □

8. Bounded-degree graphs

In this last section, we consider graphs whose vertices have maximum degree d.

Theorem 17. *For* DominatingSet *on graphs of size n with maximum degree d, at least $(d-1)n/(d+1)$ advice bits are necessary to be optimal.*

Proof. Consider the following set of input instances of size $k(d+1)$, for $k \in \mathbb{N}$. The adversary first reveals dk isolated vertices $v_{1,1}, v_{1,2}, \ldots, v_{1,d}, v_{2,1}, \ldots, v_{k,d}$. Afterwards, it reveals k vertices $v_1, \ldots, v_k$ such that v_i is connected to a subset $V_i \subseteq \{v_{i,1}, \ldots, v_{i,d}\}$ of size at least 2, for $1 \leq i \leq k$. (See Fig. 9 for an example.)

For every i, there are $2^d - (d+1)$ different subsets V_i and thus $(2^d - (d+1))^k$ different ways to select all subsets. All these instances look identical after the first dk vertices have been revealed. Nevertheless, each instance requires a different behavior on these vertices to be optimal. Thus, $(2^d - (d+1))^k$ different deterministic algorithms are necessary. We have

$$\begin{aligned} \lceil \log((2^d - (d+1))^k) \rceil &= \lceil k \log(2^d - (d+1)) \rceil \\ &\geq \lceil k \log(2^d - 2^{d-1}) \rceil \end{aligned}$$

$$= \lceil k \log(2^{d-1}) \rceil$$
$$= k(d-1),$$

where the inequality holds for all $d \geq 4$. Thus, at least $k(d-1) = (d-1)n/(d+1)$ advice bits are necessary to be optimal. □

9. Conclusion

We have considered an online version of the well-known dominating set problem which was only known to admit only a linear competitive ratio for deterministic online algorithms. Using the model of advice complexity, we generalized this result to an almost linear lower bound on the competitive ratio of any randomized online algorithm. Moreover, we investigated the advice complexity for computing an optimal solution for several restricted classes of input graphs and proved linear upper and lower bounds for them. While these bounds are almost tight (up to an additive logarithmic term) for paths and cycles and for general graphs, it remains as an open problem to close the gap for tori and maximal outerplanar graphs. It is also an interesting open problem to generalize these results by determining the advice complexity of computing a near-optimal solution.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The authors would like to thank Elisabet Burjons, Fabian Frei, and Dennis Komm for helpful discussions, and an anonymous reviewer for some comments which helped to significantly improve the section on randomized algorithms.

References

[1] S. Alanko, S. Crevals, A. Isopoussu, P. Östergård, V. Pettersson, Computing the domination number of grid graphs, Electron. J. Comb. 18 (2011).
[2] K. Barhum, H.-J. Böckenhauer, M. Forišek, H. Gebauer, J. Hromkovič, S. Krug, J. Smula, B. Steffen, On the power of advice and randomization for the disjoint path allocation problem, in: SOFSEM 2014, in: LNCS, vol. 8327, Springer, 2014, pp. 89–101.
[3] H.-J. Böckenhauer, N. Corvelo Benz, D. Komm, Call admission problems on trees with advice, in: IWOCA 2019, in: LNCS, vol. 11638, Springer, 2019, pp. 108–121.
[4] M.P. Bianchi, H.-J. Böckenhauer, J. Hromkovič, L. Keller, Online coloring of bipartite graphs with and without advice, in: COCOON 2012, in: LNCS, vol. 7434, Springer, 2012, pp. 519–530.
[5] H.-J. Böckenhauer, J. Hromkovič, D. Komm, S. Krug, J. Smula, A. Sprock, The string guessing problem as a method to prove lower bounds on the advice complexity, in: COCOON 2013, in: LNCS, vol. 7936, Springer, 2013, pp. 493–504.
[6] H.-J. Böckenhauer, J. Hromkovič, D. Komm, S. Krug, J. Smula, A. Sprock, The string guessing problem as a method to prove lower bounds on the advice complexity, Theor. Comput. Sci. 554 (2014) 95–108.
[7] H.-J. Böckenhauer, D. Komm, R. Královič, R. Královič, On the advice complexity of the k-server problem, in: ICALP 2011, in: LNCS, vol. 6755, Springer, 2011, pp. 207–218.
[8] H.-J. Böckenhauer, D. Komm, R. Královič, R. Královič, T. Mömke, On the advice complexity of online problems, in: ISAAC 2009, in: LNCS, vol. 5878, Springer, 2009, pp. 331–340.
[9] H.-J. Böckenhauer, D. Komm, R. Královič, R. Královič, T. Mömke, Online algorithms with advice: the tape model, Inf. Comput. 254 (2017) 59–83.
[10] H.-J. Böckenhauer, D. Komm, R. Královič, P. Rossmanith, On the advice complexity of the knapsack problem, Theor. Comput. Sci. 527 (2014) 61–72.
[11] H.-J. Böckenhauer, D. Komm, R. Wegner, Call admission problems on grids with advice, in: WAOA 2018, in: LNCS, vol. 11312, Springer, 2018, pp. 118–133.
[12] A. Borodin, R. El-Yaniv, Online Computation and Competitive Analysis, Cambridge University Press, 1998.
[13] J. Boyar, S.J. Eidenbenz, L.M. Favrholdt, M. Kotrbcík, K.S. Larsen, Online dominating set, Algorithmica 81 (5) (2019) 1938–1964.
[14] J. Boyar, L.M. Favrholdt, C. Kudahl, K.S. Larsen, J.W. Mikkelsen, Online algorithms with advice: a survey, ACM Comput. Surv. 50 (2) (2017) 19.
[15] J. Boyar, L.M. Favrholdt, C. Kudahl, J.W. Mikkelsen, The advice complexity of a class of hard online problems, Theory Comput. Syst. 61 (4) (2017) 1128–1177.
[16] J. Boyar, L.M. Favrholdt, C. Kudahl, J.W. Mikkelsen, Weighted online problems with advice, Theory Comput. Syst. 62 (6) (2018) 1443–1469.
[17] J. Boyar, S. Kamali, K.S. Larsen, A. López-Ortiz, Online bin packing with advice, Algorithmica 74 (1) (2016) 507–527.
[18] S. Dobrev, R. Královič, D. Pardubská, How much information about the future is needed?, in: SOFSEM 2008, in: LNCS, vol. 4910, Springer, 2008, pp. 247–258.
[19] R. Dorrigiv, M. He, N. Zeh, On the advice complexity of buffer management, in: ISAAC 2012, in: LNCS, vol. 7676, Springer, 2012, pp. 136–145.
[20] S. Eidenbenz, Online dominating set and variations on restricted graph classes, Technical Report 380, ETH Zürich, 2002.
[21] P. Elias, Universal codeword sets and representations of the integers, IEEE Trans. Inf. Theory 21 (2) (1975) 194–203.
[22] Y. Emek, P. Fraigniaud, A. Korman, A. Rosén, Online computation with advice, Theor. Comput. Sci. 412 (24) (2011) 2642–2656.
[23] S. Gupta, S. Kamali, A. López-Ortiz, On advice complexity of the k-server problem under sparse metrics, in: SIROCCO 2013, in: LNCS, vol. 8179, Springer, 2013, pp. 55–67.
[24] T.W. Haynes, S. Hedetniemi, P. Slater, Fundamentals of Domination in Graphs, Chapman & Hall/CRC, 1998.
[25] F. Hoffman, F.O. Hadlock, Block patterns and routing, in: Proc. of the Sixth Southeastern Conference on Combinatorics, Graph Theory and Computing, 1975.
[26] J. Hromkovič, R. Královič, R. Královič, Information complexity of online problems, in: MFCS 2010, in: LNCS, vol. 6281, Springer, 2010, pp. 24–36.
[27] G.-H. King, W.-G. Tzeng, On-line algorithms for the dominating set problem, Inf. Process. Lett. 61 (1997) 11–14.
[28] D. Komm, An Introduction to Online Computation, Springer, 2016.

[29] C.W. Marshall, Applied Graph Theory, Wiley, 1971.
[30] Randomization can be as helpful as a glimpse of the future in online computation, in: ICALP 2016, in: LIPIcs, vol. 55, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, 39.
[31] M.P. Renault, A. Rosén, On online algorithms with advice for the k-server problem, in: WAOA 2011, in: LNCS, vol. 7164, Springer, 2011, pp. 198–210.
[32] J.P. Schmidt, All highest scoring paths in weighted grid graphs and their applications to finding all approximate repeats in strings, SIAM J. Comput. 27 (4) (1998) 972–992.
[33] S. Seibert, A. Sprock, W. Unger, Advice complexity of the online coloring problem, in: CIAC 2013, in: LNCS, vol. 7878, Springer, 2013, pp. 345–357.
[34] B. Steffen, Advice Complexity of Online Graph Problems, PhD thesis, ETH Zurich, 2014.