Algorithmica (2021) 83:2719–2753
https://doi.org/10.1007/s00453-021-00840-9

Online Node- and Edge-Deletion Problems with Advice

Li-Hsuan Chen[1] · Ling-Ju Hung[2] · Henri Lotze[3] · Peter Rossmanith[3]

Received: 4 August 2020 / Accepted: 21 May 2021 / Published online: 30 June 2021

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 Π at all times. We consider only hereditary properties Π, for which optimal online algorithms exist and which can be characterized by a set of forbidden subgraphs $\mathcal{F}$ and analyze the advice complexity of getting an optimal solution. We give almost tight bounds on the Delayed Connected $\mathcal{F}$-Node-Deletion Problem, where all graphs of the family $\mathcal{F}$ have to be connected and almost tight lower and upper bounds for the Delayed H-Node-Deletion Problem, where there is one forbidden induced subgraph H that may be connected or not. For the Delayed H-Node-Deletion Problem the advice complexity is basically an easy function of the size of the biggest component in H. Additionally, we give tight bounds on the Delayed Connected $\mathcal{F}$-Edge-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 $\mathcal{F}$. We give a separate analysis for the Delayed Connected H-Edge-Deletion Problem, which is less general but admits a bound that is easier to compute.

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

Preliminary versions of two parts of this work were published in the proceedings of IWOCA 2020 [10] and in *Adventures Between Lower Bounds and Higher Altitudes—Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday* [22].

✉ Henri Lotze
lotze@cs.rwth-aachen.de

1 Pro Brand International (TW), Inc., Hsinchu, Taiwan

2 National Taipei University of Business, Taipei, Taiwan

3 RWTH Aachen University, Aachen, Germany

1 Introduction

A number of classical online problems can be formulated as follows: Given an instance $I = (x_1, \dots, x_n)$ as a series of elements ordered from x_1 to x_n, an algorithm receives them iteratively in this order, having to decide immediately whether to include x_i into its solution. It can base this decision only on the previously revealed $x_1, \dots, x_{i-1}$ and must neither remove x_i from its solution later nor include any of the previously discarded elements. 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 [8]. An algorithm is strictly 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, receiving the next x_i requires immediate action. This makes a lot of sense in the mentioned problems, but sometimes there is no "need to act" immediately, which is often the case for the problem that we study in this work: 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 $\mathcal{F}$ and instances in which an arbitrary number of nodes can be revealed before any forbidden induced substructure appears.

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 [24]. Cai analyzed the parameterized complexity of graph modification problems [9]. 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.

The modified online model that we use was first introduced by Komm et al. [19] as the *preemptive* model. We give a slightly different formulation, which more closely matches our problem and call it the *delayed decision* model. We consider an instance $I = (x_1, \dots, x_n)$ 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 must never remove any element from it.

Some online problems that do not allow for competitive algorithms, such as the Minimum Vertex Cover Problem and in particular general node- and edge-deletion problems allow for competitive algorithms in the delayed decision model.

In the Minimum Vertex Cover Problem, the input I is a series of induced subgraphs $G[\{v_1\}], G[\{v_1, v_2\}], \dots, G[\{v_1, \dots, v_n\}]$ for which C states: "$S_i^I(ALG)$ is a vertex cover on $G[\{v_1, \dots, v_i\}]$". In the delayed decision model, an algorithm has to include nodes into its current solution only when an edge is revealed that is not covered yet. While the Minimum Vertex Cover Problem does not allow for competitive algorithms in the classical online setting [11], 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 offline algorithm). The lower bound can be achieved by presenting an edge v_iv_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.

The competitive ratio is the classical method to analyze online algorithms and a relatively new alternative is the *advice complexity* introduced by Dobrev et al. [13], revised by Hromkovič et al. [17], and refined by Böckenhauer et al. [4]. 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* who 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 [14], the Knapsack Problem [6], Job-Shop Scheduling [3], and many more. One criticism of 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 [20, 21]. 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 [2, 5, 6].

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.

We base our work on the definitions of advice complexity from Komm [18] and Böckenhauer et al. [4], 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 of 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 a function of the size of the optimum solution. This usually does not work in classical online algorithms.

In this work, we give a lower bound of $\lceil opt \cdot \log s \rceil$ and an upper bound of $\lceil opt \cdot \log s \rceil + \log opt + 2 \log \log opt$ on the advice complexity of the Delayed Connected $\mathcal{F}$-Node-Deletion Problem, where s is the size of the biggest graph in $\mathcal{F}$ and opt is the size of the optimal solution. We show lower and upper bounds for the Delayed H-Node-Deletion Problem, which are roughly $opt \cdot \log |V(C_{max})| + \Omega(\log opt)$ and $opt \cdot (\log |V(C_{max} + |C_H|)| + O(\log opt)$

Table 1 Advice complexity for variations of node-deletion problems

Node-deletion	Single graph H forbidden	Family $\mathcal{F}$ of graphs forbidden
All graphs connected	Theorem 3†	Theorem 3†
Arbitrary graphs	Theorems 4 and 5†	Open

A dagger (†) symbolizes a remaining gap

Table 2 Advice complexity for variations of edge-deletion problems

Edge-deletion	Single graph H forbidden	Family $\mathcal{F}$ of graphs forbidden
All graphs connected	Theorem 9†	Theorems 6 and 7
Arbitrary graphs	Open	Open

A dagger (†) symbolizes a remaining gap

respectively, with C_{max} being the biggest component of H. More precise results are given in Theorems 4 and 5.

In the second main part, we give a tight bound for the Delayed Connected $\mathcal{F}$-Edge-Deletion Problem, namely $m(\mathcal{F}) \cdot opt_{\mathcal{F}}(G) + O(1)$, where computing $m(\mathcal{F})$ is rather involved. We provide an algorithm that computes $m(\mathcal{F})$ for every concrete $\mathcal{F}$. Afterwards, since the results for the Delayed Connected $\mathcal{F}$-Edge-Deletion Problem are only computable with some work for concrete sets of forbidden graphs, we provide results for a specialized version of the problem, namely the Delayed Connected H-Edge-Deletion Problem. The advice complexity for this problem is a simple function in the size of the optimal solution, lower bounded by $\lceil opt \cdot \log(||H||)\rceil$ bits where $||H||$ denotes the number of edges in H and upper bounded by $\lceil opt \cdot \log ||H||\rceil + \log opt + 2 \log\log opt$. 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.

An overview over all possible problem types, references to the concrete theorems and open problems, can be found in Tables 1 and 2.

An example that we will examine throughout this paper is the class of *trivially perfect* graphs, which were first studied by Wolk [23]. They are exactly the graphs without any induced path and cycle on 4 nodes, i.e. $\mathcal{F} = \{$, $\}$. We will see in Example 1 that the advice complexity for this graph class is exactly $2 \cdot opt$ when tasked with deleting nodes. Example 3 will show that $\log 3 \cdot opt + O(1)$ advice bits are necessary and sufficient to optimally solve the problem in the case of edge deletions.

2 Preliminaries

We will use the usual notation for graphs, which will always be simple, undirected, and loop-free.

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 denote by H a finite graph and by $\mathcal{F}$ a finite set of finite graphs, if not stated otherwise.

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 $\varphi(H) \trianglelefteq G$. A graph G is called $\mathcal{F}$-free if there is no $H_i \trianglelefteq_\varphi G$ for any $H_i \in \mathcal{F}$. A path with k nodes and a clique with k nodes are denoted by P_k and K_k respectively. An edge between nodes x and y is called xy.

For a problem Π we denote the optimal solution size on an input I by $opt_\Pi(I)$. If Π is the class of $\mathcal{F}$-free graphs, then we also write $opt_\mathcal{F}(I)$ instead of $opt_\Pi(I)$ and $opt_H(I) = opt_{\{H\}}(I)$. As it is always clear whether we refer to a node- or edge-deletion problem, we do not specifically mention it in the notation.

If the context is clear or we do not refer to a concrete instance or even problem, we sometimes abbreviate the notation for the optimal solution size to opt.

By $\log(n)$ we always denote the logarithm to base 2.

3 General Graph Deletion Problems

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 such 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. 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_3 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_3 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_3 which is in the next step expanded into a P_4 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.

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

Proof (1) Whenever the algorithm detects that the graph is not in Π, a node has to be deleted. There 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 $\lceil opt \cdot \log n \rceil$ many 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 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 $\lceil \log((n^2/2)^{opt}) \rceil \leq 2opt \cdot \log n$ advice 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 big 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.

Theorem 1 is restricted to 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*.

Definition 1 A set of graphs $\mathcal{F}$ is called *unordered* if for every $H_1, H_2 \in \mathcal{F}$ with $H_1 \neq H_2$ it holds that $H_1 \ntrianglelefteq H_2$.

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. etc.

Hereditary graph properties are *exactly* those properties that can be solved optimally in this model if "optimal" means that the online solution is not bigger than the best offline 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.

An 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. In particular, the amount of advice bits cannot be dependent only on the size of the optimal solution. For example, if the input graph has n nodes, any Π-node-deletion problem can be optimally solved with n bits of advice.

We now give definitions for the main problems that we study in this work.

Definition 2 Let $\mathcal{F}$ be an unordered set of graphs. Let I be a sequence of growing induced subgraphs $G[\{v_1\}], \dots, G[\{v_1, \dots, v_n\}]$. Then 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 $S_i^I \subseteq \{v_1, \dots, v_i\}$ an (intermediate) solution for the $\mathcal{F}$-Node-Deletion Problem on $G[\{v_1, \dots, v_i\}]$ if $G[\{v_1, \dots, v_i\}] - S_i^I$ is $\mathcal{F}$-free.

The Delayed $\mathcal{F}$-Node-Deletion Problem is defined accordingly, with the condition C stating "The graph $G[\{v_1, \dots, v_i\} - S_i^I(ALG)]$ is $\mathcal{F}$-free" for all $i \in \{1, \dots, n\}$ and some feasible 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.

We start by giving a short proof that the $\mathcal{F}$-Node-Deletion Problem does not generally admit a constant competitive ratio. We continue by giving upper bounds for the preemptive model in which no advice is used.

Lemma 1 *Let H be a connected graph with $|H| > 1$. Then there is no algorithm for the H-Node-Deletion Problem that is c-competitive for any constant c.*

Proof Any correct online algorithm has to delete a node from each copy of H. The adversary starts constructing a copy of H. If an algorithm chooses to delete any node before H is completed, the copy is not completed by the adversary. In this special case, the adversary instead chooses to construct another node-disjoint copy of H.

If any algorithm instead chooses to delete a node only once a copy of H is completed, it can only do so by deleting the node v_i that was last presented. The rest of the instance then consists of an arbitrary number of copies of v_i which *repair* H repeatedly. An optimal algorithm simply deletes a single $v_j \in V(H), v_j \neq v_i$, that is not a copy of v_i. □

Lemma 1 is not surprising. It generalizes that Vertex Cover admits no constantly bounded competitive ratio [11].

Note however that this does not mean that there are no $\mathcal{F}$ for which the problem admits a constantly bounded competitive ratio. An example is the $\mathcal{F}$-Node-Deletion Problem with $\mathcal{F} = \{K_1K_1, P_2\}$, i.e. both two isolated nodes and an edge are forbidden. For this problem, an algorithm can simply delete any node that is presented after the first one. As the optimal solution for any graph over this $\mathcal{F}$ is deleting every node except for a single one, this algorithm is optimal.

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

Proof Let $\mathcal{F} = \{P_k\}$ for any fixed $k > 3$. Any correct online algorithm has to delete an edge from each copy of the path P_k. The adversary starts constructing a copy of P_k finishing with a vertex at the end of the path. If an algorithm chooses to delete any edge before P_k is completed, the copy is not completed by the adversary. In this special case, the adversary instead chooses to construct another node-disjoint copy of P_k.

If any algorithm instead chooses to delete an edge only once a copy of P_k is completed, it can only do so by deleting an edge e_j presented last adjacent to the newest node v_i. The rest of the instance then consists of an arbitrary number of copies of v_i which *repair* P_k repeatedly. An optimal algorithm simply deletes a single $e_k \in E(P_k), e_k \notin e_j$. □

Next, we take a look at a bound on the competitive ratio for algorithms that use the *delayed* model.

Lemma 3 *The Delayed $\mathcal{F}$-Node-Deletion Problem admits an algorithms that is k-competitive for $k = \max_{H \in \mathcal{F}}\{|H|\}$ and the Delayed $\mathcal{F}$-Edge-Deletion Problem admits and algorithm that 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. □

Note that this may seem like a very rough upper bound at first, but there are sets $\mathcal{F}$ for which this bound on the Delayed $\mathcal{F}$-Node-Deletion Problem is tight, as we will show now. For the following lemma, C_k denotes a cycle with k nodes.

Lemma 4 *Let $k > 4$ and $\mathcal{F} = \{C_k\}$. Then any algorithm solving the Delayed $\mathcal{F}$-Node-Deletion Problem cannot achieve a competitive ratio better than k.*

Proof There are k adversarial strategies. The ith strategy is the following: First, a C_k is presented. Whenever a node is deleted, another node with the same current neighborhood of the deleted node is reinserted. This is done for all nodes except for the node v_i of the cycle. We call the set of these reinserted nodes V'. We call the graph built by the ith strategy G_i.

We now want to show that deleting the last node of the cycle makes the gadget C_k-free. Since this last node is chosen arbitrarily, this would show that we can force any algorithm to use k deletions for a gadget for which one deletion would be sufficient.

It is clear by construction that deleting any other single node does not make the gadget C_k-free due to the reinsertions. For a contradiction we assume that there remains an induced C_k in the gadget after the deletion of v_i.

First, it is clear that deleting any $v \in V'$ and thus forcing a reinsertion of a copy v' does not produce any new C_k: If C_k^v is a cycle for which $v \in V(C_k^v)$ and $C_k^{v'}$ is a cycle for which $v' \in V(C_k^{v'})$ then $C_k^{v'} - v' = C_k^v - v$. Each replacement node only closes new C_4 in $G_i - v_i$. W.l.o.g. we look at a gadget for which only one copy of each replacement node is presented, i.e., $|V'| = k - 1$.

As there are only $k - 1$ replacement nodes, any C_k in the gadget - except for the original C_k - has to consist of at least one node of the original C_k and at least one of the replacement nodes.

We break any C_k that consists of $k - 1$ of the original nodes and one replacement node by deleting v_k, as there is no replacement node for v_k by construction.

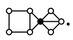
Thus, at least two nodes of V' have to be in any remaining C_k and none of the nodes of any C_k may include v_k.

Since by construction, every node of V' closes a new C_4, any bigger cycle of the graph must have at least one edge that connects two non-neighboring nodes, thus not inducing any cycle of length bigger than four. Thus, there are no additional C_k in the graph. □

It is easy to see that this proof holds even if $\mathcal{F}$ is a family of cycles of length at least 5.

4 The Delayed H-Node-Deletion Problem with Advice

If $\mathcal{F}$ consists of connected subgraphs, we can provide a simple proof giving us an almost tight bound on the advice complexity as follows.

In the following we use a *gluing* operation that works as follows: Given two graphs G and G', we identify a single node from G and a single node from G'. A small example where the identified nodes are marked black is the following: Gluing together with results in .

Theorem 3 *Let $\mathcal{F}$ be a not necessarily finite set of connected graphs and G the input graph. To be optimally solved the* Delayed Connected $\mathcal{F}$-Node-Deletion Problem *requires at least $\lceil opt_{\mathcal{F}}(G) \cdot \log s \rceil$ many advice bits, where s is the size of the biggest graph in $\mathcal{F}$. There is an algorithm almost matching this bound using $\lceil opt_{\mathcal{F}}(G) \cdot \log s \rceil + \log opt_{\mathcal{F}}(G) + 2 \log \log opt_{\mathcal{F}}(G)$ advice bits. If $s = \infty$ then no algorithm can be optimal with $f(opt)$ advice bits for any function f.*

Proof Let H be the biggest graph in $\mathcal{F}$ 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 add nodes to build 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 $\log(|H|^k) = k \cdot \log(|H|)$ advice bits. It is easy to see—using self-delimiting encoding as in [7]—that $\lceil k \cdot \log s \rceil + \log k + 2 \log \log k$ bits are also sufficient if $k = opt_{\mathcal{F}}(G)$ because the algorithm has to pick the right node of H when finding $H \in \mathcal{F}$ as an induced subgraph. The self-delimiting encoding is of course not needed if the size of s is a power of 2. In this case, the bound is tight.

If s is infinite, one can take an arbitrarily big H, so no finite amount of advice bits is sufficient. □

Example 1 Given $\mathcal{F} = \{$, $\}$. To compute the advice complexity, we identify the size of the biggest $H \in \mathcal{F}$, which is 4. Thus, the advice complexity as stated in Theorem 3 is exactly $\lceil opt_{\mathcal{F}}(G) \cdot \log 4 \rceil = 2 \cdot opt_{\mathcal{F}}(G)$.

The problem becomes harder when the graphs in $\mathcal{F}$ are disconnected, as we will see in the proof of Theorem 4. We solve it partially by determining the advice complexity for the Delayed H-Node-Deletion Problem, where H can be disconnected.

We occasionally speak of "deleting a graph" in this section. By this we refer to the removal of nodes such that a given substructure is no longer induced. If not specified further, assume that the minimum number of nodes is removed.

Definition 3 Let $C_G = \{C_1, C_2, \dots, C_j\}$ 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.

We are restricting the classical notion of a *packing* to only include graphs in the same packing if they are not connected by an edge as follows.

Definition 4 Given a graph G. For a connected graph C we define the packing $p_C(G)$ of C in G as the family of sets of pairwise node-disjoint copies of C in G and the packing number of C in G, $\nu_C(G)$, as $\max_{H \in p_C(G)}(|H|)$. We further demand that there is no edge between two graphs of such a set.

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

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

In other words: We transform a component C in such a way that a single node deletion is not sufficient to remove C from the transformed graph, while not introducing additional copies of C in the process.

Definition 5 Let $\varphi_1 : \mathcal{G} \to \mathcal{G}$, $\varphi_2 : \mathcal{G} \to \mathcal{G}$ be isomorphisms. We call the graph H' a redundancy construction of a connected graph H with $|H| > 1$ if:

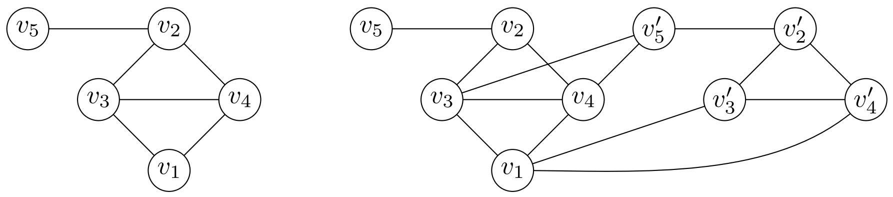
- For all $v \in V(H')$ there exists a φ_1 with $\varphi_1(H) \trianglelefteq H' - v$
- For all φ_1: $\varphi_1(H) \ntrianglelefteq H' - V(\varphi_2(H))$ if $V(\varphi_2(H)) \subseteq V(H')$

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

Definition 6 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 $E(H') = E(H) \cup$ $\mid (v_i, v_j) \in E(H), v'_i, v'_j \in V(H')\} \cup \{(v_1, v'_i) \mid (v_1, v_i) \in E(H)\} \cup \{(v_k, v'_j) \mid (v_1, v_j) \in E(H)\}$.

Intuitively, we create a copy of H except for a single arbitrary node v_1. The copied neighbors of v_1 (v'_3 and v'_4 in Example 2) are then connected with v_1. Lastly, some copied node is chosen and connected with the original neighbors of v_1 (v'_5 in Example 2).

Example 2 A graph H and its redundancy construction H':



Lemma 5 *The transformation in Definition 6 is a redundancy construction.*

Proof Given any connected graph H with $|H| > 1$. Let H' be the graph we obtain from Definition 6. If any single node $v \in V(H'), v \neq v_1$, is removed from H', there remains a copy of H in $H' \setminus \{v\}$, as we copied H except for v_1 and the graphs were joined together at v_1. If on the other hand we remove v_1, the node v_k that is used in Definition 6 acts as a substitution for v_1, as by construction, $N(v_k) \supseteq N(v_1)$.

The total number of nodes is $2 \cdot |H| - 1$ which means that removing any copy of H from H' results in less than $|H|$ nodes.

Thus, both properties of a redundancy construction hold for H'. □

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

4.1 Lower Bound

The lower bound uses two building blocks: Selecting a correct node for deletion in each component as in the proof of Theorem 3 as well as selecting the correct copies of a component by using redundancy constructions.

Theorem 4 *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 $opt_H(G) \cdot \log |V(C_{max})| + (v_{C_{max}}(H) - 1) \cdot \log opt_H(G)$ many advice bits on input G.*

Proof Let $C_H = \{C_1, \dots, C_j\}$ and $|V(C_1)| \leq \dots \leq |V(C_j)|$. The adversary first presents $k \geq \max\{\, v_{C_i}(H) \mid C_i \in C_H \,\}$ 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. W.l.o.g. we assume that any algorithm does not delete any nodes before an H is completed and then only deletes nodes of any H in the graph until no H is any longer induced in the graph.

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 $v_{C_i}(H) - 1$ 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 $\max_{C_i \in C_H}\{v_{C_i}(H)\} - v_{C_{max}}(H) + 1$ copies of each $C_i \in C_H \setminus C_{max}$ are presented. Deleting all C_{max} except for $v_{C_{max}}(H) - 1$ occurrences will thus only need $k - v_{C_{max}}(H) + 1$ deletions, while deleting any other component will need at least $k - \max_{C_i \in C_H}\{v_{C_i}(H)\} + 1 + \max_{C_i \in C_H}\{v_{C_i}(H)\} - v_{C_{max}}(H) + 1 = k - v_{C_{max}}(H) + 2$ 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 6 is used in order to *repair* an arbitrary set of $v_{C_{max}}(H) - 1$ copies of C_{max}. Every optimal algorithm will leave exactly $v_{C_{max}}(H) - 1$ copies of C_{max} after G is completely revealed. There are $\binom{opt_H(G)+v_{C_{max}}(H)-1}{opt_H(G)}$ 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 $v_{C_{max}}(H) - 1$ 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. It has $|V(C_{max})|$ ways to do so for each copy of C_{max}.

We now measure how much advice an algorithm needs at least. First of all, it is 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 $v_{C_{max}}(H) > 1$, 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 $\binom{opt_H(G)+v_{C_{max}}(H)-1}{opt_H(G)}$ 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 in the size of at least $\log\binom{opt_H(G)+v_{C_{max}}(H)-1}{opt_H(G)} \geq (v_{C_{max}}(H) - 1) \cdot \log(opt_H(G))$ advice bits. $\square$

4.2 Upper Bound

For simplicity of writing down the algorithm, we will assume in this section that we are only ever presented instances in which the graph induces 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. Additionally, the algorithm only looks at a single C_o to focus on for deletion. In most cases except for some corner cases, this is optimal. In the general case, the adversary gives the algorithm a list as in line 5 for each single component of H. This list is empty for all components except for those from which we need to delete nodes. The algorithm then labels the components in the same way as C_o in lines 12–16 and cycles through all lists for deletion. Our arguments for the case of a single C_o can be easily generalized for this case of multiple components. We call this complete version of the algorithm the *extended version*.

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

```
1: Input: Online graph $G$ with $V(G) = \{v_1, \dots, v_n\}$, $H$
2: Advisor computes $C_{o\nu} \in \arg\min_{C \in C_H} \{\nu_C(G) - \nu_C(H)\}$
3: Advisor computes $C_o \in \arg\min_{C \in C_{o\nu}} \{|V(C)|\}$
4: Advisor computes $L$, the list of labels marked for keeping
5: Read advice: $C_o$                                    ▷ Which $C_o \in C_H$ to delete
6: Read advice: List $L$ of numbers in range $[1, opt_H(G) + O(1)]$
7: $k \leftarrow 1$
8: Define $l\colon \mathcal{G} \to \mathbf{N}$, $l(G) = 0$, $D = \emptyset$
9: Define $labeled\colon \mathcal{G} \to \{0,1\}$, $labeled(G) = 1$ iff $l(G) \neq 0$, otherwise $labeled(G) = 0$
10: for all $i \in \{1, \dots, n\}$ do
11:     $G_i \leftarrow G[\{v_1, \dots, v_i\} \setminus D]$                ▷ Reveal next node
12:     if $\nu_{C_o}(G_i) \geq \nu_{C_o}(H)$ then
13:         $W \leftarrow \arg\max_{P \in p_{C_o}(G_i)} |P|$             ▷ Biggest Packings
14:         $\mathcal{H} \leftarrow \arg\max_{P \in W} \sum_{g \in P} labeled(g)$      ▷ Most labels
15:         Select $P \in \mathcal{H}$                              ▷ Arbitrary set
16:         for all $C \in P$ do                        ▷ Label everything unlabeled
17:             if $l(C) = 0$ then $l(C) \leftarrow k$; $k \leftarrow k + 1$
18:         $S \leftarrow \{C \in P \mid l(C) \notin L\}$      ▷ Select everything not marked for keeping
19:         for all $C \in S$ do
20:             Read advice: Which $v \in V(C)$ to delete
21:             $D = D \cup \{v\}$                            ▷ Delete $v$ out of $G_i$
```

For an instance with an online graph G with $V(G) = \{v_1, \dots, v_n\}$ and a forbidden subgraph H, the advisor first computes the advice the algorithm is going to read during its run. It first runs an optimal offline algorithm on G and determines which is the component that is focussed on for deletion, named C_o from here on. Finally, the advisor computes a list L of labels by simulating the online algorithm. These labels will thus coincide with the labels given by the algorithm to copies of C_o which are not to be deleted in an optimal solution. As there are at most $\nu_{C_o}(H) \cdot opt_H(G)$ disjoint copies of H in G and as Lemma 8 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 used for every deletion of a concrete node in each copy of C_o.

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

Whenever the next node x_i is revealed that fulfills $H \trianglelefteq_\varphi G_i$, 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 packings of C_o. Of them it identifies a set P of which the most components have already received a label. Then all previously unlabeled copies of $C_o \in P$ receive a new unique label. The algorithm now looks at the label list L given by the advisor. Every copy of $C_o \in P$ whose label is not in L is now marked for deletion. The algorithm reads advice about which concrete node out of every copy of C_o is optimal to delete.

Lemma 6 *The extended version of Algorithm* 1 *is correct.*

Proof The algorithm is correct iff whenever $H \trianglelefteq_\varphi G_i$ holds, a set of nodes $S \subseteq V(G_i)$ is deleted such that $G_i - S$ is H-free. The condition of the if-branch in line 11 triggers when there already is an isomorphic copy of H in the graph. A largest packing of copies of a C_o from H is then chosen in G in line 12, with the set including the most labeled components being chosen if the choice is ambiguous. All members of this set are then labeled in lines 15 and 16. $|P| - \nu_{C_o}(H)$ copies need to be deleted from G_i in order to make the graph H-free. In line 17 the algorithm reads from L which of at most $\nu_{C_o}(H)$ copies of H are not to be deleted and deletes all other copies. If the graph is not yet H-free, the algorithm repeats this process with the next component that has a non-empty list of labels, making the graph ultimately H-free at the end of line 20, i.e. before the next node is revealed. □

Lemma 7 *The extended version of Algorithm* 1 *is optimal.*

Proof We already know by Lemma 6 that Algorithm 1 is correct and that our algorithm only deletes nodes from types of components that an optimal algorithm would delete from as well. Thus, we only have to show that each node deleted by the algorithm is part of the solution of an optimal offline algorithm.

Our algorithm determines a node for deletion by reading advice telling it from which labeled component it should delete a node. It then also reads advice which node of the selected copy it should delete. Thus, it can only not simulate an optimal offline algorithm if no set of labeled components covers a component such that it is the only optimal extension of the algorithms current solution. This component then does not have a label. By definition, it shares at least one node with a labeled component of the same type from which we do delete a node in this step or it is connected with by an edge with a labeled component.

Each time $H \trianglelefteq_\varphi G_i$ holds (especially the first time), we cover a complete copy of H by our labeled components. If we assume that none of the covered components was optimal for deletion, a supposedly optimal offline algorithm would leave all of these components in the graph after doing some other deletions. Thus, $H \trianglelefteq_\varphi G_i$ would still hold. This means that any optimal offline algorithm has to delete at least one of the labeled components. We can communicate which of these components our algorithm should delete by advice.

Thus, Algorithm 1 simulates an optimal offline algorithm. □

Definition 7 Given graphs G, H and a labeling function $l : \mathcal{G} \to \mathbf{N}$. We call a family $\mathcal{C}$ of induced subgraphs of G a *configuration* if every element of $\mathcal{C}$ is isomorphic to H, $\mathcal{C}$ is a packing and $l(C) \neq 0$ for each $C \in \mathcal{C}$. 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.

The following lemma refers to the simple version of the algorithm but can be easily generalized, as each component type is labeled independently of all others. This is discussed briefly after the proof.

Lemma 8 *Given an online graph G, a forbidden graph H, as well as a graph $C \in C_H$ (as in line* 4 *of the algorithm) of which there may be at most $k = \nu_C(H) - 1$ disjoint copies present in G. Algorithm* 1 *assigns no more than $\nu_C(H) \cdot opt_H(G) + O(1)$ labels to G.*

Proof In the worst case, whenever $H \trianglelefteq_\varphi G_i$ holds, the biggest configuration that we can find does not contain a single labeled component, thus the algorithm labels every member of the configuration. □

It is possible that our algorithm labels more than one type of component, as they may pairwise overlap and the connecting node could be the optimal one to delete. Thus, we can bound the total number of labels over all components by $opt_H(G) \cdot |C_H|$
.

Theorem 5 *Let H be a graph and C_{max} be a component of H of maximum size and G be an online graph. The* Delayed H-Node-Deletion Problem *can be solved optimally using at most $opt_H(G) \cdot (\log |V(C_{max})| + |C_H|) + O(\log opt_H(G))$ advice bits.*

Proof We count the number of advice bits used by Algorithm 1. We know by Lemmata 6 and 7 that it is correct and optimal. The advice in line 4 is of constant size. As each L only contains the labels for components which are not to be deleted and we limited the number all labels by $opt_H(G) \cdot |C_H|$ in Lemma 8, $opt_H(G) \cdot |C_H| + O(\log opt_H(G))$ advice bits—using self-delimiting encoding to encode $opt_H(G)$—are needed in line 5.

Finally, the algorithm reads advice on which node of each copy of C_{max} that is part of $sol_H(G)$ to delete in line 20. This can be done using $opt_H(G) \cdot \log |V(C_{max})|$ advice bits in total. □

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

The problem of deleting edges from a graph needs a separate approach from that of deleting nodes. There is a simple example that highlights a problem which makes it hard to simply adapt the ideas of node deletion to the task of edge-deletion: Let

$\mathcal{F} = \{P_3, K_3\}$, i.e. a path with two edges and a triangle. Obviously, $P_3 \ntrianglelefteq_\varphi K_3$, but deleting any single edge from the K_3 produces a P_3. This is a problem that does not occur in the case of node deletions, as the graphs are induced by their nodes. In this section we present a way to calculate the advice complexity for each Delayed Connected $\mathcal{F}$-Edge-Deletion Problem directly from $\mathcal{F}$. Since our tight bounds on the advice complexity are not trivial to calculate for a concrete problem instance, we give a separate analysis for the case that only a single graph is forbidden in the following section, which results in an almost tight bound that is a simple function of the size of the number of edges of this forbidden graph.

In contrast to Sect. 4, "deleting a graph" means removing all of its nodes.

Definition 8 Let $\mathcal{F}$ be a family 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 9 *Let $\mathcal{F} = \{H_1, \dots, H_k\}$ be a family 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'.*

Proof As D is H_i-good for G there must be some $\bar{D} \supseteq D$ that is optimal for G by the definition of goodness. Let us construct the graph $G' = G - (\bar{D} - D)$, i.e., we get G' from G by removing all edges that are in $\bar{D}$, but not in D. Let us assume that D is not optimal for G'. Then there must be an optimal D' for G' with $|D'| < |D|$. Then, however, $G - ((\bar{D} - D) \cup D') = G' - D'$ is also $\mathcal{F}$-free by construction, which is impossible because $(\bar{D} - D) \cup D'$ is smaller than the already optimal $\bar{D}$. Hence, D is optimal for G. □

5.1 Upper Bound

An important tool that we will use in the analysis of the number of advice bits is the solution of a special recurrence relation: Let $(d_1, \dots, d_k) \in \mathbf{N}^k$. Let $m(n)$ be the solution to the recurrence relation with

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

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

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

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 root of the characteristic polynomial with multiplicity k [15]. However, in (1) 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 [12, 16]. Therefore $m(n) = \Theta(\beta(S)^n)$.

Theorem 6 *Let $\mathcal{F} = \{H_1, \dots, H_k\}$ be a family of connected graphs and let S_i be H_i-sound and H_i-sufficient for all $i \in \{1, \dots, k\}$. 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 $i \in \{1, \dots, k\}$.*

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 string is 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 $m(opt_{\mathcal{F}}(G)) = \max_i\left(\sum_{D \in S_i} m(opt_{\mathcal{F}}(G) - |D|)\right)$ if $opt_F(G)$ is at least as big as every $D \in S_i$. Standard techniques show that $m(opt_{\mathcal{F}}(G)) = O(\max\{\beta(S_1), \dots, \beta(S_k)\}^{opt_{\mathcal{F}}(G)})$. □

In Fig. 1 you can find the behavior of an algorithm that solves the Delayed Connected $\mathcal{F}$-Edge-Deletion Problem with $S_1 = \{D_1, D_2, D_3\}$ and $|D_1| = 2$, $|D_2| = |D_3| = 1$ when it encounters only the forbidden H_1 as a tree of possible different computation paths. In the tree nodes you find the number of edges that will still be deleted. This corresponds to the recurrence $m(n) = m(n-2) + m(n-1) + m(n-1)$ for $n \geq 2$ and $m(0) = 1$, $m(1) = 2$. Then $m(5) = 70$ and $m(n) = \Theta((\sqrt{2}+1)^n)$ because $\beta(2,1,1) = \sqrt{2}+1$. The exact solution is $m(n) = \frac{1}{4}(\sqrt{2}+1)^n(\sqrt{2}+2) - \frac{1}{4}(1-\sqrt{2})^n(\sqrt{2}-2)$.

5.2 Lower Bound

Let $\mathcal{F} = \{H_1, \ldots, H_k\}$ be a family of connected graphs. Let A be a correct algorithm 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 $G[X] \cong H_i$, $\varphi : V(H) \to X$, 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. Figure 2 shows an example. Let U be the vertices on top and V on the bottom. The vertices in U are ordered according to their label and drawn from left to right in ascending order in the figure. The matching should have the following properties. Let U' be the nodes in the matching on top and V' on the bottom. In the example $U' = \{1, 3, 4, 6\}$ and V' are the nodes marked in gray.

The first property is $N(U') = V$, i.e., every node in V is connected to at least one node in U'. To check that this property is fulfilled in Fig. 2 you have to check that every node on the bottom is connected to one 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 $N(v)$. You can check the third property in the figure easily: Just verify that the matching edge is the leftmost emerging edge of each $v \in V'$. For example m has two neighbors 6 and 7. The smaller one is its partner in the matching and hence in U'.

Lemma 10 *Let $G = (U, V, E)$ be a bipartite graph where $U = \{u_1, \ldots, u_k\}$. Let $\leq$ be a reflexive and transitive relation 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* $v \in V'$.

Proof We claim that Algorithm 2 computes sets U' and V' that fulfill the three properties stated in the lemma. As $G[U' \cup V']$ is an induced matching in G, the matching can be found easily from U' and V'.

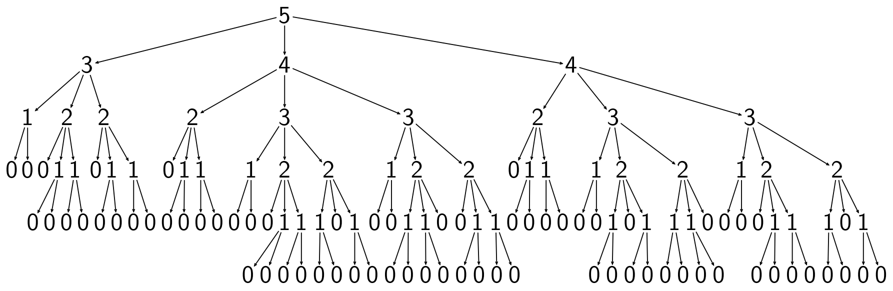


Fig. 1 Possible ways to delete H_1's

Algorithm 2 Algorithm for Finding a Matching from Lemma 10

```
1: $U' \leftarrow \{ \min N(v) \mid v \in V \}$
2: $V' \leftarrow \emptyset$
3: for $u \in U'$ in descending order do
4:     if there is a $v \in V$ such that $N(v) \cap U' = \{u\}$ then
5:         $V' \leftarrow V' \cup \{v\}$                ▷ pick an arbitrary one
6:     else
7:         $U' \leftarrow U' \setminus \{u\}$
```

We prove all three properties separately.

1. "$N(U') = V$": The precondition of the lemma states that $N(U) = V$ and therefore that $N(v) \neq \emptyset$ for every $v \in V$. In line 1 we add to U' a neighbor of each $v \in V$, which already guarantees $N(U') = V$.

   We have to prove that the invariant $N(U') = V$ is maintained in the **for**-loop. Only in line 7 a node is removed from U', which could destroy the property $N(U') = V$. However, we remove u only if there is no $v \in V$ for which u is the only neighbor. Hence, every v retains at least one other neighbor in U'.

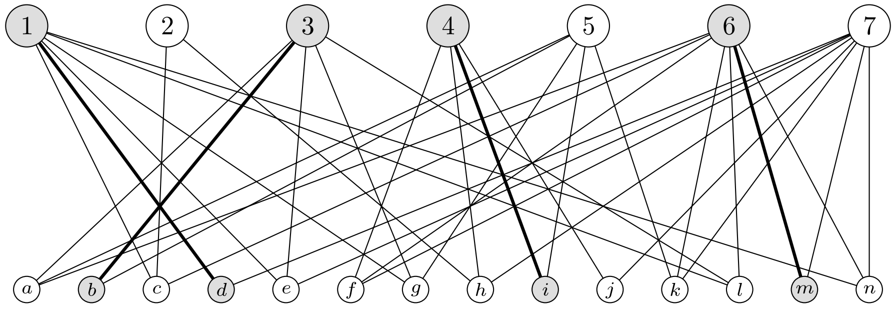


Fig. 2 An example for the algorithm in the proof of Lemma 10. The sets U' (top, ordered from left to right), V' (bottom) and the matching are highlighted

2. "$G[U' \cup V']$ is a matching": If in the end $u \in U'$, then u was *not* removed from U' in line 7, which means that line 5 was executed and a v added to V' with $N(v) \cap U' = \{u\}$. At this point of time v has only one neighbor in U'. Afterwards U' is only shrinking, so v cannot have more than one neighbor in U' at the end. Also, afterwards u cannot be deleted from U' as each u is only considered once in the body of the **for**-loop. This means that at the end each $v \in V'$ has exactly one neighbor in U'. There cannot be a $u \in U'$ with no neighbor in V' because it would have been removed in line 7. Hence, $U' \cup V'$ induces a matching.
3. "$\min N(v) \in U'$ for every $v \in V'$": Directly after line 2 the statement is obviously true.

   Let us assume that this property does not hold at any later point. Then there is a $v \in V'$ and $\min N(v) = u$, where $u \notin U'$. For this to hold, there must be a different $u' \in U', u' \in N(v)$ that is matched to v because $U' \cup V'$ induces a matching. After line 1 was executed, U' thus contained u and u'. Because $u \leq u'$ and in line 3 the vertices in U' are visited in descending order, u' was visited *before u*.

   We are now looking at the moment when u' was visited in line 3. Then U' still contained both $u, u' \in U'$ and $u, u' \in N(v)$. u' cannot be matched with v at this point, as $|N(v) \cap U'| > 1$. This means that after u' was considered in the **if**-condition in line 4, either u' was removed from U' or u' was matched with a node from $V \setminus v$, both leading to a contradiction.

□

Lemma 11 *Let $\mathcal{F} = \{H_1, \ldots, H_k\}$ be a family 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 $|D| \geq |D'|$.

Proof We will use Lemma 10. In the following we look at a fixed i and write H for H_i, S for S_i, and S' for S'_i.

If $H \trianglelefteq G_1$ and $H \trianglelefteq G_2$, then we define that $G_1 \equiv_S G_2$ iff for all $D \in S$ it holds that D is H-good for G_1 if and only if D is H-good for G_2. There are only $2^{|S|}$ many possibilities which $D \in S$ are H-good for some G and consequently the equivalence relation $\equiv_S$ has at most $2^{|S|}$ many equivalence classes. Assume that $R = \{G_1, \ldots, G_m\}$ is a set of representatives of all equivalence classes.

We build a bipartite graph (S, R, E) where there is an edge between $D \in S$ and $G_i \in R$ iff D is H-good for G_i. Furthermore we define a reflexive and transitive relation on S by defining $D \leq D'$ iff $|D| \leq |D'|$. By Lemma 10 there is a matching between S and R that fulfills all three conditions that are stated there. In particular, we can determine the set $S' \subseteq S$ that corresponds to U' in the lemma.

We prove that S' then also fulfills the conditions stated in this lemma:

1. S' is H-sound because it is a subset of S, which is H-sound itself.

2. S' is H-sufficient because by Lemma 10 we know that $N(S') = R$ in the bipartite graph. That means that there is an edge from every G_i to some $D \in S'$, which means there is some $D \in S'$ such that D is H-good for G_i.
3. S' is H-minimal, because for every $D \in S'$, its matched graph is covered exactly by D, as $G[U' \cup V']$ is an induced matching.
4. Lemma 10 states that $G[S' \cup R]$ is a matching. By the H_i-minimality of S', for every $D' \in S'$ there is a graph G such that only D' is H_i-good for G and no other member of S'. This is exactly the graph $G_i \in R$ matched with D', as $G[S' \cup R]$ is an induced matching. By condition 3 of Lemma 10, the cardinally smallest neighbor of G_i is D', thus there cannot be a $D \in S$ that is smaller than D'. □

Theorem 7 *Let $\mathcal{F} = \{H_1, \dots, H_k\}$ be a family 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 $i \in \{1, \dots, k\}$.*

Proof By Lemma 11 there is an $S_i' = \{D_1, \dots, D_r\} \subseteq S_i$ 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 in several rounds until the size of the optimum solution for the presented graph exceeds $l - \max\{|D_1|, \dots, |D_r|\}$.

1. The adversary presents a disjoint copy of H_i.
2. Then the adversary computes G_j with $H_i \trianglelefteq G_j$ 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_j's 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 $l - \max\{|D_1|, \dots, |D_r|\}$ 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:

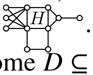
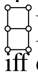
$$N(l) = \begin{cases} \sum_{j=1}^{r} N(l - |D_j|) & \text{if } l \geq \max\{|D_1|, \dots, |D_r|\} \\ 1 & \text{otherwise} \end{cases}$$

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 $G'_1, \dots, G'_r$, it deletes different edge sets for each of the r possibilities.

The adversary constructed an instance that consists of a sequence of disjoint graphs $G'_{i_1}, \dots, G'_{i_t}$ from the set $\{G'_1, \dots, G'_r\}$ of which the total size is at least $\sum_{j=1}^{t} opt_{\mathcal{F}}(G_{i_j}) - \max\{|D_1|, \dots, |D_r|\}$ 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 $\log N(l) = l \cdot \log \beta(S'_i) + O(1) = opt_{\mathcal{F}}(G) \cdot \log \beta(S'_i) + O(1)$ 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 $i \in \{1, \dots, k\}$ because $opt_{\mathcal{F}}(G)$ can be become arbitrarily big. □

Lemma 12 *Let $\mathcal{F}$ be a family 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 9 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 of them $\mathcal{F}$-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 many possibilities for $\Upsilon(G)$ and we can enumerate all of them. Let us say this enumeration is $\Upsilon_1, \dots, \Upsilon_t$. 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 some $D \in S$ is H-good for G. Iff these tests pass for all i then S is indeed H-sufficient. □

Theorem 8 *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 6. "$\geq$" by Theorem 7. 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 12). Then $\beta(S)$ is computed by finding the only real root of the characteristic polynomial of the corresponding recurrence relations [15]. □

We will now give an example on how to apply this theorem to the class of trivially perfect graphs. For demonstrative purposes we will not use the algorithm described in Lemma 12, but manually construct the sets S_i, such that the reader may get a better intuition why the constructed sets are indeed H_i-sound and H_i-sufficient.

Example 3 Given $\mathcal{F} = \{$, $\}$. In order to be able to apply Theorem 8, we need to compute sets $S_i \subseteq 2^{E(H_i)}$ that are H_i-sound and H_i-sufficient. We start with $H_1 =$.
A single edge cannot be in S_1, as this would violate the demanded H_1-soundness by leaving a P_4 in the graph. S_1 may not only consist of edge subsets of size three or larger, as this would violate the H_1-sufficiency for the graph . Thus, S_1 contains at least one two-element subset of $E(H_1)$. Indeed, every subset including exactly two edges is part of S_1, as the following construction shows, which is also visualized in Figure 3. By constructing an input graph which extends one of the nodes of the cycle to a P_3, each pair of neighboring edges is exactly the optimal solution. On the other hand, if we attach an edge to each of two neighboring nodes of the cycle, we need to delete exactly two opposing edges of . As we cover each two-element edge subset of , we do not need to look at edge subsets bigger than size two (even if including them would not necessarily violate our two conditions, but increase the size of $\beta(S_1)$).

We continue by computing S_2 for $H_2 =$, which is a bit simpler. S_2 may not only consist of edge subsets of size two or larger, as this would violate the H_2-sufficiency for the graph . On the other hand, every one-edge subset of $E(H_2)$ is part of S_2. The outer edges of the path are optimal for deletion if we attach a P_2 to either end of the original P_4. The middle edge is optimal for deletion if we attach an edge to both ends of the path. Again, as we cover every single-edge subset, we do not need to look at bigger subsets for S_2.

Thus, S_1 consists of six sets of size two and S_2 of three sets of size one. We obtain $\beta(S_1) = \beta(2,2,2,2,2,2) < 2.5$ and $\beta(S_2) = \beta(1,1,1) = 3$. Thus, by Theorem 8 the advice complexity is $\log 3 \cdot opt_{\mathcal{F}}(G) + O(1)$.

6 The Delayed *H*-Edge-Deletion Problem and Further Edge-Deletion Problems

As announced in the previous section, we now deal with the simpler case of forbidding only a single connected graph. For this, we give the following definition, which allows us to glue two graphs together at an edge.

Definition 9 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, $G\,{}_{xy}{\ominus}_{uv}\,H$ 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 4 shows an example.

A crucial difference between these two ways of gluing is that basically "nothing can go wrong" when gluing 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 gluing 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 10 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

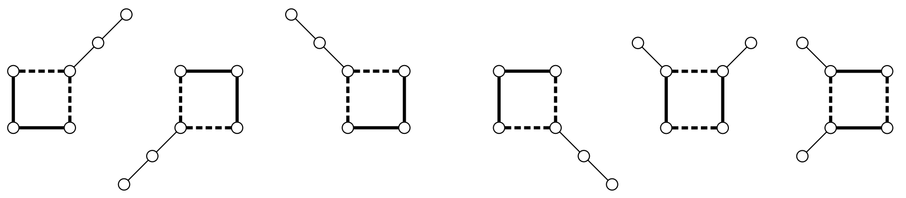


Fig. 3 In Example 3 the set S_1 contains six edge sets shown here as dashed lines. The corresponding supergraphs of are depicted demonstrating that all six sets in S_1 are necessary

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 $\mathcal{F} = \{P_3, K_3\}$. Because of symmetry we have to look only at three different gluing operations: P_3 to P_3, K_3 to K_3, and P_3 to K_3. Whenever K_3 is involved, the resulting graph contains P_3 as an induced subgraph. For example, gluing K_3 to itself results in a cycle of length four, which is isomorphic to $K_3 \ominus_e K_3$. This means that the edges in K_3 are not critical. Gluing P_3 to itself can be done in two ways. The first result are two disjoint edges and the second is again P_3. Hence, the edges in P_3 are critical. In total, $\#crit(\mathcal{F}) = 2$.

Lemma 13 *Let G be a 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 it is also a cut vertex in $G \ominus_e G$. 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 14 *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 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 proof of the next lemma is surprisingly complicated. 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 similar way of how Theorem 3 works for node-deletion problems.

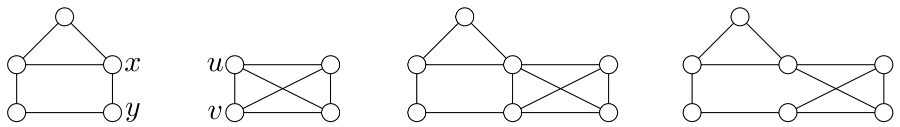


Fig. 4 From left to right: G, H, $G\ _{xy}\!\oplus_{uv} H$, $G\ _{xy}\!\ominus_{uv} H$

Lemma 15 *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 counter-example with respect to taking subgraphs. If $u \in V(G)$ then let u_1 and u_2 be the respective copies of u in both parts of $G \ominus_e G$. In particular $x_1 = x_2$, $y_1 = y_2$ and $u_1 \neq u_2$ if $u \notin \{x, y\}$.

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. Because 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 $G \ominus_e G$).

In the following we establish some notation.

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. 5 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:

$$H' \ominus_{xy} H' \supseteq H \simeq G \supseteq H' \tag{2}$$

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 end 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. 6). By Lemma 14 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 (2)

If we look at the preconditions of Lemma 15 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, $H' = G$.

Because $H \simeq G$ 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 higher than in H'.

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 15 is wrong. □

The condition that G is two-connected in Lemma 15 is necessary. Figure 7 shows that gluing 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 15. 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 16 *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 15. 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 15 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: Because 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:

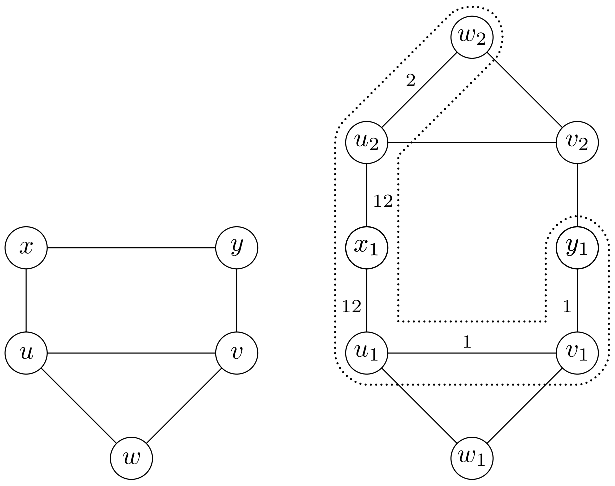


Fig. 5 A graph G is depicted left. The corresponding graph $G \ominus_{xy} G$ is on the right (here $x_1 = x_2$ and $y_1 = y_2$). Note that the edge corresponding to xy is missing in the right graph. An subgraph H is indicated by the dotted outline. (Please note that H is not isomorphic to G, which would be impossible by the very lemma we are about to prove.)

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

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 two 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 Lemmas 15 and 16 have several consequences. First, it means that $\#crit(\{G\}) = ||G||$. It is also the key to the next theorem, which gives an almost tight bound for the Delayed Connected H-Edge-Deletion Problem, and is much simpler than using the techniques from Sect. 5.

Theorem 9 *Let H be a connected graph and G be an arbitrary graph. Every online algorithm requires $\lceil opt_H(G) \cdot \log(||H||) \rceil$ advice bits to solve the Delayed H-Edge-Deletion Problem optimally on input G. There is a deterministic algorithm that solves this problem using $\lceil opt_H(G) \cdot \log ||H|| \rceil + \log opt_H(G) + 2 \log\log opt_H(G)$ advice bits.*

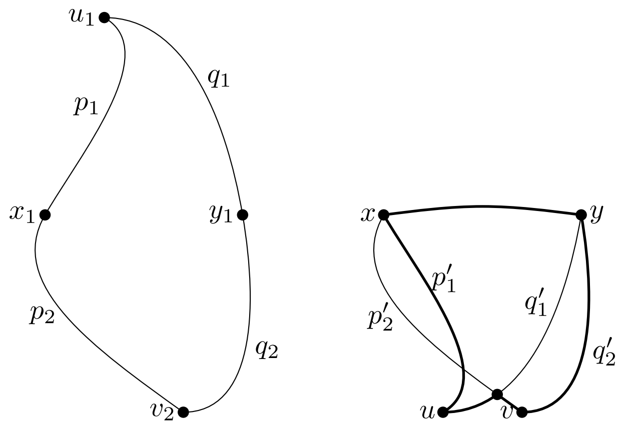


Fig. 6 u_1 and v_2 are two-connected in H (left). The resulting vertices u and v in H' are then also two-connected (right)

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 another by the adversary. Each component is some H_i.

It is clear that making H_i H-free requires removing of at least one edge and can be accomplished by removing exactly the edge e_i by Lemma 15: 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 $\lceil k \cdot \log(||H||)\rceil$ 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.

For an upper bound, 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 $\|H\|$. As only k times an edge is selected, k such numbers suffice and we can encode them using $\lceil k \cdot \log ||H||\rceil$ bits. Assuming k is not a power of 2, we communicate its size using self-delimiting encoding using an additional $\log k + 2 \log\log k$ bits. $\square$

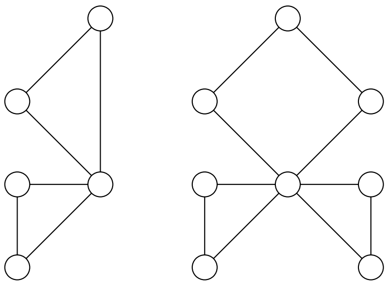


Fig. 7 Lemma 15 does not hold for single-connected graphs

7 Further Discussion of General $\mathcal{F}$-Edge-Deletion Problems

The following theorem generalizes the lower bound from Theorem 9 to arbitrary families $\mathcal{F}$. 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 bigger disadvantage is the missing matching upper bound for the general case, which we discuss after stating and proving the theorem.

Theorem 10 *Let $\mathcal{F}$ be a set of graphs and G be an arbitrary graph. Solving the online* Delayed $\mathcal{F}$-Edge-Deletion Problem *requires at least* $\lceil opt_{\mathcal{F}}(G) \cdot \log(\#crit(\mathcal{F}))\rceil$ *advice bits on input G.*

Proof The proof is very similar to the proof of Theorem 9. 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. $\square$

It would be nice to have a matching upper bound for Theorem 10, too, but it is easy to see that Theorem 10 is not always optimal. For example, consider claw- and diamond-free graphs. The edges in the diamond are not critical and Theorem 10 gives us only $opt \cdot \log(3)$ as a lower bound on the advice complexity. However, you can find five different supergraphs that induce a 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 $\mathcal{F} = \{P_3, K_3\}$. Remember that P_3 contains critical and K_3 non-critical edges. Hence, as $P_3 \subseteq K_3$ 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 10.

Unfortunately, the algorithms does not work in all cases and indeed such an algorithm cannot exist because the lower bound in Theorem 10 is not optimal. Figure 8 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 $\mathcal{F}$. While we are able to provide tight bounds for the family $\mathcal{F}$ of Fig. 8 using Theorem 8, we leave open the problem of finding an upper bound and a better lower bound for the general Delayed $\mathcal{F}$-Edge-Deletion Problem as well as for the general Delayed $\mathcal{F}$-Node-Deletion Problem.

The last example we consider shows a problem whose advice complexity is not bounded by any 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 Delayed 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 a function of the size of the optimal solution.

Theorem 11 *Given a set $\mathcal{F}$ of all distance hereditary graphs and an arbitrary graph G as an input graph. The Delayed DH-Edge-Deletion Problem requires at least $opt_{\mathcal{F}}(G) \cdot (\log(n-1) - 2)$ advice bits.*

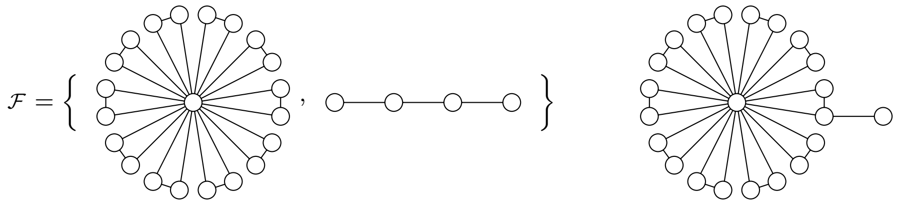


Fig. 8 A counterexample to the optimality of Theorem 10. The adversary presents the big graph in $\mathcal{F}$. The algorithm has to delete an edge e. Then the adversary adds another vertex as shown on the right side. The optimal solution is to delete two edges. The algorithm is not optimal if it did not choose e from the correct petal out of ten. Repeating this scheme for k rounds leads to $opt = 2k$ and at least $\log(10^k) = \log(10)/2 \cdot opt > 1.66 opt$ advice bits to achieve optimality. Theorem 10 provides a lower bound of only $\lceil opt \cdot \log 3 \rceil \leq \lceil 1.59 opt \rceil$

Proof We construct k graphs of size $4t + 1$ depicted in Fig. 9 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 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 another, the adversary can choose between t^k instances in total. The optimal solution deletes only $k = opt$ edges. The online algorithm therefore requires $k \log t$ advice bits. The graph has size $n = 4t + 1$, so $t = (n - 1)/4$, making the number of advice bits at least $k \log((n - 1)/4) = k(\log(n - 1) - 2)$. □

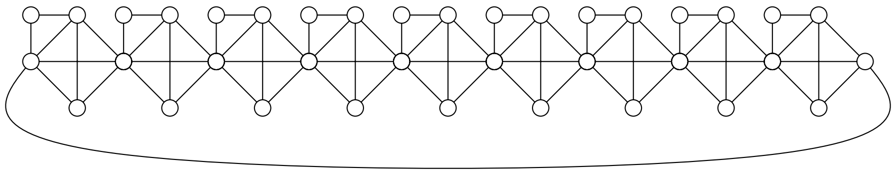


Fig. 9 Gadget for DH-edge deletion

Acknowledgements We like to thank Rastislav Královič for helping significantly to simplify the proof of Lemma 8 and Niklas Berndt for pointing out an error in the proof of Theorem 9. We also thank Walter Unger and Janosch Fuchs for stimulating discussion about the proof of Lemma 15. Finally, we want to thank the anonymous reviewers for their detailed and very constructive feedback.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

1. Bandelt, H., Mulder, H.M.: Distance-hereditary graphs. J. Comb. Theory, Ser. B **41**(2), 182–208 (1986). https://doi.org/10.1016/0095-8956(86)90043-2
2. Böckenhauer, H., Hromkovič, J., Komm, D., Královič, R., Rossmanith, P.: On the power of randomness versus advice in online computation. In: H. Bordihn, M. Kutrib, B. Truthe (eds.) Languages Alive—Essays Dedicated to Jürgen Dassow on the Occasion of His 65th Birthday, Lecture Notes in Computer Science, vol. 7300, pp. 30–43. Springer (2012). https://doi.org/10.1007/978-3-642-31644-9_2
3. Böckenhauer, H., Komm, D., Královič, R., Královič, R., Mömke, T.: On the advice complexity of online problems. In: Algorithms and Computation, 20th International Symposium, ISAAC 2009, Honolulu, Hawaii, USA, December 16–18, 2009. Proceedings, pp. 331–340 (2009). https://doi.org/10.1007/978-3-642-10631-6_35
4. Böckenhauer, H., Komm, D., Královič, R., Královič, 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
5. Böckenhauer, H., Komm, D., Královič, R., Rossmanith, P.: On the advice complexity of the knapsack problem. In: D. Fernández-Baca (ed.) LATIN 2012: Theoretical Informatics—10th Latin American Symposium, Arequipa, Peru, April 16–20, 2012. Proceedings, Lecture Notes in Computer Science, vol. 7256, pp. 61–72. Springer (2012). https://doi.org/10.1007/978-3-642-29344-3_6
6. Böckenhauer, H., Komm, D., Královič, R., Rossmanith, P.: The online knapsack problem: advice and randomization. Theor. Comput. Sci. **527**, 61–72 (2014). https://doi.org/10.1016/j.tcs.2014.01.027
7. Böckenhauer, H.J., Komm, D., Královič, R., Královič, R.: On the advice complexity of the k-server problem. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) Automata, Languages and Programming, pp. 207–218. Springer, Berlin, Heidelberg (2011)
8. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press (1998)
9. Cai, L.: Fixed-parameter tractability of graph modification problems for hereditary properties. Inf. Process. Lett. **58**, 171–176 (1996)
10. Chen, L., Hung, L., Lotze, H., Rossmanith, P.: Further results on online node- and edge-deletion problems with advice. In: L. Gasieniec, R. Klasing, T. Radzik (eds.) Combinatorial Algorithms—31st International Workshop, IWOCA 2020, Bordeaux, France, June 8–10, 2020, Proceedings, Lecture Notes in Computer Science, vol. 12126, pp. 140–153. Springer (2020). https://doi.org/10.1007/978-3-030-48966-3_11
11. Demange, M., Paschos, V.T.: On-line vertex-covering. Theor. Comput. Sci. **332**(1–3), 83–108 (2005). https://doi.org/10.1016/j.tcs.2004.08.015

12. 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)
13. Dobrev, S., Královič, R., Pardubská, D.: Measuring the problem-relevant information in input. ITA **43**(3), 585–613 (2009). https://doi.org/10.1051/ita/2009012
14. Emek, Y., Fraigniaud, P., Korman, A., Rosén, A.: Online computation with advice. Theor. Comput. Sci. **412**(24), 2642–2656 (2011). https://doi.org/10.1016/j.tcs.2010.08.007
15. Greene, D.H., Knuth, D.E.: Mathematics for the Analysis of Algorithms, 3rd edn. Birkhäuser (1990)
16. Henrici, P.: Applied and Computational Complex Analysis, vol. 1. Wiley (1988)
17. Hromkovič, J., Královič, R., Královič, R.: Information complexity of online problems. In: P. Hlinený, A. Kucera (eds.) Mathematical Foundations of Computer Science 2010, 35th International Symposium, MFCS 2010, Brno, Czech Republic, August 23–27, 2010. Proceedings, Lecture Notes in Computer Science, vol. 6281, pp. 24–36. Springer (2010). https://doi.org/10.1007/978-3-642-15155-2_3
18. Komm, D.: An Introduction to Online Computation—Determinism, Randomization, Advice. Texts in Theoretical Computer Science. An EATCS Series. Springer (2016). https://doi.org/10.1007/978-3-319-42749-2
19. Komm, D., Královič, R., Královič, R., Kudahl, C.: Advice complexity of the online induced subgraph problem. In: P. Faliszewski, A. Muscholl, R. Niedermeier (eds.) 41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22–26, 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
20. 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, July 10–15, pp. 3302–3311 (2018)
21. Purohit, M., Svitkina, Z., Kumar, R.: Improving online algorithms via ML predictions. Adv. Neural Inf. Process. Syst. **31**, 9684–9693 (2018)
22. 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
23. Wolk, E.S.: The comparability graph of a tree. Proc. Am. Math. Soc. **13**(5), 789–795 (1962)
24. Yannakakis, M.: Node- and edge-deletion NP-complete problems. In: R.J. Lipton, W.A. Burkhard, W.J. Savitch, E.P. Friedman, A.V. Aho (eds.) Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1–3, 1978, San Diego, California, USA, pp. 253–264. ACM (1978). https://doi.org/10.1145/800133.804355