On the Advice Complexity of Online Edge- and Node-Deletion Problems

Peter Rossmanith(✉)

Department of Theoretical Computer Science,
RWTH Aachen University, Aachen, Germany
rossmani@informatik.rwth-aachen.de

Abstract. We consider a model of online graph modification problems where the input graph is read piecewise in an adversarial order and the algorithm has to modify the graph by deleting vertices or edges in order to keep it inside some fixed graph class Π. These deletions cannot be taken back later. We analyze the least number of advice bits that enable an online algorithm to perform the same number of deletions as an optimal offline algorithm. We consider only hereditary properties Π, for which optimal online algorithms exist and which can be characterized by a set of forbidden subgraphs $\mathcal{F}$. It is clear that, if $\mathcal{F}$ is finite, then the number of required advice bits is at most linear in the size of an optimal solution. We prove lower and upper bounds based on the structure of the graphs in $\mathcal{F}$, often determining the complexity exactly or nearly exactly. The techniques also work for infinite $\mathcal{F}$ and we can determine whether then the advice complexity can be bounded by a function in the optimal solution or not. For node-deletion problems we characterize the advice complexity exactly for all cases and for edge-deletion problems at least for the case of a single forbidden induced subgraph.

1 Introduction

Online algorithms do not see their whole input at once, but receive it in pieces and have to react each time they receive a new one. This is a very natural setting for situations where the future is unknown. A typical example are strategies for caches: If the cache is full and a page fault occurs, a page has to be discarded without knowing which pages will be accessed in the (near) future. One way to analyze online optimization problems is to investigate their *competitive ratio*, which tells us how much worse an online algorithm performs relative to an offline algorithm that has access to the whole input from the beginning [3].

A different way to analyze online problems is to look at their *advice complexity*. In this model the algorithm receives an advice in the form of a bit string from an all knowing oracle helping the algorithm to compute an optimal or approximate solution with a given competitive ratio [5]. Lower bounds on the advice complexity are very strong as they tell us that an online problem cannot be solved well even if we know something about the future.

H.-J. Böckenhauer et al. (Eds.): Hromkovič Festschrift, LNCS 11011, pp. 449–462, 2018.
https://doi.org/10.1007/978-3-319-98355-4_26

Online Graph Modification Problems

In an online graph problem a graph G is revealed as a sequence of growing induced subgraphs $G[v_1], G[v_1, v_2], \ldots, G[v_1, \ldots, v_n]$. Whenever a new vertex or edge is presented, the algorithm has to make a decision about the vertex or edge that cannot be taken back later. For example, if the problem consists of finding a minimum vertex cover, it has to be decided for every new vertex whether it is in- or excluded from the vertex cover. There are several results known about the advice complexity of such online graph problems. For example, Bianchi, Böckenhauer, Hromkovič, Krug, and Steffen show that linear advice is necessary for optimal or nearly-optimal $L(2,1)$-coloring on paths [2] and Steffen's whole PhD thesis is dedicated among others to various online graph problems in this setting [7]. He also considers the online vertex cover problem that can also be seen as a graph modification problem: Finding a vertex cover is equivalent to deleting vertices in order to get an empty graph. Komm et al. [6] study the advice complexity of a class of graph problems where a largest (resp. smallest) induced subgraph of the input graph with a certain property has to be chosen in an online fashion.

In this paper we consider a different model for online graph modification problems: For a Π-modification problem the graph is again presented as a sequence of growing induced subgraphs $G_1, G_2, \ldots$, but the algorithm does not have to do anything as long as $G_i \in \Pi$. If, however, $G_i \notin \Pi$, the algorithm has to perform modifications on G_i in order to make it a member of Π.

In the offline world graph modification problems are well studied. Already a long time ago Yannakakis proved that node-deletion problems are NP-hard for every non-trivial hereditary graph property and that many edge-deletion problems are NP-hard, too [8]. Cai analyzed the parameterized complexity of graph modification problems [4]. All variants are fixed-parameter tractable with respect to the solution size if the graph property can be characterized by a finite set of forbidden induced subgraphs. In this paper we are not concentrating on the running time, but as we will be considering advice given by an oracle it has to be noted that the oracle will usually be solving NP-hard problems when preparing the advice string while the online algorithm itself usually performs only simple calculations.

In the special case of node- or edge-deletion problems we can more specifically think of maintaining a set D of deleted nodes or edges. Whenever $G_i[V-D] \notin \Pi$ (resp. $G_i[E-D] \notin \Pi$), nodes or edges have to be inserted into D such that then $G_i[V-D] \in \Pi$ (resp. $G_i[E-D] \in \Pi$). Remember that this model is less restricted than the one in [2] because nodes or edges that have been presented in the past can still be deleted. It is nevertheless an online problem because decisions cannot be taken back and have to be made without knowing the future. The motivation for this model is that often problems have to be solved as soon as they arise, but not sooner.

Let us look at a simple introductory example: Cluster deletion. A cluster graph is a collection of disjoint cliques. Given a graph G the cluster deletion problem asks for a minimum set D of edges whose deletion turns G into a cluster

graph. In our model we receive the graph G piecewise vertex by vertex. Each time we receive a new vertex that turns the graph into a non-cluster graph, we have to insert edges into D in order to ensure that $G_i[E(G_i) - D]$ is a cluster graph. It is clear that in the worst case we have no chance to compute an optimal D in this way. If we denote the size of an optimal solution by *opt*, it turns out that we can find an optimal solution of the same size online if we are given *opt* advice bits: Whenever we find an induced P_2 in our graph we have to delete at least one of its edges. We can read one advice bit to find out which one is the right one. As a graph is a cluster graph iff it does not contain P_2 as an induced subgraph the algorithm is correct.

It is also easy to see that this simple algorithm is optimal: An adversary can present k times a P_2 which is in the next step expanded into a P_3 on either side. To be optimal the algorithm has to choose the correct edge to delete each time of the k times. This makes 2^k possibilities and the algorithm cannot act identically for any pair of these possibilities. Hence, the algorithm needs at least k advice bits.

In this paper we consider similar problems and find ways to compute their exact advice complexity.

2 Preliminaries

We will use the usual notation for graphs, which will always be simple, undirected, and loop-free. We write $H \subseteq G$ if H is a subgraph of G. The size of a graph is its number of vertices and denoted by $|G|$. The number of edges is denoted by $||G||$. An edge between nodes x and y is called xy. If $\mathcal{F}$ is a set of graphs and none of them is an induced subgraph of G we say that G is $\mathcal{F}$-free. If X is a set of edges or vertices we denote the subgraph of G induced by X by $G[X]$. Sometimes we will just write $G - u$ to denote the graph we get from G by deleting the vertex u. A path and circle with k edges are denoted by P_k and C_k, respectively. We will use extensively a nonstandard operation where we glue two graphs together along an edge:

Definition 1. *Let G and H be graphs and $xy \in E(G)$, $uv \in E(H)$. We define two operations that glue G and H together along their edges xy and uv. First, $G \,{}_{xy}\oplus_{uv}\, H$ is the graph that we get by identifying u with x and v with y (and replacing the double edge by a single one). If G and H are not vertex-disjoint we replace them by disjoint copies first. We say that $G \,{}_{xy}\oplus_{uv}\, H$ consists of two parts, one is the induced subgraph by $V(G)$ and the other by $V(H)$. The two parts overlap in x and y. Second, $G_{xy}\ominus_{uv}$ is the same except that the edges xy and uv are removed completely.*

We abbreviate $G \oplus_{xy} G := G \,{}_{xy}\oplus_{xy}\, G$ and $G \ominus_{xy} G := G \,{}_{xy}\ominus_{xy}\, G$. Figure 1 shows an example.

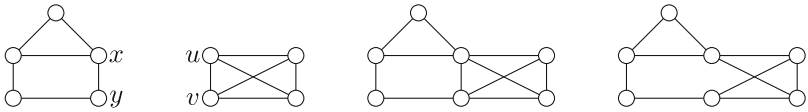

Fig. 1. From left to right: G, H, $G\ {}_{xy}\oplus_{uv} H$, $G\ {}_{xy}\ominus_{uv} H$

3 Some General Results

If we are facing a Π-graph modification problem for a graph class Π there are special cases we can consider for Π. If we know nothing about Π we can still show that $opt \log n$ advice bits are sufficient to solve the Π-edge-deletion problem optimally on a graph with n nodes.

Theorem 1. *Let Π be a hereditary graph property.*

(1) The Π-node-deletion problem can be solved optimally with $\lceil opt \log n \rceil$ advice bits.
(2) The Π-edge-deletion problem can be solved optimally with at most $2opt \log n$ advice bits.

Proof

(1) Whenever the algorithm detects that the graph is not in Π, a node has to be deleted. There are at most n nodes to choose so the correct one can be encoded with $\log n$ bits and there are at most n^{opt} possibilities to choose opt nodes. Such a number can be encoded with $\lceil opt \log n \rceil$ bits.
In that way an optimal set of vertices is deleted. As Π is hereditary, all induced subgraphs seen by the algorithm in-between also belong to Π if the same optimal solution is deleted from them.
(2) If the algorithm detects that the graph is not in Π, one or more edges have to be deleted. There are at most $n^2/2$ edges in total to choose from. There are only $(n^2/2)^{opt}$ possibilities to choose a set of opt edges. We need only $\lceil \log((n^2/2)^{opt}) \rceil \leq 2opt \log n$ bits. □

While Theorem 1 gives us a simple upper bound on the advice complexity, it is often too pessimistic and we can find a better one. On the other hand, it will turn out that there are very hard edge-deletion problems where the bound of Theorem 1 is almost optimal.

One ugly, but sometimes necessary, property of the bound in Theorem 1 is that the number of advice bits can get arbitrarily large even if the size of the optimal solution is bounded by a constant. Let us look at some special cases, where this is not the case and the number of advice bits is bounded by a function of the solution size.

One important case are hereditary properties Π, i.e., properties that are closed under taking induced subgraphs. It is well known that such properties can be characterized by a set of forbidden induced subgraphs. If $\mathcal{F}$ is such a set we can always assume that it does not contain two graphs H_1 and H_2 such that H_1 is an induced subgraph of H_2 because H_2 would be redundant. Under this

assumption $\mathcal{F}$ is determined completely by Π and can be finite or infinite and we say that $\mathcal{F}$ is *unordered*.

Moreover, it is also clear that if a hereditary class Π contains at least one graph then it also contains the null graph with no vertices (because that is an induced subgraph of any graph). There is a vast number of important hereditary graph properties, for example planar graphs, outerplanar graphs, forests, genus-bounded graphs, chordal graph, bipartite graphs, cluster graphs, line graphs, etc.

Hereditary graph properties are *exactly* those properties that can be solved optimally in this model if "optimal" means that the offline solution is not larger than the best online solution that has to modify only one graph G (while the online algorithm has to modify a sequence of induced subgraphs leading to G).

Theorem 2. *The online Π-edge and Π-node-deletion problems can be solved optimally with respect to the smallest offline solution if and only if Π is hereditary, even if arbitrarily many advice bits can be used.*

Proof. Theorem 1 already shows that an optimal solution can be found for hereditary properties.

Let now Π be a graph property that is not hereditary. Then there are graphs G_1 and G_2 such that (1) $G_1 \notin \Pi$, (2) $G_2 \in \Pi$, and (3) G_1 is an induced subgraph of G_2.

The adversary can present first G_1 and later G_2. Any correct algorithm has to delete something from G_1, but the optimal offline solution is to delete nothing. □

Because of Theorem 2 we will look only at hereditary graph properties in this paper. It should be noted, however, that a sensible treatment of non hereditary graph properties is possible if the definition of online optimality is adjusted in the right way.

4 $\mathcal{F}$-Node Deletion Problems

Let us first look at node-deletion problems because they are much easier than edge-deletion problems in the online setting. The general ideas on the lower bounds are the same.

Definition 2. *Let $\mathcal{F}$ be an unordered set of graphs. The online $\mathcal{F}$-node-deletion problem is an online graph modification problem where the algorithm chooses nodes that have to be deleted in order for the input graph to stay $\mathcal{F}$-free. Once a node is chosen for deletion this decision cannot be taken back.*

The following theorem states that the number of necessary advice bits can easily be computed by looking at $\mathcal{F}$. Essentially it depends on the size of the largest graph in $\mathcal{F}$. We will see later that in edge-deletion problems something similar, but more complicated, holds as well. The proofs will also be more involved.

Theorem 3. *Let $\mathcal{F}$ be a set of connected graphs. The online $\mathcal{F}$-node-deletion problem can be solved optimally with $\lceil opt \log s \rceil$ many advice bits, where s is the size of the largest graph in $\mathcal{F}$. This bound is tight if $s < \infty$.*

If $s = \infty$, then no algorithm can be optimal with $f(opt)$ advice bits for any function f.

Proof. Let H be the largest graph in $\mathcal{F}$ if $\mathcal{F}$ is finite and u be an arbitrary node in H. If we glue two disjoint copies of H together at u by identifying the respective nodes, the resulting graph H_u contains H as an induced subgraph and there is exactly one node (i.e., u) that we can delete in order to make H_u H-free. However, when deleting u from H_u the remaining graph becomes disconnected and both components are proper induced subgraphs of H. Hence, the components are $\mathcal{F}$-free and therefore $H_u - u$ is also $\mathcal{F}$-free.

An adversary can simply present first H and then adding nodes to get H_u for $u \in V(H)$. In this way the adversary has $|H|$ possibilities to continue and to be optimal when seeing only H the algorithm has to delete the correct u. By repeating this k times there are $|H|^k$ different possibilities and they have all to be distinguished. Hence we need at least $\log(|H|^k) = k \log(|H|)$ advice bits. It is easy to see that $\lceil k \log s \rceil$ bits are also sufficient if $k = opt$ because the algorithm has to pick the right edge of H when finding $H \in \mathcal{F}$ as an induced subgraph. □

5 $\mathcal{F}$-Edge Deletion Problems

Let $\mathcal{F}$ be a collection of graphs pairwise independent with regard to being induced subgraphs. We try to emulate results for node-deletion problems. For those we glued two graphs together at a node. Now we have to glue graphs together at an edge. A crucial difference between these two ways of glueing is that basically "nothing can go wrong" when glueing graphs at a node and then deleting the node because the graph becomes disconnected and the parts are induced subgraphs of the original graphs. This is no longer true when glueing along edges. The next definition tries to capture the idea of "nothing can go wrong" by labeling edges as *critical* if we can use them without producing a graph that contains graphs from $\mathcal{F}$.

Definition 3. *Let $\mathcal{F}$ be a collection of graphs and $H \in \mathcal{F}$. We classify all edges in H as* critical *or* non-critical. *An edge $xy \in E(H)$ is* critical *iff there is an $H' \in \mathcal{F}$ and an edge $uv \in E(H')$ such that $H \,{}_{xy}\ominus_{uv}\, H'$ does not contain any graph from $\mathcal{F}$ as an induced subgraph.*

Let $\#crit_{\mathcal{F}}(H)$ denote the number of critical edges in H and $\#crit(\mathcal{F}) = \max\{\, \#crit_{\mathcal{F}}(H) \mid H \in \mathcal{F} \,\}$.

If we would define "critical nodes" in a similar way for node-deletion problems it would turn out that *all* nodes are critical. In the following we will establish that the number of critical edges plays a crucial role in the advice complexity of online edge-deletion problems and ways how to compute the number of critical edges for special families $\mathcal{F}$. Please note first that it is quite easy to compute

$\#crit(\mathcal{F})$ if given a finite $\mathcal{F}$. A simple algorithm can achieve this by a polynomial number of subgraph isomorphism tests, which are of course by themselves NP-complete. The graphs in typical families $\mathcal{F}$ are usually small, so long running times are not a practical issue here.

Let us look at a very simple example. Let $\mathcal{F} = \{P_2, C_3\}$. Due to symmetry we have to look only at three different glueing operations: P_2 to P_2, C_3 to C_3, and P_2 to C_3. Whenever C_3 is involved, the resulting graph contains P_2 as an induced subgraph. For example, glueing C_3 to itself results in $C_4 \simeq C_3 \ominus_e C_3$. This means that the edges in C_3 are not critical. Glueing P_2 to itself can be done in two ways. The first result are two disjoint edges and the second is again P_2. Hence, the edges in P_2 are critical. In total, $\#crit(\mathcal{F}) = 2$.

Lemma 1. *Let G be a two-connected graph and $e \in E(G)$. Then G is two-connected iff $G \ominus_e G$ is two-connected.*

Proof. A connected graph is two-connected iff it has no cut-vertex. Let $e = xy$. Assume first that G has a cut-vertex. If it is x or y then clearly $G \ominus_e G$ is not even connected. Otherwise both parts of $G \ominus_e G$ have the corresponding vertex as a cut-vertex. The other direction of the proof is similar. □

Lemma 2. *Let G be a graph that contains vertices x, y, u, v and these four conditions hold:*

1. *x and y are connected by an edge.*
2. *There are two vertex-disjoint paths from u to x and from u to y.*
3. *There are two vertex-disjoint paths from v to x and from v to y.*
4. *The edge xy is not on any of those four paths.*

Then u and v are two-connected in G.

Proof. The vertices u, x, and y are on a cycle and therefore in the same two-connected component. The same holds for v, x, and y. □

The next lemma is technically the most complicated one in this paper. What it states about all two-connected graphs is also true for many other graphs and it can be checked easily for a concrete graph G. It opens a path to proving lower bounds on the advice complexity of edge-deletion problems in a way similar to how Theorem 3 works for node-deletion problems.

Lemma 3. *Let G be a two-connected graph and $e = xy \in E(G)$ one of its edges. Then $G \ominus_e G$ does not contain a subgraph that is isomorphic to G.*

Proof. Let us assume the contrary and that G is a minimal counterexample with respect to taking subgraphs. So we assume that indeed there is a $H \subseteq G \ominus_e G$ and $H \simeq G$. Because of its number of edges, H contains edges in both parts of $G \ominus_e G$ as one part has only $||G|| - 1$ edges. Hence, H contains x_1 or y_1. Since H (being isomorphic to G) is two-connected it must contain both x_1 and y_1; otherwise H would contain a cut-vertex ($\{x_1, y_1\}$ is a separator in $G \ominus_e G$).

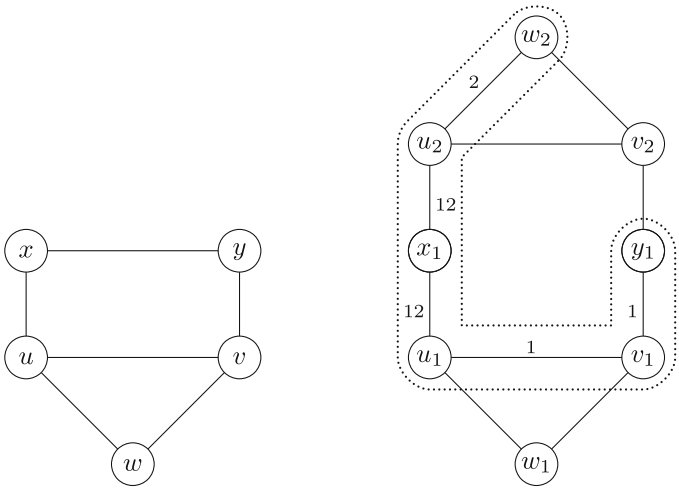

Fig. 2. A graph G is depicted left. The corresponding graph $G \ominus_{xy} G$ is on the right (here $x_1 = x_2$ and $y_1 = y_2$). Note that the edge corresponding to xy is missing in the right graph. An induced subgraph H is indicated by the dotted outline. (Please note that H is not isomorphic to G, which would be impossible by the very lemma we are about to prove.)

In the following we establish some notation. If $u \in V(G)$, then let u_1 and u_2 be the respective copies of u in the upper and lower part of $G \ominus_e G$. In particular $x_1 = x_2$, $y_1 = y_2$ and $u_1 \neq u_2$ if $u \notin \{x, y\}$.

If $u_1v_1 \in E(H)$, then we say that

- u_1v_1 is a 12-edge if $u_2v_2 \in E(H)$.
- u_1v_1 is a 1-edge if $u_2v_2 \notin E(H)$.

If $u_2v_2 \in E(H)$, then we say that

- u_2v_2 is a 12-edge if $u_1v_1 \in E(H)$.
- u_2v_2 is a 2-edge if $u_1v_1 \notin E(H)$.

In Fig. 2 the graph H has five edges. The 12-edges are x_1u_1 and x_2u_2, u_2w_2 is a 2-edge, and u_1v_1, v_1y_1 are 1-edges.

From H we construct a new graph H' (which will be a subgraph of G) as follows: The vertices of H' will be a subset of $V(G)$. Let $V(H') = \{\, u \in V(G) \mid u_1 \in V(H) \text{ or } u_2 \in V(H) \,\}$. The graph H' contains the edge uv iff $u_1v_1 \in E(H)$ or $u_2v_2 \in E(H)$ and additionally the edge xy. Then H' is clearly a subgraph of G.

What happens if we consider $H' \ominus_{xy} H'$? It must be a supergraph of H, which is isomorphic to G, which again contains H' as a subgraph:

$$H' \ominus_{xy} H' \supseteq H \simeq G \supseteq H' \tag{1}$$

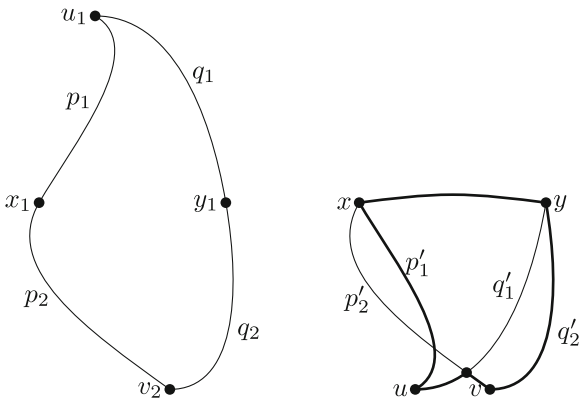

Fig. 3. The vertices u_1 and v_2 are two-connected in H (left). The resulting vertices u and v in H' are then also two-connected (right).

As H contains x_1 and y_1, H' contains x and y by definition. As H is two-connected there must be at least two vertex disjoint paths between any pair of vertices in H. Let u_1 and v_2 be such a pair. One of the paths between them has to go through x_1 and the other through y_1. Both paths start with a subpath consisting of only 1-edges and then ending with a subpath of 2-edges. Let us call these subpaths p_1, q_1, p_2, q_2 where p_1 connects u_1 with x_1 and p_2 connects x_1 with v_2. In H' there are isomorphic copies of these paths that need no longer be disjoint. There are, however, two disjoint paths p'_1 and q'_1 connecting u to x and y, and two other disjoint paths p'_2 and q'_2 connecting to v to x and y (see Fig. 3). By Lemma 2 then u and v are also two-connected in H'. If u and v do not originate from u_1 and v_2 in H, but, say, from u_1 and v_1, the situation is simpler because then all paths are automatically disjoint. Altogether, this means that H' is two-connected.

We have established the following three facts:

- $e \in E(H')$ (by construction of H'),
- H' is two-connected (previous paragraph),
- $H' \ominus_e H'$ contains a subgraph that is isomorphic to H', see (1).

If we look at the preconditions of Lemma 3 we see that H' is another counterexample. Moreover, $H' \subseteq G$. We have assumed that G is a minimal counterexample, so it cannot contain another smaller subgraph that is also a counterexample. Hence, $H' = G$.

As $H \simeq G$, the graphs H and H' must have the same number of vertices. This, however, cannot be the case if there is at least one 12-edge in H. One endpoint of a 12-edge has to be outside $\{x_1, y_1\}$ because $x_1y_1 \notin E(H)$. Let us assume x_1u_1 is such a 12-edge. Then there exists also the 12-edge $x_2u_2 = x_1u_2$. The graph H' contains the vertex u iff H contains u_1 or u_2. As H contains both u_1 and u_2, the number of vertices in H is larger than that in H'.

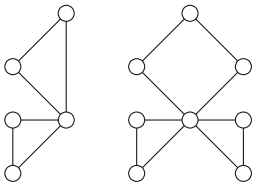

Fig. 4. Lemma 3 does not hold for single-connected graphs.

As this is impossible, the only remaining possibility is that there is not even a single 12-edge in H. The number of edges in H is $h_1 + h_2 + h_{12}$ if we denote the number of 1-, 2-, and 12-edges by h_1, h_2, and h_{12}. Then the number of edges in $G = H'$ is $h_1 + h_2 + h_{12}/2 + 1$ (note that h_{12} is always an even number). If indeed G and H are isomorphic, these counts must coincide, which is impossible if $h_{12} = 0$. This contradiction shows that our first assumption must have been wrong and the assumption was simply that Lemma 3 is wrong. □

The condition that G is two-connected Lemma 3 is necessary. Figure 4 shows that glueing an arbitrary connected graph to itself on a vertex yields a graph that is connected, but not two-connected and is a counterexample to the statement in Lemma 3. The next lemma shows that if we use $G \,{}_{yx}\ominus_{xy}\, G$ instead, a similar statement holds for graphs that are just connected.

Lemma 4. *Let G be a connected graph and $e = xy \in E(G)$ one of its edges. Then $G \,{}_{yx}\ominus_{xy}\, G$ does not contain a subgraph that is isomorphic to G.*

Proof. The outline of the proof is similar to the proof of Lemma 3. First we assume that there is a connected graph G and an edge $xy \in G(H)$ such that $H \subseteq G_{yx}\ominus_{xy}$ and H is isomorphic to G. Moreover, we assume without loss of generality that G is a minimal counterexample with respect to taking subgraphs. From this assumption we will derive a contraction.

We define H' analogous to H' in Lemma 3 and the only difference is that now $x_1 = y_2$ and $y_1 = x_2$. This time H' "automatically" contains both x and y: Since H has too many edges to fit in one part of $G_{yx}\ominus_{xy}$ it has to use x_1 or y_1. In either case H' contains both x and y.

As before we get a contradiction if there is no 12-edge:

$$||G|| = ||H||,\ ||H|| = h_1 + h_2 + h_{12},\ h_1 + h_2 \leq ||G|| - 1.$$

On the other hand, if there is at least one pair of 12-edges, say u_1v_1 and u_2v_2, then H' must be connected: G is connected therefore ${}_{yx}\oplus_{xy}$ is also connected. If ${}_{yx}\ominus_{xy}$ is disconnected then only the missing edge x_1y_1 can be responsible. Then H would consist of the connected components, one in the upper part and connected to x_1, the other in the lower part and connected to $y_1 = x_2$ (or the other way around). The corresponding parts in H' share the node u and therefore H' is connected after all. As above this shows that H' is another counterexample. Because of the assumption that G is minimal and $H' \subseteq G$ we get $H' = G$. On the other hand $h_{12} > 1$ implies that H' has fewer edges than G, a contradiction. □

Please note that Lemmata 3 and 4 have several consequences. First, it means that $\#crit(\{G\}) = ||G||$. It is also the key to the next theorem, which gives tight bounds for the $\mathcal{F}$-edge-deletion problem for the case that $\mathcal{F}$ consists of one connected graph.

Theorem 4. *Let H be a connected graph and G be an arbitrary graph. If opt is the minimal number of edges you have to delete from G to make it H-free, then every online algorithm requires $\lceil opt \log(||H||)\rceil$ advice bits to solve the $\{H\}$-edge-deletion problem optimally on input G. There is a deterministic algorithm that matches this bound.*

Proof. The adversary prepares a set $\mathcal{I}$ of $||H||^k$ different instances. We will show that every fixed algorithm (deterministic and using no advice) can solve at most one instance in $\mathcal{I}$ correctly.

Let $\{e_1, \ldots, e_m\} = E(H)$ and $H_i = H_{\,xy\oplus yx}\, H$ where $xy = e_i$. Each instance is a graph that has k components that are presented one after the other by the adversary. Each component is some H_i.

It is clear that making H_i H-free requires removing at least one edge and can be accomplished by removing exactly edge e_i by Lemma 3: Removing e_i from $H_{\,xy\oplus yx}$ leaves $H_{\,xy\ominus yx}\, H$, which does not contain H as an (induced) subgraph. On the other hand, removing any other edge will result in a graph that still contains H as an induced subgraph.

If there are less than $\lceil k \log(||H||)\rceil$ advice bits, the algorithm will react in the same way for two different instances. At least one of them will then be solved in a non-optimal way.

To show a matching upper bound is relatively easy. Whenever some H is found as an induced subgraph, one edge of it belongs to some optimal solution known to the oracle. This edge can be communicated by a number between 1 and $m = ||H||$. As only k times an edge is selected, k such numbers suffice and they can all be encoded as a single number between 1 and $||H||^k$. □

The following theorem generalizes the lower bound from Theorem 4 to arbitrary sets $\mathcal{F}$. Its small disadvantage is that it uses $\#crit(\mathcal{F})$, which has to be established for each $\mathcal{F}$ individually, which is a lot of work by hand, but easy with the help of a computer. A greater disadvantage is the missing matching upper bound, which we discuss after stating and proving the theorem.

Theorem 5. *Let $\mathcal{F}$ be an unordered set of graphs. Then solving the online $\mathcal{F}$-edge-deletion problem requires at least $\lceil opt \log(\#crit(\mathcal{F}))\rceil$ advice bits if opt is the optimal solution size.*

Proof. The proof is very similar to the proof of Theorem 4. The adversary chooses $H \in \mathcal{F}$ such that $\#crit_{\mathcal{F}}(H) = \#crit(\mathcal{F})$. Let C be the critical edges in H. Now again $|C|^k$ different instances are generated. For $xy \in C$ let $H_{xy} \in \mathcal{F}$ be another graph and $uv \in H_{xy}$ be an edge such that $H_{\,xy\ominus uv}\, H_{xy}$ is $\mathcal{F}$-free. Such a graph and edge exist by the definition of a critical edge. The instance presented by the adversary is $H_{\,xy\oplus uv}\, H_{xy}$.

An optimal online algorithm has to delete exactly the edge xy when confronted with $H_{xy \oplus_{uv}} H_{xy}$ as the deletion of any other edge either leaves H or H_{xy} as an induced subgraph. Deleting, however, xy turns $H_{xy \oplus_{uv}} H_{xy}$ into $H_{xy \ominus_{uv}} H_{xy}$, which is $\mathcal{F}$-free.

Again, there are $|C|^k$ different instances by repeating such choice k times. □

It would be nice to have a matching upper bound for Theorem 5, too, but it is easy to see that Theorem 5 is not always optimal. For example, consider claw- and diamond-free graphs. The edges in the diamond are not critical and Theorem 5 gives us only $opt \log(3)$ as a lower bound on the advice complexity. However, you can find five different induced supergraphs of the diamond that all require a different edge of the diamond to be removed in order to make it claw- and diamond-free.

Nevertheless, the following algorithm provides a matching upper bound in many, but not all, cases. The algorithm actually consists of the online algorithm and the behavior of the oracle providing the advice string.

The online algorithm proceeds as follows: It waits until the graph is no longer $\mathcal{F}$-free and then identifies one graph $H \in \mathcal{F}$ that is present as an induced subgraph. If there is no $H' \subseteq H$ such that $H' \in \mathcal{F}$ and H' has a critical edge, then it deletes an arbitrary edge from H. Otherwise it asks the oracle to name one critical edge in an appropriate subgraph H' and deletes it. It is easy to see that only $\lceil \log(\#crit(\mathcal{F})^{opt}) \rceil$ advice bits are used.

The oracle provides the following advice when the algorithm asks to identify a critical edge. The oracle identifies the edge in H that would be deleted by an optimal offline algorithm. If e is critical, it provides its number as advice. Otherwise it provides an arbitrary number.

It turns out that this algorithm is correct even for some extreme cases. For example, if $\mathcal{F}$ consists of all cycles, then the problem is to delete the minimum number of edges to make the graph acyclic. This is a very simple problem that can be solved greedily without any advice. It turns out that here all edges are non-critical. The online algorithm would just delete an arbitrary edge in a cycle it finds.

Another example is $\mathcal{F} = \{P_2, C_3\}$. Remember that P_2 contains critical and C_3 non-critical edges. Hence, as $P_2 \subseteq C_3$ the algorithm would ask the oracle which of two given edges in a triangle has to be deleted. With correct advice this yields an optimal solution and the algorithm matches the bound of Theorem 5.

Unfortunately, the algorithm does not work in all cases and indeed such an algorithm cannot exist because the lower bound in Theorem 5 is not optimal. Figure 5 shows a family $\mathcal{F}$ with $\#crit(\mathcal{F}) = 3$. The first graph has no critical edges at all. An adversary can, however, present the first graph and any algorithm has to delete one edge. It is not hard to see that the adversary can force the algorithm to make a non-optimal decision if the algorithm is not able to choose the right petal from which to delete the edge.

An important open question left is to find a construction for an optimal algorithm for every family $\mathcal{F}$.

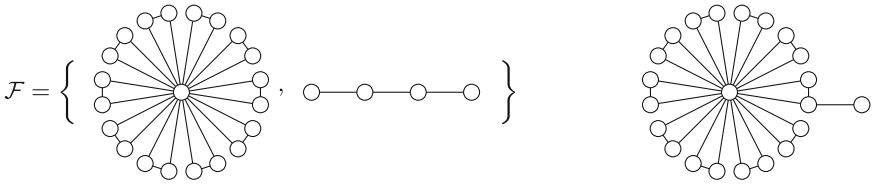

Fig. 5. A counterexample to the optimality of Theorem 5. The adversary presents the large graph in $\mathcal{F}$. The algorithm has to delete an edge e. Then the adversary adds another vertex as shown on the right side. The optimal solution is to delete two edges. The algorithm is not optimal if it did not choose e from the correct petal out of ten. Repeating this scheme for k rounds leads to $opt = 2k$ and at least $\log(10^k) = \log(10)/2 \cdot opt > 1.66opt$ advice bits to achieve optimality. Theorem 5 provides a lower bound of only $\lceil opt \log 3 \rceil \leq \lceil 1.59opt \rceil$.

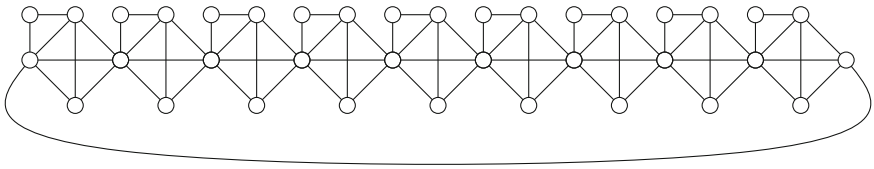

Fig. 6. Gadget for DH-edge-deletion.

The last example we consider shows a problem whose advice complexity is not bounded by a function of *opt*.

A graph is *distance hereditary* if the distance between two vertices does not change if we delete other vertices as long as they stay connected [1]. Surprisingly many graph classes are distance hereditary although it seems to be a severe restriction (which it is!). Among those graph classes are, e.g., ptolemaic graphs and cluster graphs. Let us call the corresponding problem the DH-edge-deletion problem. Its complexity is at least very close to the trivial upper bound from Theorem 1 and it is not bounded by function of the size of the optimal solution.

Theorem 6. *DH-edge-deletion requires at least* $opt(\log(n-1)-2)$ *advice bits.*

Proof. We construct k graphs of size $4t+1$ depicted in Fig. 6 for $t = 9$. The adversary presents first the cycle that consists of the long lower edge and the path going through the middle of the gadget. When the cycle is completed the algorithm is for the first time confronted with a graph that is not distance hereditary. Hence, the algorithm has to delete an edge. The only optimal choice is to delete the long edge making the graph distance-hereditary. Because of the rotation symmetry the adversary can construct t such gadgets and for each of them a different edge has to be deleted. Since there are k such gadgets arriving one after the other, the adversary can choose between t^k instances in total. The optimal solution deletes only $k = opt$ edges. The online algorithm requires therefore $k \log t$ advice bits. The graph has size $n = 4t+1$, so $t = (n-1)/4$, making the number of advice bits at least $k \log((n-1)/4) = k(\log(n-1)-2)$.□

Acknowledgement. I would like to thank Walter Unger, Janosch Fuchs, Ling-Ju Hung, and Li-Hsuan Chen for stimulating discussions about the proof of Lemma 3 and an anonymous referee for insightful feedback.

References

1. Bandelt, H.-J., Mulder, H.M.: Distance-hereditary graphs. J. Comb. Theory, Ser. B **41**(2), 182–208 (1986)
2. Bianchi, M.P., Böckenhauer, H.-J., Hromkovič, J., Krug, S., Steffen, B.: On the advice complexity of the online $L(2,1)$-coloring problem on paths and cycles. Theor. Comput. Sci. **554**, 22–39 (2014)
3. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
4. Cai, L.: Fixed-parameter tractability of graph modification problems for hereditary properties. Inf. Process. Lett. **58**, 171–176 (1996)
5. Komm, D.: An Introduction to Online Computation - Determinism, Randomization, Advice. Texts in Theoretical Computer Science. An EATCS Series. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-42749-2
6. Komm, D., Královič, R., Královič, R., Kudahl, Ch.: Advice complexity of the online induced subgraph problem. In: Proceedings of the 33rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2016), Leibniz International Proceedings in Informatics, vol. 58, pp. 59:1–59:13 (2016)
7. Steffen, B.: Advice complexity of online graph problems. Ph.D. thesis (2014)
8. Yannakakis, M.: Node- and edge-deletion NP-complete problems. In: Proceedings of the 10th ACM Symposium on Theory of Computing, pp. 253–264. ACM (1978)