Further Results on Online Node- and Edge-Deletion Problems with Advice

Li-Hsuan Chen[1], Ling-Ju Hung[2], Henri Lotze[3], and Peter Rossmanith[3(✉)]

[1] Kenkone Medical Co., Taipei, Taiwan
[2] National Taipei University of Business, Taipei, Taiwan
[3] RWTH Aachen University, Aachen, Germany

Abstract. In online edge- and node-deletion problems the input arrives node by node and an algorithm has to delete nodes or edges in order to keep the input graph in a given graph class at all times. We consider graph classes that can be characterized by forbidden sets of induced subgraphs and analyze the advice complexity of getting an optimal solution. We give almost tight lower and upper bounds for the Delayed H-Node Deletion Problem, where there is one forbidden induced subgraph that may or may not be disconnected and tight bounds on the Delayed $\mathcal{F}$-Node Deletion Problem, where we have an arbitrary number of forbidden connected graphs. For the latter result we present an algorithm that computes the advice complexity directly from $\mathcal{F}$. For the Delayed H-Node Deletion Problem the advice complexity is basically an easy function of the size of the biggest component in H.

Keywords: Online algorithm · Advice complexity · Node deletion · Edge deletion · Delayed decision model · Graph modification

1 Introduction

Many classical online problems can be formulated as follows: Given an instance $I = \{x_1, \ldots, x_n\}$ as a series of elements ordered from x_1 to x_n, an algorithm receives them iteratively in this order, having to decide whether to include x_i into its solution at the point it receives it. It can base this decision only on the previously revealed $x_1, \ldots, x_{i-1}$ and must neither remove x_i from its solution later nor include any of the previously revealed elements into its solution. A way to measure the performance of such an online algorithm is the *competitive ratio*, which compares how much worse it performs compared to an optimal offline algorithm [4]. An algorithm is c-competitive if the competitive ratio of the algorithm is bounded by a constant c.

In most classical online problems such as the k-Server Problem, the Paging Problem or the Knapsack Problem as well as most other online problems, receiving the next x_i of an instance coincides with an algorithm having to process this request. This makes a lot of sense in the previously mentioned

L. Gąsieniec et al. (Eds.): IWOCA 2020, LNCS 12126, pp. 140–153, 2020.
https://doi.org/10.1007/978-3-030-48966-3_11

problems, but arguably less sense when there is no "need to act" after an item of the instance is presented, which may regularly happen in the instances of the problem that we study in this paper: Informally, the requests are single nodes of a graph that are iteratively revealed and our task is to keep the graph induced by these nodes free of a set $\mathcal{F}$ of forbidden induced subgraphs by deleting nodes or edges. Obviously, there are sets and instances in which an arbitrary number of nodes can be revealed before any forbidden induced substructure is revealed. The offline variant of this problem was shown to be NP-Hard by Yannakakis [17].

In this work, we use a modified version, which we call the delayed decision model, which was already used in [16] and which is similar to the preemptive model used by Komm et al. [13]. We consider an instance $I = \{x_1, \ldots, x_n\}$ of an online minimization problem for which a solution $S \subseteq I$ has to satisfy some condition C. Again, an algorithm ALG has to decide whether to include any element into its solution S. We denote the intermediate solution of an algorithm on an instance I at the revelation of element x_i – before the decision on whether to include it in S – by $S_i^I(ALG)$. While in the classical definition, an algorithm has to decide on whether to include an element into its solution at the point of revelation, the algorithm may now wait until the condition C is violated by $S_i^I(ALG)$. It may then include any of the previously revealed elements into its solution, but is still unable to revert any of its previous selections.

A selection of online problems that do not admit any algorithm with a constantly bounded competitive ratio, such as the Minimum Vertex Cover Problem and in particular general node and edge deletion problems are constantly competitive with delayed decision.

A simple example is the online Minimum Vertex Cover Problem. The input I is a series of induced subgraphs $G[\{v_1\}], G[\{v_1, v_2\}], \ldots, G[\{v_1, \ldots, v_n\}]$ for which C states that $S_i^I(ALG)$ *is a vertex cover on* $G[\{v_1, \ldots, v_i\}]$. In this setting, an algorithm has to include nodes into its current solution only once an edge is revealed that is not covered yet. While the Minimum Vertex Cover Problem is competitive in the maximum degree Δ of an input graph in the classical online setting [5], a competitive ratio of 2 can be proven for the delayed decision setting: The upper bound is given by always taking both nodes of an uncovered edge into the solution (this is the classical 2-approximation algorithm). The lower bound can be achieved by presenting an edge $\{v_i, v_j\}$ and adding another edge to either v_i or v_j, depending on which node is not taken into the solution by a deterministic online algorithm. If both nodes are taken into the solution then no additional edge is introduced. This gadget can be repeated and forces a deterministic algorithm to take two nodes into the vertex cover where one suffices.

We denote by H a finite graph and by $\mathcal{F}$ a finite set of finite graphs. For a problem Π we denote the optimal solution size on an input I by $opt_\Pi(I)$.

The competitive ratio is a standard method to analyze online algorithms and a relatively new alternative is the *advice complexity* introduced by Dobrev, Královič, and Pardubská [7], revised by Hromkovič, Královič and Královič [11] and refined by Böckenhauer et al. [2]. The advice complexity measures the

amount of information about the future that is necessary to solve an online problem optimally or with a given competitive ratio. There is an oracle called "advisor" that knows the whole input instance and gives the online algorithm "advice" in the form of a binary string that can be read from a special advice tape. Many problems have been successfully analyzed in this model including the k-server problem [8], the Knapsack Problem [3], Job-Shop Scheduling [1] and many more. One criticism on the advice model is that in the real world such a powerful advisor usually cannot exist. However, the new research area of *learning-augmented algorithms* uses an AI-algorithm to guide classical algorithms to solve optimization problems and they are closely related to the advice complexity [14,15]. A strong application of advice complexity are the lower bounds it provides: For example, the online knapsack problem can be solved with a competitive ratio of two by a randomized algorithm. It has been shown that this competitive ratio cannot be improved with $o(\log n)$ advice bits.

We base our work on the definitions of advice complexity from [12] and [2], with a variation due to the modified online model we are working on: The length of the advice string is often measured as a function in the input length n, which usually almost coincides with the number of decisions an online algorithm has to make during its run. In the delayed decision model, the number of decisions may be smaller than n by a significant amount and we can measure the advice as $f(opt_{\Pi}(I))$, i.e., a function of the size of the optimum solution. This usually does not work in classical online algorithms.

Tight results for the advice complexity of the Delayed Connected $\mathcal{F}$-Node Deletion Problem and of the Delayed Connected H-Edge Deletion Problem were shown in [16]. We show upper and lower bounds for the general Delayed H-Node Deletion Problem and a tight bound for the Delayed Connected $\mathcal{F}$-Edge Deletion Problem. We leave open the exact advice complexity for the general Delayed $\mathcal{F}$-Node Deletion Problem and Delayed $\mathcal{F}$-Edge Deletion Problem, for which we can only provide lower bounds. Some proofs can be found only in the full version of this paper.

2 The $\mathcal{F}$-Node Deletion Problem and $\mathcal{F}$-Edge Deletion Problem Without Advice

For a graph $G = (V, E)$ we write $|G|$ to denote $|V(G)|$ and $||G||$ to denote $|E(G)|$. We use the symbol $\trianglelefteq$ to denote an induced subgraph relation, i.e. $A \trianglelefteq B$ iff A is an induced subgraph of B. We write $\mathcal{G}$ to denote the set of all graphs.

We write $G - U$ for $G[V(G) - U]$ and $G - u$ for $G - \{u\}$ and also use $G - E$ similarly for an edge set E. For graphs H and G we write $H \trianglelefteq_{\varphi} G$ if there exists an isomorphism φ such that $\varphi(H) \trianglelefteq G$. We call a set of graphs *unordered* if the members are pairwise maximal according to the induced subgraph relation $\trianglelefteq$. It is easy to see that every Delayed $\mathcal{F}$-Node Deletion Problem can be reduced to one with an unordered $\mathcal{F}$. A graph G is called $\mathcal{F}$-free if there is no $H_i \trianglelefteq_{\varphi} G$ for any $H_i \in \mathcal{F}$.

Definition 1. Let $\mathcal{F}$ be an unordered set of graphs. Let I be a sequence of growing induced subgraphs $G[\{v_1\}], \ldots, G[\{v_1, \ldots, v_n\}]$. The $\mathcal{F}$-Node Deletion Problem is to delete a minimum size set of nodes S from G such that $G - S$ is $\mathcal{F}$-free. We call $S_i^I \subseteq \{v_1, \ldots, v_i\}$ an (intermediate) solution for the $\mathcal{F}$-Node Deletion Problem on $G[\{v_1, \ldots, v_i\}]$ if $G[\{v_1, \ldots, v_i\}] - S_i^I$ is $\mathcal{F}$-free.

The Delayed $\mathcal{F}$-Node Deletion Problem is defined accordingly, with the condition C stating *The Graph* $G[\{v_1, \ldots, v_i\} - S_i^I(ALG)]$ *is* $\mathcal{F}$*-free* for all $i \in \{1, \ldots, n\}$ and some algorithm ALG. $\mathcal{F}$-Edge Deletion and Delayed $\mathcal{F}$-Edge Deletion are defined accordingly, with the solution being a set of edges. The graph is always revealed as a sequence of nodes. We will denote the Delayed $\mathcal{F}$-Node Deletion Problem for $\mathcal{F} = \{H\}$ as the Delayed H-Node Deletion Problem.

Lemma 1. *There is at least one* $\mathcal{F}$ *for which the* $\mathcal{F}$-Node Deletion Problem *is not* c*-competitive for any constant* c*.*

Lemma 1 is not surprising. It generalizes that Vertex Cover admits no constantly bounded competitive ratio [5].

Lemma 2. *There is at least one* $\mathcal{F}$ *for which the* $\mathcal{F}$-Edge Deletion Problem *is not* c*-competitive for any constant* c*.*

Lemma 3. *The* Delayed $\mathcal{F}$-Node Deletion Problem *is* k*-competitive for* $k = \max_{H \in \mathcal{F}}\{|H|\}$*. The* Delayed $\mathcal{F}$-Edge Deletion Problem *is* k*-competitive for* $k = \max_{H \in \mathcal{F}}\{||H||\}$*.*

Proof. Whenever an algorithm finds an induced H, it deletes all of its nodes, resp. edges. □

3 The Delayed H-Node Deletion Problem with Advice

If $\mathcal{F}$ consists of connected subgraphs, tight results have already been proven in [16]. The advice complexity is exactly $opt_{\mathcal{F}}(G)\log(|H|) + O(1)$ for a biggest graph $H \in \mathcal{F}$. The problem becomes harder when the graphs in $\mathcal{F}$ are disconnected and was left as an open question. We answer it partially by determining the advice complexity for the Delayed H-Node Deletion Problem, where H can be disconnected.

Definition 2. Let $C_G = \{C_1, C_2, \ldots, C_j\}$ denote the set of components of G.

If a forbidden graph H is disconnected, it may contain multiple copies of the same component, e.g., three disjoint triangles among other components. If we were only to delete triangles, we would thus have to delete all but two copies to make the graph of an instance H-free. We introduce some notation to determine the number and the actual copies of a *type* of component.

Definition 3. Given a graph G. For a connected graph C we define the packing $p_C(G)$ of C in G as the set of sets of pairwise node-disjoint copies of C in G and the packing number of C in G, $\nu_C(G)$, as $\max_{H \in p_C(G)}(|H|)$.

In other words, $\nu_C(G)$ is the maximal number of C's that can be packed node-disjointly into G.

We use the multiplicity of components in H in a lower bound that forces any algorithm to leave specific components such as the two specific triangles in our small example. To punish a wrong selection, we use a *redundancy construction* that maps a component C into a C' such that $C \trianglelefteq C'$ and even $C \trianglelefteq C' - \{v\}$ for every v holds, while C' does not contain two disjoint copies of C.

Definition 4. We call the graph H' a redundancy construction of a connected graph H with $|H| > 1$ if there exists an isomorphism $\varphi_1 \colon \mathcal{G} \to \mathcal{G}$ such that for every isomorphism $\varphi_2 \colon \mathcal{G} \to \mathcal{G}$ the following holds:

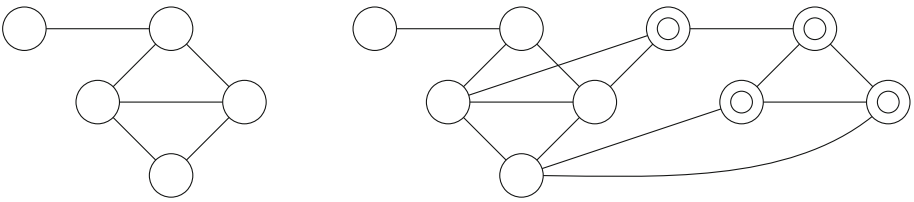
- $\varphi_1(H) \trianglelefteq H' - v$ for all $v \in V(H')$
- $\varphi_1(H) \not\trianglelefteq H' - V(\varphi_2(H))$ if $V(\varphi_2(H)) \subseteq V(H')$

To show that such a redundancy construction actually exists, we use the following transformation.

Definition 5. Given a connected graph $H = (V, E)$ with $V = \{v_1, \ldots, v_n\}$, $n > 1$, in some order and some $k \in [2, n]$ s.t. $(v_1, v_k) \in E(H)$. H' is then constructed in the following way: $V(H') = V(H) \cup \{v'_i \mid v_i \in V(H), i \geq 2\}$ and $E(H') = E(H) \cup \{(v'_i, v'_j) \mid (v_i, v_j) \in E(H), v'_i, v'_j \in V(H')\} \cup \{(v_1, v'_i) \mid (v_1, v_i) \in E(H)\} \cup \{(v_k, v'_j) \mid (v_1, v_j) \in E(H)\}$

Intuitively, we create a copy of H except for a single node v_1. The copied neighbors of v_1 are then connected with v_1. Lastly, some copied node is chosen and connected with the original neighbors of v_1.

Example 1. A graph H and its redundancy construction H':



Lemma 4. *The transformation in Definition 5 is a redundancy construction.*

We denote an optimal solution of the Delayed H-Node Deletion Problem on a graph G by $sol_H(G)$.

3.1 Lower Bound

Theorem 1. *Let H be a graph. Let C_{max} be a component of H of maximum size. Any online algorithm optimally solving the* Delayed H-Node Deletion Problem *uses at least $opt_H(G) \cdot \log|V(C_{max})| + (\nu_{C_{max}}(H) - 1) \cdot \log(opt_H(G))$ many advice bits on input G.*

Proof. Let $C_H = \{C_1, \ldots, C_j\}$ and $|V(C_1)| \leq \ldots \leq |V(C_j)|$. The adversary first presents $k \geq \max\{\, \nu_{C_i}(H) \mid C_i \in C_H \,\}$ disjoint copies of each $C_i \in C_H$ in an iterative way such that in each iteration one copy of each C_i is revealed node by node. If an algorithm deleted any nodes before an H is completed, the adversary would simply stop and the algorithm would not be optimal.

As soon as G is no longer H-free, any algorithm has to delete some node(s). For a $C_i \in C_H$ it can either delete all C_i except for $\nu_{C_i}(H) - 1$ occurrences and optionally some additional node(s). Obviously, deleting an additional node is not optimal, as the adversary would simply stop presenting nodes.

The following strategy will force an optimal online algorithm always to delete copies of C_{max}. After all k copies of all $C_i \in C_H$ are presented, additionally $\max_{C_i \in C_H}\{\nu_{C_i}(H)\} - \nu_{C_{max}}(H) + 1$ copies of each $C_i \in C_H \setminus C_{max}$ are presented. Deleting all C_{max} except for $\nu_{C_{max}}(H) - 1$ occurrences will thus only need $k - \nu_{C_{max}}(H) + 1$ deletions, while deleting any other component will need at least $k - \max_{C_i \in C_H}\{\nu_{C_i}(H)\} + 1 + \max_{C_i \in C_H}\{\nu_{C_i}(H)\} - \nu_{C_{max}}(H) + 1 = k - \nu_{C_{max}}(H) + 2$ deletions. Thus, it is always optimal for any algorithm to focus on C_{max} for deletion.

After all components have been revealed - and some deletion(s) had to be made - a redundancy construction such as the one from Definition 5 is used in order to *repair* an arbitrary set of $\nu_{C_{max}}(H) - 1$ copies of C_{max}. Every optimal algorithm will leave exactly $\nu_{C_{max}}(H) - 1$ copies of C_{max} after G is completely revealed. There are $\binom{opt_H(G) + \nu_{C_{max}}(H) - 1}{opt_H(G)}$ many different ways to distribute the affected components onto all components and an algorithm without advice cannot distinguish them. In particular, each of these instances is part of a different, unique optimal solution, which deletes a node from all but the $\nu_{C_{max}}(H) - 1$ subgraphs. If an algorithm has chosen to delete a node from a component that is affected by the redundancy construction, this component is now *repaired* and demands an additional deletion. By definition, applying the redundancy construction does not result in additional disjoint copies of C_{max}. Thus, it is still optimal to focus on C_{max} for deletion.

Finally, for every component that is not affected by a redundancy construction, the adversary glues a copy of C_{max} to one of its nodes as defined in [16]. It has $|V(C_{max})|$ ways to do so for each copy of C_{max}. Intuitively, the glueing operation joins two graphs by identifying a single node from both and connecting them by joining these two nodes into one.

We now measure how much advice an algorithm needs at least. First of all, it should be easy to see that the adversary is able to present $|V(C_{max})|^{opt_H(G)}$ many different instances regarding the deletion of nodes for the copies of C_{max} not selected for the redundancy construction.

Assuming $\nu_{C_{max}}(H) > 1$, any algorithm needs to determine the correct subset of $opt_H(G)$ components out of $k - 1$ presented ones to delete one node from. As the adversary has $\binom{opt_H(G) + \nu_{C_{max}}(H) - 1}{opt_H(G)}$ different ways to distribute these redundancies and since every single of these instances has a different unique optimal solution, any correct algorithm has to get advice on the complete distribution

Algorithm 1. Upper Bound: Delayed H-Node Deletion Problem

Input: Online graph G with $V(G) = \{v_1, \ldots, v_n\}$, H
Advisor computes $C_{min\nu} \in \arg\min_{C \in C_H} \{\nu_C(G) - \nu_C(H)\}$
Advisor computes $C_{min} \in \arg\min_{C \in C_{min\nu}} \{|V(C)|\}$
Advisor computes L, the list of labels marked for keeping
Read advice: C_{min} ▷ Which $C_{min} \in C_H$ to delete
Read advice: List L of numbers in range $[1, opt_H(G) + O(1)]$
$k \leftarrow 1$
Define $l\colon \mathcal{G} \to \mathbf{N}$, $l(G) = 0$
Define $labeled\colon \mathcal{G} \to \{0, 1\}$, $labeled(G) = 1$ iff $l(G) \neq 0$, otherwise $labeled(G) = 0$
for all $i \in \{1, \ldots, n\}$ **do**
    $G_i \leftarrow G[v_1, \ldots, v_i]$ ▷ Reveal next node
    if $\nu_{C_{min}}(G_i) \geq \nu_{C_{min}}(H)$ **then**
        $W \leftarrow \arg\max_{P \in p_{C_{min}}(G_i)} |P|$ ▷ Biggest Packings
        $\mathcal{H} \leftarrow \arg\max_{P \in W} \sum_{g \in P} labeled(g)$ ▷ Most labels
        Select $P \in \mathcal{H}$ ▷ Arbitrary set
        for all $C \in P$ **do** ▷ Label everything unlabeled
            if $l(C) = 0$ **then** $l(C) \leftarrow k$; $k \leftarrow k + 1$
        $S \leftarrow \{ C \in P \mid l(C) \notin L \}$ ▷ Select everything not marked for keeping
        for all $C \in S$ **do**
            Read advice: Which $v \in V(C)$ to delete
            Delete v out of G_i

in the size of at least $\log\binom{opt_H(G)+\nu_{C_{max}}(H)-1}{opt_H(G)} \geq (\nu_{C_{max}}(H) - 1) \cdot \log(opt_H(G))$ advice bits. □

3.2 Upper Bound

For simplicity of writing down the algorithm, we will assume in this section that we are only ever presented graphs which induce at least one forbidden subgraph H. Our algorithm can be easily transformed into one that only starts to read any advice once the first forbidden subgraph is completely revealed. For an instance with an online graph G with $V(G) = \{v_1, \ldots, v_n\}$ and a forbidden subgraph H, the advisor first computes the advice the algorithm is going to read during its run. It first identifies the set of components $C_{min\nu}$ which each require the fewest node deletions in G to make the graph H-free. Of these possible components, the advisor chooses the component with the fewest nodes which an optimal offline algorithm would choose, named C_{min} from here on. Finally, the advisor computes a list L of labels which will coincide with labels given by the algorithm to copies of C_{min} which are not to be deleted in an optimal solution. As there are at most $\nu_{C_{min}}(H) \cdot opt_H(G)$ node-disjoint copies of H in G and as Lemma 7 states that our algorithm uses at most $opt_H(G) + O(1)$ labels, we can limit the range of possible labels by $[1, opt_H(G) + O(1)]$. Finally, a number of advice bits is written for every deletion that the algorithm will make which encode the concrete node out of a copy of C_{min} is optimal to delete.

The algorithm starts by reading from the advice tape which component C_{min} to focus on for deletion and the list L, using self-delimiting encoding.

Whenever the next node x_i of the instance is revealed which fulfills $\nu_{C_{min}}(G_i) \geq \nu_{C_{min}}(H)$, i.e. that there are at least as many node-disjoint copies of C_{min} in the current graph as in H, the algorithm will delete nodes from the graph as described in the following, otherwise the algorithm simply waits for the next node to be revealed.

To identify which node(s) of G_i are to be deleted, the algorithm first identifies all biggest sets of node-disjoint copies of C_{min}. Of them it identifies a set P of which the most components have already received a label. Then all previously unlabeled copies of $C_{min} \in P$ receive a new unique label. The algorithm now looks at the label list L given by the advisor. Every copy of $C_{min} \in P$ whose label is not in L is now marked for deletion. The algorithm reads advice which concrete node out of every copy of C_{min} is optimal to delete.

Lemma 5. *Algorithm 1 is correct.*

Lemma 6. *Algorithm 1 is optimal.*

Definition 6. Given graphs G, H and a labeling function $l\colon \mathcal{G} \to \mathbf{N}$. We call a family $\mathcal{C}$ of induced subgraphs of G a *configuration*, if every element of $\mathcal{C}$ is isomorphic to H, $l(C) \neq 0$ for each $C \in \mathcal{C}$ and $V(C_1) \cap V(C_2) = \emptyset$ for all $C_1, C_2 \in \mathcal{C}$, $C_1 \neq C_2$. The *size* of a configuration is the number of induced subgraphs it contains.

Informally speaking, a configuration is a set of disjoint induced subgraphs of G that already have a label.

Lemma 7. *Given an online graph G, a forbidden graph H, as well as a subgraph $C \in C_H$ of which there may be at most $k = \nu_C(H) - 1$ disjoint copies present in G. Algorithm 1 assigns no more than $opt_H(G) + O(1)$ labels to G if the advisor assigns $C_{min} = C$ as specified in line 5.*

Theorem 2. *Let H be a graph. Let $C_{min\nu} = arg\,min_{C \in C_H}\{\nu_C(G) - \nu_C(H)\}$ and $C_{min} = arg\,min_{C \in C_{min\nu}}\{|V(C)|\}$. The* Delayed H-Node Deletion Problem *can be solved optimally using at most $opt_H(G) \cdot \log|V(C_{min})| + O(\log opt_H(G))$ many advice bits on input G.*

Proof. We count the number of advice bits used by Algorithm 1. We know by Lemma 5 and 6 that it is correct and optimal. The advice in line 5 is of constant size. As L only contains the labels for components which are not to be deleted and we limited the number of them by a constant in Lemma 7, only $O(\log opt_H(G))$ advice, using self-delimiting encoding, is needed in line 6.

Finally, the algorithm reads advice on which node of each copy of C_{min} that is part of $sol_H(G)$ to delete in line 21. This can be done using $opt_H(G) \cdot \log|V(C_{min})|$ advice bits in total. □

4 The Delayed Connected $\mathcal{F}$-Edge Deletion Problem

Let $(d_1, \ldots, d_k) \in \mathbf{N}^k$. Let $m(n)$ be the solution to the recurrence relation

$$m(n) = \begin{cases} \sum_{i=1}^{k} m(n-d_i) & \text{if } n \geq \max\{d_1, \ldots, d_k\} \\ c_n & \text{otherwise} \end{cases}$$

where $c_n \geq 0$ and some $c_i > 0$ for $0 \leq i < \max\{d_1, \ldots, d_k\}$. Let $\beta(d_1, \ldots, d_k) = \inf_\tau \{ \tau \mid m(n) = O(\tau^n) \}$. Note that β does not depend on the c_i's.

If $S = \{D_1, \ldots, D_k\}$ is a set of sets, then we define $\beta(S) = \beta(|D_1|, \ldots, |D_k|)$.

A homogeneous linear recurrence relation with constant coefficients usually has a solution of the form $\Theta(n^{k-1}\tau^n)$ if τ is the dominant singularity of the characteristic polynomial with multiplicity k [9]. However, here the coefficients of the characteristic polynomial are real numbers and there is exactly one sign change. By Descartes' rule of signs there is exactly one positive real root and therefore its multiplicity has to be one [6,10]. Therefore $m(n) = \Theta(\beta(S)^n)$.

Definition 7. Let $\mathcal{F}$ be a set of forbidden connected induced subgraphs and $H \in \mathcal{F}$. Let $S \subseteq 2^{E(H)}$.

1. A set $D \subseteq E(H)$ is *H-optimal* for a graph G if $H \trianglelefteq G$ and $G - D$ is $\mathcal{F}$-free and $opt_{\mathcal{F}}(G) = |D|$.
2. A set $D \subseteq E(H)$ is *H-good* for a graph G if $H \trianglelefteq G$ and D is a non-empty subset of some $\bar{D} \subseteq E(G)$ where $opt_{\mathcal{F}}(G) = |\bar{D}|$ and $G - \bar{D}$ is $\mathcal{F}$-free.
3. S is *H-sound* if $H - D$ is $\mathcal{F}$-free for every $D \in S$.
4. S is *H-sufficient* if for every connected graph G with $H \trianglelefteq G$ there is a $D \in S$ such that D is H-good for G.
5. S is *H-minimal* if for every $D \in S$, there is a graph G such that D is H-good for G, but every $D' \in S$, $D' \neq D$ is not.

Lemma 8. *Let $\mathcal{F} = \{H_1, \ldots, H_k\}$ be a set of connected graphs, G a graph and $D \subseteq E(H_i)$ that is H_i-good for G. Then there is a subgraph $G' \subseteq G$ such that D is H_i-optimal for G'.*

4.1 Upper Bound

Theorem 3. *Let $\mathcal{F} = \{H_1, \ldots, H_k\}$ be a set of connected graphs and let S_i be H_i-sound and H_i-sufficient for all $i \in \{1, \ldots, k\}$. Then there is an $m \in \mathbf{R}$ and an algorithm that solves the* Delayed Connected $\mathcal{F}$-Edge Deletion Problem *for every graph G with $m \cdot opt_{\mathcal{F}}(G) + O(1)$ many advice bits where $2^m \leq \beta(S_i)$ for all $i \in \{1, \ldots, k\}$.*

Proof. The algorithm receives $opt_{\mathcal{F}}(G) \cdot \log(\max_i\{\beta(S_i)\}) + O(1)$ many advice bits and then a graph G as a sequence of growing induced subgraphs. The algorithm interprets the advice as a number that can be between 0 and $O((\max_i\{\beta(S_i)\})^{opt_{\mathcal{F}}(G)})$.

The algorithm will delete in total exactly $opt_{\mathcal{F}}(G)$ edges. We analyze the total number of different advice strings the algorithm might use when deleting $opt_{\mathcal{F}}(G)$ edges.

When the algorithm receives a new node and its incident edges to form the next graph G it proceeds as follows: While G is not $\mathcal{F}$-free, choose some $H_i \in \mathcal{F}$ for which $H_i \trianglelefteq_{\varphi} G$. The advisor chooses one $D \in S_i$ for which $\varphi(D)$ is $\varphi(H_i)$-good for the graph at hand and puts it in the advice.

The advice strings are therefore partitioned into $|S_i|$ subsets, one for each $D \in S_i$. After deleting $\varphi(D)$ the algorithm proceeds on the graph $G - \varphi(D)$, where $opt_{\mathcal{F}}(G)$ is now by $|D|$ smaller. If $m(opt_{\mathcal{F}}(G))$ is the total number of advice strings we get the recurrence $m(opt_{\mathcal{F}}(G)) = \max_i\big(\sum_{D\in S_i} m(opt_{\mathcal{F}}(G) - |D|)\big)$ if $opt_F(G)$ is at least as big as every $D \in S_i$. Standard techniques show that $m(opt_{\mathcal{F}}(G)) = O(\max\{\beta(S_1), \ldots, \beta(S_k)\}^{opt_{\mathcal{F}}(G)})$. □

4.2 Lower Bound

Let $\mathcal{F} = \{H_1, \ldots, H_k\}$ be a set of connected graphs. We fix some correct algorithm A for the Delayed Connected $\mathcal{F}$-Edge Deletion Problem.

We define the sets $S_i = S_i(A)$ for $i = 1, \ldots, k$ as follows: $D \in S_i$ if and only if there is some input sequence $G_1, G_2, \ldots, G_t$ such that algorithm A deletes the edge set D' from G_t. Moreover, there is a set X and an isomorphism φ such that $G[X] \cong H_i$, $\varphi\colon V(H) \to X$, and $\varphi(D) = D' \cap E(G[X])$. Informally speaking, the edge sets in S_i are those that are deleted from some isomorphic copy of H_i by algorithm A in some scenario.

We will need the following technical lemma. It states that we can find a matching with special properties in every connected bipartite graph. The matching should have the following properties. Let U' be the partners in the matching on top and V' on the bottom.

The first property is $N(U') = V$, i.e., every node in V is connected to at least one node in U'. The second property states that we have an *induced* matching, i.e., that the graph induced by $U' \cup V'$ *is* a matching. The third property concerns the vertices in V': If $v \in V'$ then $N(v)$ contains several vertices from U, but exactly one node in U', i.e., its partner in the matching. We require that this partner is the *smallest* one in $N(v)$.

Lemma 9. *Let $G = (U, V, E)$ be a bipartite graph where $U = \{u_1, \ldots, u_k\}$. Let $\leq$ be a preorder on U such that $u_1 \leq \cdots \leq u_k$. Moreover, assume that $V \subseteq N(U)$, i.e., every node in V is connected to some node in U. Then there is a $U' \subseteq U$ and $V' \subseteq V$ such that*

1. *$N(U') = V$,*
2. *$G[U' \cup V']$ is a matching,*
3. *$\min N(v) \in U'$ for every $v \in V'$.*

Lemma 10. *Let $\mathcal{F} = \{H_1, \ldots, H_k\}$ be a set of connected graphs and S_i be H_i-sound and H_i-sufficient for all $i \in \{1, \ldots, k\}$. Then there are $S_i' \subseteq S_i$ such that S_i' is H_i-sound, H_i-sufficient and H_i-minimal and moreover:*

For every $D' \in S'_i$ there is a graph G with $H_i \trianglelefteq G$ such that D' is H_i-good for G and for every $D \in S_i \setminus S'_i$ that is also H_i-good for G, it holds that $|D| \geq |D'|$.

Theorem 4. *Let $\mathcal{F} = \{H_1, \ldots, H_k\}$ be a set of connected graphs and assume that there is an algorithm A that can solve the* Delayed Connected $\mathcal{F}$-Edge Deletion Problem *for all inputs G with at most $m \cdot opt_{\mathcal{F}}(G) + O(1)$ advice for some $m \in \mathbf{R}$. Then there exist S'_i that are H_i-sound, H_i-sufficient, and H_i-minimal and $\beta(S'_i) \leq 2^m$ for every $i \in \{1, \ldots, k\}$.*

Proof. By Lemma 10 there is an $S'_i = \{D_1, \ldots, D_r\} \subseteq S_i$ that is H_i-sound, H_i-sufficient, and H_i-minimal. It additionally has the property that for every $D' \in S'_i$ there is a graph G with $H_i \trianglelefteq G$ such that D' is H_i-good for G and for every $D \in S_i \setminus S'_i$ that is also H_i-good for G, it holds that $|D| \geq |D'|$.

Let $l \in \mathbf{N}$. The adversary prepares $\Theta(\beta(S')^l)$ many instances by repeating the following procedure until the size of the optimum solution for the presented graph exceeds $l - \max\{|D_1|, \ldots, |D_r|\}$.

1. The adversary presents a disjoint copy of H_i.
2. Then the adversary computes an induced supergraph G_j of H_i for which D_j is H_i-good, but all $D_{j'} \in S'_i$ with $j' \neq j$ are not H_i-good, for all $1 \leq j \leq r$. The existence of the graph G_j is guaranteed by the H_i-minimality of S'_i. In particular there is a $\bar{D}_j \supseteq D_j$ such that $\bar{D}_j$ is H_i-optimal for G_j. Let $D'_j = \bar{D}_j - D_j$. Let $G'_j = G_j - D'_j$. It is easy to see that D_j is H_i-optimal for G'_j.

We show that no other $D_{j'} \in S'_i$ is H_i-good for G'_j. Assume otherwise. If $D_{j'}$ is H_i-good for G'_j then there must be a $\bar{D}_{j'} \supseteq D_{j'}$ that is H_i-optimal for G'_j. Then $G_j - D_{j'} - ((\bar{D}_{j'} - D_{j'}) \cup D'_j)$ is $\mathcal{F}$-free. This implies that $D_{j'}$ is H_i-good for G_j contradicting the H_i-minimality of S'_i. Next the adversary transforms the H_i into one of the r possible G'_js and presents the new vertices. Then $opt_{\mathcal{F}}(G'_j) = |D_j|$. Hence, the optimal solution size increases by $|D_j|$.

In each round the input graph grows and the optimal solution size grows by $|D_j|$. As soon as that size exceeds $l - \max\{|D_1|, \ldots, |D_r|\}$ the adversary keeps presenting disjoint copies of H_i without turning them into bigger connected graphs until the size reaches exactly l. The number $N(l)$ of different instances is given by the following recurrence:

$$N(l) = \begin{cases} \sum_{j=1}^{r} N(l - |D_j|) & \text{if } l \geq \max\{|D_1|, \ldots, |D_r|\} \\ 1 & \text{otherwise} \end{cases}$$

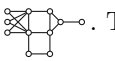
It is easy to see that $N(l) = \Theta(\beta(S'_i)^l)$. The algorithm has to react differently on all of these instances: When the algorithm sees a new H_i to be turned into one of $G'_1, \ldots, G'_r$, it deletes different edge sets for each of the r possibilities.

The adversary constructed an instance that consists of a sequence of disjoint graphs $G'_{i_1}, \ldots, G'_{i_t}$ from the set $\{G'_1, \ldots, G'_r\}$ of which the total size is at least $\sum_{j=1}^{t} opt_{\mathcal{F}}(G_{i_j}) - \max\{|D_1|, \ldots, |D_r|\}$ and $O(1)$ many copies of H_i. If G is the

whole constructed instance we have $opt_{\mathcal{F}}(G) = l + O(1)$ because $opt_F(H_i)_{\mathcal{F}} = O(1)$. Together with $N(l) = \Theta(\beta(S'_i)^l)$ this means that Algorithm A uses at least $\log N(l) = l \cdot \log \beta(S'_i) + O(1) = opt_{\mathcal{F}}(G) \log \beta(S'_i) + O(1)$ advice bits. Assume Algorithm A uses at most $m \cdot opt_{\mathcal{F}}(G) + O(1)$ advice bits on every graph G as stated in the precondition above. Then m cannot be smaller than $\log \beta(S'_i)$ for every $i \in \{1, \ldots, k\}$ because $opt_{\mathcal{F}}(G)$ can be become arbitrarily big. □

Lemma 11. *Let $\mathcal{F}$ be a set of connected forbidden graphs, $H \in \mathcal{F}$, and $S \subseteq 2^{E(H)}$. There is an algorithm that can decide whether S is H-sufficient.*

Proof. It is sufficient to verify for all connected graphs G with $H \trianglelefteq G$ that some $D \in S$ is H-good for G, i.e., there is an optimal solution for G that contains D. By Lemma 8 we can restrict our search to all such G's that have an optimal solution that is a subset of $E(H)$. There are infinitely many graphs G to check. To overcome this we define the *unfolding* of G, written $\Upsilon(G)$, as the set of the following graphs: Remember that $H \trianglelefteq G$. If there is some $H' \in \mathcal{F}$ with $H' \trianglelefteq_{\varphi} G$ then $G[E(H) \cup E(\varphi(H'))] \in \Upsilon(G)$ (for every possible φ). If, however, $\Upsilon(G)$ contains two graphs G' and G'' that are isomorphic via an isomorphism that is the identity on $V(H)$, then only the lexicographically smaller one is retained.

This means that the unfolding of G contains all induced subgraphs that consist of H and one other copy of some forbidden induced subgraph from $\mathcal{F}$ that must overlap with H in some way (because we assumed that G has an optimal solution that consists solely of edges from H). Here is a small example: Let $\mathcal{F} = \{$  $\}$, $H = \{$$\}$, $G =$  . Then $\Upsilon(G) = \{$  $\}$.

It is easy to see that deleting some $D \subseteq E(H)$ from G makes it $\mathcal{F}$-free iff deleting the same D from all graphs $G' \in \Upsilon(G)$ makes all these G' $\mathcal{F}$-free. Hence, there is an optimal solution for G that is a subset of $E(H)$ iff there is such a subset that is "optimal" for $\Upsilon(G)$ (i.e., deletion of no smaller edge set can make all graphs in $\Upsilon(G)$ $\mathcal{F}$-free).

There are only finitely possibilities for $\Upsilon(G)$ and we can enumerate all of them. Let us say this enumeration is $\Upsilon_1, \ldots, \Upsilon_t$. For each Υ_i we first find out, whether there is a G with $\Upsilon(G) = \Upsilon_i$. We can do this by enumerating all graphs G up to a size that does not exceed the sum of the sizes of all graphs in Υ_i and computing $\Upsilon(G)$ for them. If indeed $\Upsilon(G) = \Upsilon_i$ then we test whether S is H-good for G. Iff these tests pass for all i then S is indeed H-sufficient. □

Theorem 5. *Let $\mathcal{F} = \{H_1, \ldots, H_k\}$ be connected graphs. The advice complexity for* Delayed Connected $\mathcal{F}$-Edge Deletion *is $m \cdot opt_{\mathcal{F}}(G) + O(1)$ where $m = \max_{i \in \{1,\ldots,k\}} \min\{\log \beta(S) \mid S \subseteq 2^{E(H)}, S$ is H_i-sound and H_i-sufficient$\}$. There is an algorithm that can compute m from $\mathcal{F}$. More specifically, there is an algorithm that gets $\mathcal{F}$ and $t \in \mathbf{N}$ as the input and returns the tth bit of the binary representation of m.*

Proof. "$\leq$" by Theorem 3. "$\geq$" by Theorem 4. An algorithm can enumerate all possible $S \subseteq E(H)$ and then test if S is H_i-sound and H_i-sufficient (by Lemma 11). Then $\beta(S)$ is computed by finding the only real root of the characteristic polynomial of the corresponding recurrence relations [9]. □

Acknowledgement. We like to thank Ratislav Královič for helping significantly to simplify the proof of Lemma 7.

References

1. Böckenhauer, H., Komm, D., Královic, R., Královic, R., Mömke, T.: On the advice complexity of online problems. In: Algorithms and Computation, 20th International Symposium, ISAAC 2009, Honolulu, Hawaii, USA, 16–18 December 2009, Proceedings, pp. 331–340 (2009). https://doi.org/10.1007/978-3-642-10631-6_35
2. Böckenhauer, H., Komm, D., Královic, R., Královic, R., Mömke, T.: Online algorithms with advice: the tape model. Inf. Comput. **254**, 59–83 (2017). https://doi.org/10.1016/j.ic.2017.03.001
3. Böckenhauer, H., Komm, D., Královic, R., Rossmanith, P.: The online knapsack problem: advice and randomization. Theoret. Comput. Sci. **527**, 61–72 (2014). https://doi.org/10.1016/j.tcs.2014.01.027
4. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
5. Demange, M., Paschos, V.T.: On-line vertex-covering. Theoret. Comput. Sci. **332**(1–3), 83–108 (2005). https://doi.org/10.1016/j.tcs.2004.08.015
6. Descartes, R.: Discours de la methode pour bien conduire sa raison, et chercher la verité dans les sciences. Plus la Dioptriqve. Les Meteores. Et la Geometrie. - Qui sont des essais de cete Methode. De l'Imprimerie de Ian Maire (1637)
7. Dobrev, S., Královic, R., Pardubská, D.: Measuring the problem-relevant information in input. ITA **43**(3), 585–613 (2009). https://doi.org/10.1051/ita/2009012
8. Emek, Y., Fraigniaud, P., Korman, A., Rosén, A.: Online computation with advice. Theoret. Comput. Sci. **412**(24), 2642–2656 (2011). https://doi.org/10.1016/j.tcs.2010.08.007
9. Greene, D.H., Knuth, D.E.: Mathematics for the Analysis of Algorithms, 3rd edn. Birkhäuser, Boston (1990)
10. Henrici, P.: Applied and Computational Complex Analysis, vol. 1. Wiley, New York (1988)
11. Hromkovič, J., Královič, R., Královič, R.: Information complexity of online problems. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 24–36. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15155-2_3
12. 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
13. Komm, D., Královic, R., Královic, R., Kudahl, C.: Advice complexity of the online induced subgraph problem. In: Faliszewski, P., Muscholl, A., Niedermeier, R. (eds.) 41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, 22–26 August 2016 - Kraków, Poland, LIPIcs, vol. 58, pp. 59:1–59:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2016). https://doi.org/10.4230/LIPIcs.MFCS.2016.59
14. Lykouris, T., Vassilvitskii, S.: Competitive caching with machine learned advice. In: Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, 10–15 July 2018, pp. 3302–3311 (2018)
15. Purohit, M., Svitkina, Z., Kumar, R.: Improving online algorithms via ML predictions. In: Advances in Neural Information Processing Systems, vol. 31, pp. 9684–9693 (2018)

16. Rossmanith, P.: On the advice complexity of online edge- and node-deletion problems. In: Adventures Between Lower Bounds and Higher Altitudes - Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday, pp. 449–462 (2018). https://doi.org/10.1007/978-3-319-98355-4_26
17. Yannakakis, M.: Node- and edge-deletion np-complete problems. In: Lipton, R.J., Burkhard, W.A., Savitch, W.J., Friedman, E.P., Aho, A.V. (eds.) Proceedings of the 10th Annual ACM Symposium on Theory of Computing, 1–3 May 1978, San Diego, California, USA, pp. 253–264. ACM (1978). https://doi.org/10.1145/800133.804355