Forbidden Subgraph Problems with Predictions

Hans-Joachim Böckenhauer
Department of Computer Science, ETH Zurich, Switzerland

Melvin Jahn
Department of Computer Science, ETH Zurich, Switzerland

Dennis Komm
Department of Computer Science, ETH Zurich, Switzerland

Moritz Stocker
Department of Computer Science, ETH Zurich, Switzerland

Abstract

In the Online Delayed Connected H-Node-Deletion Problem, an unweighted graph is revealed vertex by vertex and it must remain free of any induced copies of a specific connected induced forbidden subgraph H at each point in time. To achieve this, an algorithm must, upon each occurrence of H, identify and irrevocably delete one or more vertices. The objective is to delete as few vertices as possible. We provide tight bounds on the competitive ratio for forbidden subgraphs H that do not contain true twins or that do not contain false twins.

We further consider the problem within the model of predictions, where the algorithm is provided with a single bit of advice for each revealed vertex. These predictions are considered to be provided by an untrusted source and may be incorrect. We present a family of algorithms solving the problem with predictions and show that it is Pareto-optimal with respect to consistency and robustness for the online vertex cover problem, for 2-connected forbidden subgraphs that do not contain true twins or that do not contain false twins, as well as for any forbidden path of length at least five.

2012 ACM Subject Classification Theory of computation → Adversary models

Keywords and phrases online node deletion, competitive ratio, forbidden subgraphs, predictions

Digital Object Identifier 10.4230/LIPIcs.MFCS.2026.41

Acknowledgements The authors would like to thank Fabian Frei, Matthias Gehnen, and Peter Rossmanith for inspiring this paper by proposing the algorithm $\textsc{Alg}_\lambda$ for the special case $H = P_2$, corresponding to the Delayed Online Vertex Cover Problem.

1 Introduction

Online algorithms receive their input piece by piece as a sequence of *requests* and have to make irrevocable decisions, called *answers* upon each such request. As a result, they face a significant disadvantage compared to traditional algorithms, as they must act on portions of the instance without having complete information about it.

The performance of an online algorithm on an optimization problem is classically measured by comparing its cost to that of an optimal offline solution, which is a best possible solution given complete knowledge of the input in advance. The *competitive ratio* of an online algorithm is defined as the worst-case ratio of its cost on an instance compared to the cost of the optimal offline solution [6]. Some models attempt to further analyze the online nature of a problem by asking how much additional information (so-called *advice*) is needed to improve the competitive ratio of an online algorithm. *Advice complexity* was first studied by Dobrev et al. [10] and further refined by Böckenhauer et al. [4, 5], Hromkovič et al. [13], and Emek et al. [11, 12]. A natural trade-off arises between the size of the advice and the performance of the algorithm utilizing this information. Of course, if the advice is sufficiently large to encode

the entire optimal offline solution, the algorithm can typically obtain a competitive ratio of 1. There is a broad survey on online algorithms with advice by Boyar et al. [7]. For a complete and rigorous introduction to online problems we refer to the textbook by Komm [14].

In a *node-deletion problem* on a graph, the objective is to delete a minimal number of vertices such that the graph satisfies a certain property. Many graph problems are (or can be viewed as) node-deletion problems. The corresponding properties are, for example, cycle-free graphs (Feedback Vertex Set), edge-free graphs (Vertex Cover), or complete graphs (Max-Clique). Yannakakis [21] showed that this kind of problem is NP-complete for general, non-trivial hereditary graph properties. The advice complexity of the corresponding online problems was then studied by Komm et al. [15] using results of Boyar et al. [8]. In this paper, we study deterministic algorithms for online induced forbidden subgraph problems. The input graph is revealed vertex by vertex, along with its corresponding induced edges, and an algorithm has to keep the graph free of any induced subgraph isomorphic to a fixed forbidden subgraph H by deleting vertices. It seems only natural that such an algorithm does not have to decide immediately if a revealed vertex should be deleted or not but only has to decide which vertices to delete once an induced copy of H appears in the online graph. This approach is referred to as the *delayed decision model*, introduced by Rossmanith [20] and named by Chen et al. [9]. It is based on the *preemptive model* used by Komm et al. [15]. The problem itself is known as the Delayed H-Node-Deletion Problem and was studied by Chen et al. [9] and by Berndt and Lotze [3] together with other vertex and edge deletion problems with respect to their advice complexity.

Note that this problem is not equivalent to the classical model where an immediate decision whether or not to delete is required after receiving each vertex. In fact, Chen et al. [9] showed that the problem does not admit any competitive algorithm under the classical model.

A naïve algorithm can just delete all vertices of the induced copy of a forbidden induced subgraph whenever it appears. The intuition behind this is that the "correct" vertex, which could be part of arbitrarily many such subgraphs, is deleted, at the cost of (possibly) deleting a constant number of unnecessary vertices. Chen et al. showed that this strategy is optimal in the case where H is a cycle. In this paper, we extend this analysis to the case where H does not contain true twins or does not contain false twins. This class includes cycles, cliques, and triangle-free graphs, which in turn include bipartite graphs.

We further use advice of the form described by Emek et al. [12] where the online algorithm is augmented by a sequence of advice queries u_t with $t = 1, 2, \ldots$. The query u_t maps the whole request sequence σ to an advice value $u_t(\sigma)$ of fixed size. At the t-th request, the algorithm is provided with advice $u_t(\sigma)$. This model differs from other advice models since it does not reveal the whole advice immediately to the online algorithm but only parts of the advice together with each request. Nevertheless, the *advice oracle* knows the whole input and can optimally design advice for the algorithm accordingly. Specifically, in our model, each time a new vertex is revealed, the algorithm has access to one additional bit of advice that indicates whether the vertex is part of a fixed optimal solution or not.

In traditional models analyzing advice complexity, the oracle is always correct and infallible. In practice however, the advice will be computed under some assumptions that are not completely reliable (e.g., by a machine learning algorithm). If an algorithm blindly trusts the advice and it turns out that it contains errors, the consequences for the performance of the algorithm can be dire. Therefore, it makes sense that an algorithm should be robust against errors in the advice. Several similar models were introduced by Lykouris and Vassilvitskii [17, 18] and by Purohit et al. [19], as the *prediction model* or the model of

machine-learned advice, as well as by Angelopoulos et al. [1, 2] as the model of *untrusted advice*. The concept has gained much interest in recent years and has been considered for many different online problems under various names. For an overview, see the list compiled by Lindermayr and Megow [16]. In this paper, we use the name "predictions" throughout.

Under this model, an algorithm should fulfill two requirements. If the prediction turns out to be correct, the algorithm should perform close to the optimal solution. We define the *consistency* r_{ALG} of an algorithm ALG as its competitive ratio achieved with the best possible prediction that is correct and of the expected form. Nevertheless, if the prediction is incorrect or even maliciously designed by an adversary, the performance of the algorithm should not be compromised too much. The algorithm should hence be robust against incorrect predictions. We define the *robustness* w_{ALG} to be the competitive ratio of the algorithm ALG under worst-case predictions. Therefore, the performance of ALG working with predictions can be expressed as a two-dimensional point $(r_{\text{ALG}}, w_{\text{ALG}})$. An algorithm ALG_1 dominates an algorithm ALG_2 if $r_{\text{ALG}_1} \leq r_{\text{ALG}_2}$ and $w_{\text{ALG}_1} \leq w_{\text{ALG}_2}$. For some online problems, there might not exist a single online algorithm that dominates all others and two algorithms can generally be incomparable. Thus, we search for a *Pareto-optimal* family of algorithms $\mathcal{A}$, i.e., a family of pairwise incomparable algorithms $\mathcal{A}$ such that, for every algorithm ALG', there exists an algorithm $\text{ALG} \in \mathcal{A}$ that dominates ALG'.

We give a family of algorithms ALG_λ with a suitable parameter λ that solves the Delayed H-Node-Deletion Problem with predictions and prove that it is Pareto-optimal for certain forbidden subgraphs H when limiting the predictions to the model we described above. Specifically, our results apply to connected forbidden subgraphs that are 2-vertex connected, and do not contain true twins or do not contain false twins, as well as for forbidden paths with at least five vertices.

2 Preliminaries

We use standard graph notation and consider simple and undirected graphs only. For a given graph $G = (V, E)$, $|G|$ denotes the number of vertices (or nodes) $|V(G)|$; C_k denotes the cycle, P_k the path, and K_k the complete graph consisting of k vertices. For a subset $S \subseteq V$, we define $G - S$ to be the graph $G[V \setminus S]$ induced by the deletion of all vertices $v \in S$.

A graph G is H-free if there is no induced copy of the subgraph H in G, i.e., there exists no induced subgraph isomorphic to H in G. The *open neighborhood* $N(v)$ of vertex v is the set of vertices adjacent to v; the *closed neighborhood* of v is defined as $N[v] = N(v) \cup \{v\}$. Two vertices are *true twins* if they have the same closed neighborhood and *false twins* if they have the same open neighborhood.

An online graph G is a graph that is induced by its vertices, which are revealed one by one. The set of vertices $V(G) = \{v_1, v_2, \ldots, v_n\}$ is ordered by their occurrence in the online instance. The graph G_t is the graph induced by the first t vertices of the online graph G, i.e., $G_t = G[\{v_1, \ldots, v_t\}]$. For a fixed subgraph H, the Delayed H-Node-Deletion Problem on a graph G is to select for every t with $1 \leq t \leq n$ a set $S_t \subseteq V(G_t)$ such that $G_t - S_t$ is H-free and $S_1 \subseteq \cdots \subseteq S_n$. The goal is to minimize the size of S_n. The Delayed Connected H-Node-Deletion Problem is the same problem for a fixed connected subgraph H. An online algorithm working on this problem has to decide on S_t based only on G_t, independently of any vertices that are revealed afterwards. If the algorithm works with advice, it additionally has access to values $u_1(G), \ldots, u_t(G)$ at step t. In the case of predictions (untrusted advice), these can be correct or incorrect. We only consider the case where $u_t(G) \in \{0, 1\}$. The value $u_t(G)$ gives advice for vertex v_t of graph G. If $u_t(G) = 1$,

the advice suggests that the vertex v_t is part of a fixed optimal solution of the problem and should be deleted, i.e., added to the set S_t. If $u_t(G) = 0$, the advice suggests that the vertex v_t should not be deleted. We denote by $u(G)$ the sequence of all values $(u_1(G), \ldots, u_n(G))$.

For this minimization problem, $\text{cost}(\textsc{Opt}(G))$ denotes the cost of an optimal solution on graph G, i.e., the smallest number of vertices that need to be deleted in order for the graph G to be H-free. For a deterministic online algorithm Alg, $\text{cost}(\textsc{Alg}(G))$ represents the number of vertices Alg deletes during its execution on the online graph G and $\text{cost}(\textsc{Alg}^{u(G)}(G))$ represents the number of vertices Alg deletes with access to advice $u_t(G)$ for $1 \leq t \leq n$. Alg is *c-competitive* if, for every online graph G and some constant non-negative α, $\text{cost}(\textsc{Alg}(G)) \leq c \cdot \text{cost}(\textsc{Opt}(G)) + \alpha$. The competitive ratio of Alg is defined as $c_{\textsc{Alg}} = \inf\{c \geq 1 \mid \textsc{Alg} \text{ is } c\text{-competitive}\}$.

If Alg works with predictions, it is (r, w)-competitive if, for every online graph G,

$$\text{cost}(\textsc{Alg}^{u(G)}(G)) \leq r \cdot \text{cost}(\textsc{Opt}(G)) + \alpha$$

for some correct prediction $u(G)$ (as defined above) and

$$\text{cost}(\textsc{Alg}^{u(G)}(G)) \leq w \cdot \text{cost}(\textsc{Opt}(G)) + \alpha$$

for every possible prediction $u(G)$, correct or incorrect, and some constant non-negative α. The consistency $r_{\textsc{Alg}}$ and robustness $w_{\textsc{Alg}}$ are the corresponding competitive ratios of Alg.

3 A Family of Algorithms

A naïve algorithm can solve the Delayed H-Node-Deletion Problem without predictions by deleting every vertex of an induced copy of H whenever it appears. It is k-competitive for $k = |H|$ and was already presented by Chen et al. [9].

Let us now consider a simple family of algorithms $\textsc{Alg}_\lambda$, which solve the Delayed H-Node-Deletion Problem with predictions and establish a first upper bound on the competitive ratios.

▶ **Definition 1** (Algorithm $\textsc{Alg}_\lambda$). *The algorithm $\textsc{Alg}_\lambda$ with parameter $\lambda \in [0, 1)$ works on an online graph G and queries prediction $u_t(G)$ each time a vertex v_t is revealed for t with $1 \leq t \leq |G|$. The value of $u_t(G)$ is a single bit. If a vertex has prediction 1, it suggests that the vertex is part of a fixed optimal solution and should be deleted. $\textsc{Alg}_\lambda$ keeps track of two counters d and e which are initialized to 0.*

Whenever an intact copy of an induced subgraph H appears in the online graph G, $\textsc{Alg}_\lambda$ distinguishes between the following cases to keep the graph H-free:

- *Case 1. If the copy of H does not contain any vertices with prediction 1, then the prediction must be incorrect. Going forward, the algorithm $\textsc{Alg}_\lambda$ deletes all vertices of any induced subgraph H that appears without incrementing e or d.*
- *Case 2. If $e/(e + d) > \lambda$ or $d = 0$, $\textsc{Alg}_\lambda$ deletes all $k = |H|$ vertices of the copy of H and increments d by 1.*
- *Case 3. Else, $\textsc{Alg}_\lambda$ deletes one vertex with prediction 1 of the copy of H and increments e by 1. If the copy of H contains multiple vertices with prediction 1, it deletes only a single one of them, for example, the one that appeared first.*

If multiple intact copies of H appear at once in G, $\textsc{Alg}_\lambda$ chooses one arbitrary copy of H first for the above case distinction and then continues choosing another copy of H until G is H-free. This is done before the next vertex of G is revealed.

It is clear that $\textsc{Alg}_\lambda$ keeps any online graph G H-free since, in every iteration of the above case distinction, at least one induced copy of H is destroyed, and $\textsc{Alg}_\lambda$ iterates until G is H-free before the next vertex is revealed.

The parameter λ essentially indicates how much $\textsc{Alg}_\lambda$ trusts the predictions. The counter e tracks how often $\textsc{Alg}_\lambda$ follows the predictions, while the counter d records how often it disregards it. Before examining the competitiveness of $\textsc{Alg}_\lambda$, we provide two statements that show that $e/(e+d)$ approximates λ well enough.

▶ **Lemma 2.** *Consider an arbitrary online graph G that requires at least one vertex deletion to become H-free. Denote the final value of d after the online execution of $\textsc{Alg}_\lambda$ on G by $\tilde{d}$ and the final value of e by $\tilde{e}$. If $\tilde{d}+\tilde{e}>0$, then $\tilde{e}/(\tilde{e}+\tilde{d}) \leq \lambda + 1/(\tilde{e}+\tilde{d})$.*

Proof. We show that $\tilde{e}/(\tilde{e}+\tilde{d}) \leq \lambda + 1/(\tilde{e}+\tilde{d})$ by proving by induction over $e'+d'>0$ that $e'/(e'+d') \leq \lambda + 1/(e'+d')$ holds for any values e' and d' of e and d during the execution of $\textsc{Alg}_\lambda$.

The base case of $e'+d'=1$ is only possible after Case 2 of $\textsc{Alg}_\lambda$ has been executed once. Therefore, $d'=1$ and $e'=0$, and it follows that $e'/(e'+d') = 0 \leq \lambda + 1$ for any $\lambda \in [0,1)$.

So assume that the induction hypothesis $e'/(e'+d') \leq \lambda + 1/(e'+d')$ holds for some $e'+d'>0$. We show that the property holds for $e'+d'+1$ by case distinction over which counter has been incremented last by $\textsc{Alg}_\lambda$.

- *Case 1.* The counter e has been incremented last to $e'+1$ in Case 3 of the algorithm. Therefore, $e'/(e'+d') \leq \lambda$. Then it follows directly that

$$\frac{e'+1}{e'+d'+1} \leq \frac{e'}{e'+d'} + \frac{1}{e'+d'+1} \leq \lambda + \frac{1}{e'+d'+1}\,.$$

- *Case 2.* The counter d has been incremented last to $d'+1$ in Case 2 of the algorithm. Then, $e'/(e'+d') \leq \lambda + 1/(e'+d')$ holds by the induction hypothesis. If $e'=0$, then $e'/(e'+d'+1) \leq \lambda + 1/(e'+d'+1)$ trivially. If $e' \geq 1$, then

$$\begin{aligned}\frac{e'}{e'+d'+1} &\leq \frac{e'}{e'+d'} - e' \cdot \left(\frac{1}{e'+d'} - \frac{1}{e'+d'+1}\right)\\ &\leq \lambda + \frac{1}{e'+d'} - \left(\frac{1}{e'+d'} - \frac{1}{e'+d'+1}\right)\\ &= \lambda + \frac{1}{e'+d'+1}\,.\end{aligned}$$

◀

▶ **Lemma 3.** *Consider an arbitrary online graph G that requires at least one vertex deletion to become H-free. Denote the final value of d after the online execution of $\textsc{Alg}_\lambda$ on G by $\tilde{d}$ and the final value of e by $\tilde{e}$. If $\tilde{d}+\tilde{e}>0$, then $\tilde{d}/(\tilde{e}+\tilde{d}) \leq (1-\lambda) + 1/(\tilde{e}+\tilde{d})$.*

Proof. This proof works similar to the proof of Lemma 2. The base case $e'+d'=1$ is again only possible after case 2 of $\textsc{Alg}_\lambda$ has been executed once. Therefore, $d'=1$ and $e'=0$, it follows that $d'/(e'+d') = 1 \leq 2-\lambda$ for any $\lambda \in [0,1)$.

So assume that the induction hypothesis $d'/(e'+d') \leq (1-\lambda) + 1/(e'+d')$ holds for some $e'+d'>0$. We show that the property holds for $e'+d'+1$ by case distinction over which counter has been incremented last by $\textsc{Alg}_\lambda$.

- *Case 1.* The counter d has been incremented last to $d'+1$ in Case 2 of the algorithm. Therefore, $e'/(e'+d') > \lambda$, and thus $d'/(e'+d') < 1-\lambda$. It follows directly that

$$\frac{d'+1}{e'+d'+1} \leq \frac{d'}{e'+d'} + \frac{1}{e'+d'+1} \leq 1-\lambda + \frac{1}{e'+d'+1}\,.$$

- *Case 2.* The counter e has been incremented last to $e' + 1$ in case 3 of the algorithm. Since e is only ever incremented if $d > 0$, we know that $d' \geq 1$. This implies that

$$\begin{aligned}\frac{d'}{e' + d' + 1} &= \frac{d'}{e' + d'} - d' \cdot \left(\frac{1}{e' + d'} - \frac{1}{e' + d' + 1}\right) \\ &\leq 1 - \lambda + \frac{1}{e' + d'} - \left(\frac{1}{e' + d'} - \frac{1}{e' + d' + 1}\right) \\ &= 1 - \lambda + \frac{1}{e' + d' + 1}\,.\end{aligned}$$

◀

Now we are ready to prove the core result of this section.

▶ **Theorem 4.** *$\textsc{Alg}_\lambda$ is $(k - \lambda \cdot (k-1), k + \lambda/(1-\lambda))$-competitive for $k = |H|$.*

Proof. Consider an arbitrary online graph G that requires $i > 0$ vertex deletions to become H-free, i.e., $\text{cost}(\textsc{Opt}(G)) = i$. We denote the final value of d after the online execution of $\textsc{Alg}_\lambda$ on G by $\tilde{d}$ and the final value of e by $\tilde{e}$.

If the prediction $u(G)$ is correct, G contains i *optimal vertices* which have prediction 1. There are no other vertices with prediction 1 in G. Every induced copy of H in G contains at least one vertex with prediction 1. $\textsc{Alg}_\lambda$ deletes at most those i vertices with prediction 1 and at most $\tilde{d} \cdot (k-1)$ vertices with prediction 0, because it only deletes vertices with prediction 0 of one induced copy of H when incrementing d, and there are at most $k-1$ vertices with prediction 0 per copy of H. We know that $\tilde{d} + \tilde{e} \leq i$, because every time d or e are incremented in $\textsc{Alg}_\lambda$, at least one of the i vertices with prediction 1 is deleted. Since $i > 0$ and the prediction $u(G)$ is correct, d is incremented at least once by $\textsc{Alg}_\lambda$ and $\tilde{d} + \tilde{e} > 0$. From Lemma 3, it follows that $\tilde{d} \leq (1-\lambda) \cdot (\tilde{e} + \tilde{d}) + 1 \leq i \cdot (1-\lambda) + 1$, and we therefore get

$$\begin{aligned}\text{cost}(\textsc{Alg}_\lambda^{u(G)}(G)) &\leq i + \tilde{d} \cdot (k-1) \\ &\leq i + i \cdot (1-\lambda) \cdot (k-1) + (k-1) \\ &\leq (k - \lambda \cdot (k-1)) \cdot \text{cost}(\textsc{Opt}(G)) + (k-1)\,,\end{aligned}$$

and thus $r_{\textsc{Alg}_\lambda} \leq k - \lambda \cdot (k-1)$ as $k-1$ is constant.

If the prediction $u(G)$ is incorrect but $\textsc{Alg}_\lambda$ does not encounter any induced subgraph H without a vertex with prediction 1, then $\textsc{Alg}_\lambda$ deletes exactly $\tilde{e} + \tilde{d} \cdot k$ vertices. $\textsc{Alg}_\lambda$ guarantees that every time d is incremented, at least one out of the i optimal vertices from a distinct $\textsc{Opt}(G)$ of G is deleted, because every induced copy of H in G must include at least one of those, and every time d is incremented, all vertices of an induced H-subgraph are deleted. Therefore, $\tilde{d} \leq i$ holds. Since $i > 0$, d is incremented at least once by $\textsc{Alg}_\lambda$ and $\tilde{d} + \tilde{e} > 0$. From Lemma 2, it follows that $\tilde{e} \leq \lambda \cdot (\tilde{e} + \tilde{d}) + 1$, which implies

$$\tilde{e} \leq \frac{\lambda \cdot \tilde{d} + 1}{1 - \lambda} \leq \frac{\lambda \cdot i + 1}{1 - \lambda}\,.$$

Hence, we get

$$\text{cost}(\textsc{Alg}_\lambda^{u(G)}(G)) \leq \tilde{e} + \tilde{d} \cdot k \leq \frac{\lambda \cdot i + 1}{1 - \lambda} + i \cdot k \leq \left(k + \frac{\lambda}{1 - \lambda}\right) \cdot \text{cost}(\textsc{Opt}(G)) + \frac{1}{1 - \lambda}\,.$$

If the prediction $u(G)$ is incorrect and $\textsc{Alg}_\lambda$ does in fact encounter an induced copy of H without a vertex with prediction 1, $\textsc{Alg}_\lambda$ still deletes $\tilde{e} + \tilde{d} \cdot k$ vertices until such a copy of H arrives. After that it deletes k vertices v times for some $v > 0$. Note that, after the first copy

of H without any vertex with prediction 1 appeared, e and d are not further incremented and reached their final value $\tilde{e}$ and $\tilde{d}$. The implications of Lemmas 2 and 3 still hold. It further holds that $\tilde{d}+v \leq i$, because every time d is incremented and for every one of the v deletions, ALG_λ deletes at least one of the i optimal vertices of a distinct $\text{OPT}(G)$ of G since ALG_λ deletes all vertices of the appearing H-subgraphs. If $\tilde{d}+\tilde{e} > 0$, then $\tilde{e} \leq (\lambda \cdot i + 1)/(1-\lambda)$ still holds by Lemma 2. Otherwise, $\tilde{e} = 0 \leq (\lambda \cdot i + 1)/(1-\lambda)$ holds trivially. We get

$$\text{cost}(\text{ALG}_\lambda^{u(G)}(G)) \leq \tilde{e} + (\tilde{d}+v)\cdot k \leq \frac{\lambda \cdot i + 1}{1-\lambda} + i \cdot k \leq \left(k + \frac{\lambda}{1-\lambda}\right) \cdot \text{cost}(\text{OPT}(G)) + \frac{1}{1-\lambda}.$$

It follows that $w_{\text{ALG}_\lambda} \leq k + \lambda/(1-\lambda)$ holds since $1/(1-\lambda)$ is constant for fixed λ. ◀

Note that ALG_λ is not defined for $\lambda = 1$, but we can easily define an algorithm ALG_1 which always trusts the predictions and only deletes vertices with prediction 1 if possible. If the prediction is trusted, ALG_1 deletes an optimal solution and is 1-competitive. But if the prediction is untrusted, there exists an online graph G with adversarially chosen predictions on which ALG_1 does not have finite competitive ratio: Consider a forbidden connected subgraph H with $|H| > 1$ and an online graph G that presents arbitrarily many induced copies of H which all overlap at exactly one vertex and are otherwise disjoint such that deleting this vertex would result in an H-free graph. The optimal solution would thus require one deletion. But in the online problem with predictions, the adversary could now choose prediction 0 for this optimal vertex and prediction 1 for some other, non-optimal vertex for each appearing induced copy of H. ALG_1 deletes the vertices with prediction 1 for each copy of H and since there is no copy of H without any vertex with prediction 1, ALG_1 never deletes the optimal vertex. If enough copies of H presented in such a way, the competitive ratio of ALG_1 can be arbitrarily large.

While algorithm ALG_λ (Definition 1) even works for unconnected subgraphs H, we now focus on connected forbidden subgraphs H and hence on the Delayed Connected H-Node-Deletion Problem.

4 Lower Bounds without Predictions

We now inspect the competitive ratios of algorithms solving the Delayed Connected H-Node-Deletion Problem for connected induced forbidden subgraphs H without predictions. Chen et al. [9] showed that there is no online algorithm without advice solving the Delayed H-Node-Deletion Problem for the forbidden subgraph $H = C_k$ (i.e., the cycle on k vertices) with a competitive ratio better than k, for any $k > 4$, by giving adversarial strategies that force k deletions for a gadget that requires 1 deletion in the offline setting. We use and expand their idea for more general subgraphs H.

▶ **Lemma 5.** *Let H be a connected subgraph that does not contain false twins. There does not exist any deterministic algorithm solving the Delayed Connected H-Node-Deletion Problem with a competitive ratio better than $k = |H| > 1$.*

Proof. We construct k different gadgets g^i for $i \in [1,k]$ in the following way. Fix some $1 \leq i \leq k$. First, a copy of H is presented. Label the vertices of H by $v_1, \ldots, v_k$, in the order they are presented. Every time a vertex $v \neq v_i$ is deleted, we present a new vertex v' with the same open neighborhood as v and no other edges. In particular, there is no edge between v and v'. It is clear that each of these re-insertions produces a new induced copy of H in g^i, thus forcing another vertex deletion to keep the gadget H-free. Since v_i can be chosen arbitrarily and is indistinguishable from all remaining vertices of the originally presented

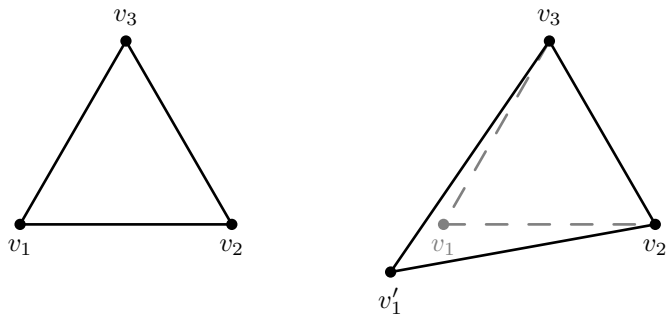

Figure 1 A gadget g^i as used in Lemma 5 for $H = K_3$ and $i \neq 1$. On the left after presenting the first copy of H and on the right after an algorithm deleted v_1 and v_1' was reinserted. Note that any two vertices in H are true twins, but H contains no false twins. The vertices v_1 and v_1' are in the same equivalence class V_1^i. Grey vertices indicate that they were deleted by the algorithm and the incident edges of deleted vertices are displayed as dashed.

subgraph H, there is, for any deterministic algorithm, a gadget g^i for which the algorithm deletes every vertex of the originally presented subgraph H until it deletes v_i (if ever). Thus, on this gadget, the algorithm deletes k vertices before the gadget is H-free.

We now prove that only deleting v_i would have sufficed, i.e., $g^i - \{v_i\}$ is H-free for any g^i formed by the above construction under an arbitrary online algorithm. We partition the set V^i of vertices of g^i in k equivalence classes $V_1^i, \ldots, V_k^i$, such that $v \in V_j^i$ if $v = v_j$ or $v = u'$ was reinserted for some $u \in V_j^i$ for $j \in [1, k]$; see Figure 1 for an illustration. Note that by construction of the reinsertions, if $u, v \in V_j^i$ and $u \neq v$, then u and v are non-adjacent. Also, $V_i^i = \{v_i\}$ since there are no reinsertions for v_i. Therefore, $V^i \backslash \{v_i\}$ is partitioned into $k - 1$ equivalence classes with $V^i \backslash \{v_i\} = \bigcup_{j \neq i} V_j^i$. Any subset of k vertices of $V^i \backslash \{v_i\}$ must include at least two vertices from the same equivalence class, which by design are non-adjacent and have the same neighborhood and thus are false twins. Therefore, there cannot be an induced subgraph H in $g^i - \{v_i\}$ and v_i forms an optimal solution of size one for g^i. Note that, if an algorithm chooses to never delete v_i, it does not have finite competitive ratio because it deletes an arbitrarily large number of vertices on g^i, where one would suffice.

For any deterministic online algorithm Alg that solves this problem and any $m \geq 1$, there exists an adversarial strategy which presents an online graph G^m that repeats m such vertex-disjoint gadgets $g^{i_1}, \ldots, g^{i_m}$, forcing at least k vertex deletions for each gadget where one would suffice by always choosing an i_l such that v_{i_l} is deleted last by Alg in that gadget. Therefore, $\text{cost}(\text{Alg}(G^m)) \geq m \cdot k$ and $\text{cost}(\text{Opt}(G^m)) = m$. It follows that there does not exist any deterministic online algorithm solving this problem with a competitive ratio better than k. ◀

Note that the class of graphs that contain no false twins includes all cliques (because by definition, false twins cannot be connected), as well as cycles C_k for $k \neq 4$. We can show the same property for forbidden subgraphs H that do not contain true twins, i.e., two adjacent vertices with the same closed neighborhood. This graph class contains all connected triangle-free graphs with at least three vertices, such as the cycles C_k for $k \geq 4$, as well as trees and other bipartite graphs.

▶ **Lemma 6.** *Let H be a connected subgraph that does not contain true twins. There does not exist any deterministic algorithm solving the* Delayed Connected H-Node-Deletion Problem *with a competitive ratio better than $k = |H| > 1$.*

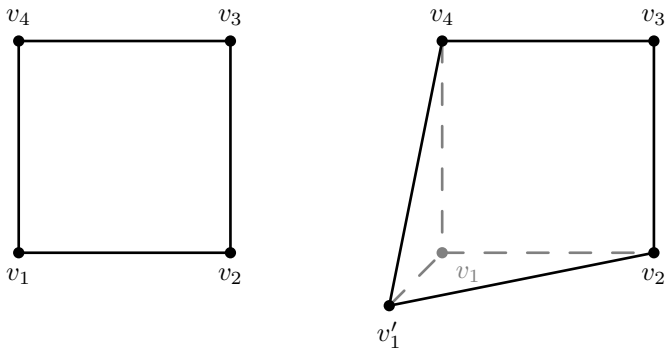

Figure 2 A gadget g^i according to Lemma 6 for $H = C_4$ and $i \neq 1$. On the left after the presentation of the first copy of H and on the right after an algorithm deleted v_1 and v_1' was reinserted. The vertices v_1 and v_1' are in the same equivalence class V_1^i. Note that H contains false twins (for example, v_1 and v_3), but no true twins.

Proof. We prove this with similar gadgets g^i for $i \in [1, k]$ as in Lemma 5. First, a copy of H formed by vertices $v_1, \ldots, v_k$ is presented. We again define reinsertions and a partition of the set V^i of vertices of g^i into k equivalence classes $V_1^i, \ldots, V_k^i$ such that $v \in V_j^i$ if $v = v_j$ or v was reinserted for some v' and $v' \in V_j^i$ for $j \in [1, k]$. Whenever a vertex $v \neq v_i$ is deleted, a new vertex v' is reinserted. Vertex v' shares an edge with every vertex of the neighborhood of v and with every vertex in its equivalence class. As v and v' are in the same equivalence class, the reinsertion leads to them having the same closed neighborhood; see Figure 2 for an illustration.

The remaining part of the proof works similar to the proof of Lemma 5. In particular, since every reinsertion produces a new induced subgraph H in g^i, any deterministic algorithm can be forced to delete at least k vertices until g^i is H-free for some gadget g^i. Since $V_i^i = \{v_i\}$, any subset of k connected vertices of $V^i \backslash \{v_i\}$ must include at least two vertices v and v' from the same equivalence class, which are by construction adjacent and share the same closed neighborhood and are thus true twins. As H consists of k connected vertices and does not include true twins, there cannot be an induced copy of H in $g^i - \{v_i\}$. Therefore, only one vertex deletion is required to make any gadget g^i H-free and vertex v_i is an optimal solution.

By combining m such vertex-disjoint gadgets to a graph G^m, we force any deterministic algorithm Alg to have $\text{cost}(\text{Alg}(G^m)) \geq mk$ and $\text{cost}(\text{Opt}(G^m)) = m$. It follows that there does not exist any deterministic online algorithm solving this problem with a competitive ratio better than k. ◀

Let us note some properties of these gadgets that directly follow from our considerations and which we will further use in the proofs of Theorems 9 and 10.

▶ Remark 7. Let H be a connected subgraph that does not contain true twins or does not contain false twins. For any deterministic algorithm Alg that solves the Delayed Connected H-Node-Deletion Problem and has finite competitive ratio, there exists a gadget g^i described in Lemma 5 (resp. Lemma 6) such that Alg has to eventually delete all k vertices of the copy of H originally presented in g^i, with v_i being the last vertex deleted. Furthermore, every induced copy of H in g^i must consist of exactly one vertex out of each of the k equivalence classes and during the execution of the algorithm Alg on g^i, there exists at most one undeleted vertex of each equivalence class. The adversary can choose $m > 0$ such vertex-disjoint gadgets that form an online graph G and force at least $m \cdot k$ vertex deletions where m deletions would suffice.

The above proofs of Lemmas 5 and 6 introduce two ways to construct powerful gadgets g^i which we continue to use in this paper. Some common subgraphs H which are covered by those lemmas and hence do not yield an algorithm that is better than k-competitive for the Delayed Connected H-Node-Deletion Problem are cliques K_k and induced cycles. Also included are triangle-free connected subgraphs H with $|H| \geq 3$, i.e., subgraphs that do not contain an induced triangle, because true twins form an induced triangle with any of their neighbors and thus cannot exists in triangle-free connected subgraphs with more than two vertices.

5 Lower Bounds with Predictions

We can apply the results we got in Section 4 to our model of predictions. For this, we inspect the competitive ratios of algorithms solving the Delayed Connected H-Node-Deletion Problem with predictions on the gadgets used in Lemmas 5 and 6. Furthermore, we give a class of induced subgraphs H on which the family of algorithms $\textsc{Alg}_\lambda$ (Definition 1) is Pareto-optimal in Theorem 10.

First, we show that no algorithm can distinguish between correct and incorrect predictions early on:

▶ **Lemma 8.** *Consider an online graph G with $|G| = n$ with potentially incorrect predictions $u(G)$ for the Delayed Connected H-Node-Deletion Problem and some fixed H with $|H| > 1$. If the prediction $u(G)$ is not trivially incorrect, i.e., if for each induced copy of H in G at least one vertex has prediction 1, then G can be expanded to an online graph G' with prediction $u(G')$ such that*

- **(i)** *$G'_j = G_j$ for $1 \leq j \leq n$ and $u_t(G') = u_t(G)$ for $1 \leq t \leq n$,*
- **(ii)** *$u_t(G') = 0$ for $t > n$,*
- **(iii)** *$u(G')$ is the unique correct prediction for G'.*

This means that, unless its predictions contain an induced copy of H without vertices with prediction 1, an algorithm can never conclude that the predictions are incorrect.

Proof. Recall that the value of $u_t(G)$ delivers one bit for each vertex v_t of G with $1 \leq t \leq n$. If $u_t(G) = 1$, then the prediction suggests that v_t is part of a fixed optimal solution and should be deleted. Assume that every induced copy of H in G contains at least one vertex v_t with prediction 1, i.e., $u_t(G) = 1$. Let i denote the total number of vertices with prediction 1 in G and $k = |H|$. We show that there exists a graph G' with $G'_j = G_j$ for all $1 \leq j \leq n$ and $|G'| = n + 2i(k-1)$ for which the prediction $u_t(G')$ with $1 \leq t \leq n + 2i \cdot (k-1)$ is correct and encodes a fixed optimal solution. Recall that G'_n is the graph induced by the first n vertices that occur in G'.

To construct G' we expand G by presenting $k-1$ new vertices with prediction 0 for each of the i vertices with prediction 1 in G. These vertices form an induced copy of H together with the vertex with prediction 1 of G and are disjoint to the rest of the graph. We do this twice, so that each of the i vertices with prediction 1 is part of two otherwise disjoint induced copies of H with $2(k-1)$ newly introduced vertices. The graph G' consists of G and these $2i \cdot (k-1)$ new vertices. The prediction queries u_t deliver equal predictions for the vertices of G and G'_n and prediction 0 for the newly added vertices of G', i.e., $u_t(G') = u_t(G)$ for $1 \leq t \leq n$ and $u_t(G') = 0$ for $t > n$. It follows that there are at least $i > 0$ vertex-disjoint induced H-subgraphs in G' and thus $\text{cost}(\textsc{Opt}(G')) \geq i$. The i vertices with prediction 1 therefore form an optimal solution for G', if deleting those makes G' H-free. To show this, assume that there is an induced copy of H in G' after deleting all vertices with prediction 1

to show a contradiction. After the deletions, there cannot be an induced subgraph H that contains any of the $2i \cdot (k-1)$ newly introduced vertices because each of those are, after the deletion of the vertices with prediction 1, in a connected component of size at most $k-1$. Therefore, there must be an induced subgraph H formed by the original vertices of G that does not contain any vertex with prediction 1. But there is by assumption no induced copy of H in G that contains no vertex with prediction 1, which leads to a contradiction. Hence, the i vertices with prediction 1 form an optimal solution for G' and prediction $u(G')$ is correct. To show that $u(G')$ is the only correct prediction for G' we show that the optimal solution formed by the vertices with prediction 1 is unique. For that, assume that there is an optimal solution of size i which does not include some vertex v with prediction 1. The $2(k-1)$ newly introduced vertices for v form two otherwise disjoint induced subgraphs H together with v. If v is not part of an optimal solution, there must be two other vertices out of those $2(k-1)$ newly introduced vertices with prediction 0 that are part of this optimal solution. But deleting v instead of those two vertices would suffice because the newly introduced vertices are not part of any induced subgraph H after deleting v as shown above. Therefore, we would get a smaller optimal solution which is a contradiction. This concludes the proof. ◀

We now state a general result on the relationship between the consistency of robustness of any algorithm solving the Delayed Connected H-Node-Deletion Problem with predictions for graphs satisfying the conditions of Lemmas 5 and 6.

▶ **Theorem 9.** *Let H be a connected subgraph that does not contain true twins or does not contain false twins. Then there does not exist any deterministic algorithm solving the Delayed Connected H-Node-Deletion Problem with predictions that has both a consistency of less than $k-(k-1)/(2-\lambda)$ and a robustness of less than $k+\lambda/(1-\lambda)$ for any $\lambda \in [0,1)$ and $k = |H| > 1$.*

Proof. Consider an arbitrary algorithm Alg solving the Delayed Connected H-Node-Deletion Problem with predictions that has finite competitive ratio. An adversary constructs the following graph G and prediction $u(G)$ which gives bitwise information for each vertex of G. Consider the gadget g^i that we used in the proof of Lemma 5 or, if subgraph H does not contain true twins, of Lemma 6 respectively. The adversary chooses predictions for each vertex $v \in V^i$ for such g^i by assigning prediction 1 to v if $v \in V^i_1$ and prediction 0 to v otherwise. Recall that V^i_j is the equivalence class such that $v \in V^i_j$ if $v = v_j$ or v was reinserted for some v' and $v' \in V^i_j$. From the properties listed in Remark 7, exactly one vertex for each induced subgraph H in g^i has prediction 1 and there is always at most one undeleted vertex with prediction 1 in g^i. The properties of Remark 7 still hold on Alg with such potentially incorrect predictions, because the predictions could be easily constructed by an algorithm without predictions as they do not reveal any additional information. In particular, the adversary can choose $m > 0$ such vertex-disjoint gadgets that form an online graph G and force at least $m \cdot k$ vertex deletions where m deletions would suffice.

The prediction sequence $u(G)$ delivers predictions for each vertex of each gadget as described above. According to Lemma 8, the adversary can now expand the online graph G to an online graph G' such that the graph G' reveals G first with the same prediction for each vertex, and $u(G')$ is the unique correct prediction. This is possible since for every disjoint gadget in G, there is a vertex with prediction 1 in each induced copy of H as shown above. There are, by construction, no other induced copies of H in G. Algorithm Alg performs the same deletions on the vertices of G, independent of whether G is expanded to G' or not, because it operates on the same input until G is fully revealed.

We now analyze the performance of ALG on G. Let x denote the total number of vertex deletions algorithm ALG performs on G with prediction $u(G)$. We already showed that $x \geq m \cdot k$. If G is not expanded to G', the prediction $u(G)$ is potentially incorrect. Furthermore, we know that $\text{cost}(\text{OPT}(G)) = m$. Since m can be chosen arbitrarily large, it follows that

$$w_{\text{ALG}} \geq \frac{\text{cost}(\text{ALG}^{u(G)}(G))}{\text{cost}(\text{OPT}(G))} = \frac{x}{m} \geq k\,.$$

Recall that during its execution on G, ALG deletes at least k vertices on each gadget. Since only one vertex of each gadget has prediction 1, ALG deletes at least $m \cdot (k-1)$ vertices with prediction 0. We denote the number of deleted vertices with prediction 1 by i, which is bounded by the total number of vertex deletions x minus the number of deleted vertices with prediction 0. Therefore, $i \leq x - m \cdot (k-1)$. Further recall that there is at most one undeleted vertex with prediction 1 for each gadget at any point during the execution of algorithm ALG on G. Hence, there are at most m undeleted vertices with prediction 1 in G after the execution of ALG on G. We denote the number of undeleted vertices with prediction 1 in G after the execution of ALG as $a \leq m$. In the construction of G' no additional vertices with prediction 1 are introduced as shown in Lemma 8. The total number of vertices with prediction 1 in G' with $u(G')$ is therefore $i + a$. Since the prediction $u(G')$ is correct by construction of G', the vertices with prediction 1 encode an optimal solution. Thus, $\text{cost}(\text{OPT}(G')) = i + a$. In Lemma 8, G' is constructed by introducing disjoint subgraphs H for each vertex with prediction 1. In particular, those do not contain any other vertex of the original graph G. Therefore, algorithm ALG has to delete at least one additional vertex for each of the a undeleted vertices with prediction 1 in order for G' to become H-free. Algorithm ALG thus deletes a total number of at least $x + a$ vertices during its execution on G' with $u(G')$. Hence,

$$\frac{\text{cost}(\text{ALG}^{u(G')}(G'))}{\text{cost}(\text{OPT}(G'))} \geq \frac{x+a}{i+a} \geq \frac{x+a}{x - m(k-1) + a}\,. \tag{1}$$

We know that $k > 1$, $m w_{\text{ALG}} \geq x \geq m \cdot k$, $m > 0$ and $m \geq a \geq 0$. It follows that

$$m \cdot w_{\text{ALG}} + m \geq x + a\,,$$

which implies

$$\frac{x+a}{(x+a) - m \cdot (k-1)} \geq \frac{m \cdot w_{\text{ALG}} + m}{(m \cdot w_{\text{ALG}} + m) - m \cdot (k-1)} = \frac{w_{\text{ALG}} + 1}{w_{\text{ALG}} - k + 2}\,. \tag{2}$$

Since $\text{OPT}(G') \geq \text{OPT}(G) = m$ can be chosen arbitrarily large, it follows from Equations (1) and (2) that

$$r_{\text{ALG}} \geq \frac{w_{\text{ALG}} + 1}{w_{\text{ALG}} - k + 2}\,.$$

Since $w_{\text{ALG}} \geq k$ we can substitute $w_{\text{ALG}} = \lambda/(1-\lambda) + k$ with $\lambda \in [0, 1)$, which leads to

$$r_{\text{ALG}} \geq \frac{w_{\text{ALG}} + 1}{w_{\text{ALG}} - k + 2} = \frac{\frac{\lambda}{1-\lambda} + k + 1}{\frac{\lambda}{1-\lambda} + k - k + 2} = \frac{k(2-\lambda) - k + 1}{2 - \lambda} = k - \frac{k-1}{2-\lambda}\,. \qquad \blacktriangleleft$$

Note that the lower bound of Theorem 9 does not match the upper bound given by the family of algorithms ALG_λ (Definition 1). But for subgraphs H where we can combine the used gadgets such that the number of leftover vertices with prediction 1 is constant, we are able to prove that the family of algorithms ALG_λ is Pareto-optimal. This is the case for those subgraphs H which are additionally at least 2-vertex-connected.

▶ **Theorem 10.** *Let H be a connected subgraph that does not contain true twins or does not contain false twins. If, additionally, H is 2-vertex-connected, then there does not exist any deterministic algorithm solving the* Delayed Connected H-Node-Deletion Problem *with predictions that has both a consistency of less than $k - \lambda \cdot (k-1)$ and a robustness of less than $k + \lambda/(1-\lambda)$ for any $\lambda \in [0,1)$ and $k = |H| > 1$.*

Proof. We consider an arbitrary algorithm Alg that solves the Delayed Connected H-Node-Deletion Problem for a forbidden subgraph H and has finite competitive ratio. We construct an adversarial online graph G with prediction $u(G)$ for Alg that keeps the number of undeleted vertices with prediction 1 constant. Recall the gadgets g^i from the proof of Lemma 5 for subgraphs H without false twins or of Lemma 6 for subgraphs H without true twins. We assign prediction 1 to vertex v if $v \in V_1^i$ and prediction 0 otherwise. Recall that V_j^i is the equivalence class such that $v \in V_j^i$ if $v = v_j$ or v was reinserted for some v' and $v' \in V_j^i$. The properties of Remark 7 still hold on Alg with such potentially incorrect predictions, because the predictions could again be easily constructed by an algorithm without predictions since they do not reveal any additional information. In particular, we can choose $m > 0$ such vertex-disjoint gadgets that form an online graph G and force at least $m \cdot k$ vertex deletions where m deletions would suffice. It follows that there is exactly one vertex with prediction 1 in each induced copy of H in g^i and there is at most one undeleted vertex with prediction 1 in g^i.

We construct G by choosing $m > 0$ such gadgets $g^{i_1}, \ldots, g^{i_m}$ and combining them in a specific way. The first gadget g^{i_1} is presented normally. If there is no undeleted vertex with prediction 1 after the execution of algorithm Alg on gadget g^{i_j} for $j \in [1, m-1]$, the next gadget $g^{i_{j+1}}$ is presented disjoint from the rest of the graph. This is the case exactly if the vertex with prediction 1 is deleted last in g^{i_j}. Otherwise, vertices with prediction 1 are reinserted upon deletion, which means there is one undeleted vertex v with prediction 1 in gadget g^{i_j} after the execution of the algorithm. The next gadget $g^{i_{j+1}}$ now uses vertex v as the first vertex of the presented copy of H. Note that this vertex would receive prediction 1 by the construction of the predictions anyways. If v is deleted during the construction of $g^{i_{j+1}}$, the vertex which is reinserted for v only shares edges with the other vertices in $g^{i_{j+1}}$ and does not copy the edges between v and any previous vertices of gadget g^{i_j}. All other presented vertices in $g^{i_{j+1}}$ are disjoint to the rest of the graph. This ensures that there is always at most one undeleted vertex with prediction 1 during the execution of Alg on G because every time an undeleted vertex with prediction 1 remains after the execution on one gadget, it is reused for the next gadget.

It is clear that each gadget, even when they are combined in this manner, is still able to enforce that at least all k vertices of the copy of H that is originally presented have to be deleted by algorithm Alg, if there are no induced copies of H between different gadgets, which we prove later. Thus, Alg deletes a total number of $x \geq m \cdot k$ vertices on G with prediction $u(G)$. At least $m \cdot (k-1)$ of those deleted vertices have prediction 0 and thus, at most $x - m \cdot (k-1)$ of them have prediction 1. We now show that, with the given properties of subgraph H, only m vertex deletions are required to make the graph G H-free, i.e., $\text{cost}(\text{Opt}(G)) = m$. It is easy to see that $\text{cost}(\text{Opt}(G)) \geq m$. For this consider the copy of H that is originally presented in each of the m gadgets. Since the gadget enforces that all of its k vertices are deleted, any leftover undeleted vertex, which is used in the next gadget, cannot be part of it. Since all gadgets are otherwise disjoint, those m copies of H are also disjoint and hence require at least m vertex deletions. Now, suppose that $\text{cost}(\text{Opt}(G)) > m$ to show a contradiction. We know that, by construction of each gadget from Lemma 5 (resp. Lemma 6), one vertex deletion suffices to delete all induced copies of H in this gadget. If

$\text{cost}(\textsc{Opt}(G)) > m$, then there must be a remaining copy of H in graph G after the deletion of those m vertices, which are optimal for each gadget. Thus, this remaining copy of H must include vertices from at least two different gadgets, which are not part of both gadgets. We know that subgraph H must be connected. As the gadgets are only connected through at most one vertex v (the reused vertex with prediction 1), this remaining copy of H must include vertex v and at least one other vertex from each of the two different gadgets, since v is part of both gadgets. But since v is the only connection between those two gadgets, deleting v would lead to those other vertices, which are part of the copy of H, being unconnected. This is a direct contradiction to subgraph H being 2-vertex-connected. Hence, it follows that $\text{cost}(\textsc{Opt}(G)) = m$ and because m can be chosen arbitrarily large, $w_{\text{ALG}} \geq x/m \geq k$.

Since there are no copies of H that are not contained within a single gadget, there is a vertex with prediction 1 in each copy of H in the graph G and so by Lemma 8, we can expand the online graph G to an online graph G' such that $u(G')$ is the unique correct prediction for G'. Since the vertices of G are a prefix of the vertices of G', algorithm $\textsc{Alg}$ performs the same on both graphs until all vertices of graph G are fully revealed. We denote by i the total number of vertices with prediction 1 deleted by $\textsc{Alg}$ on G. This number is bounded by $i \leq x - m \cdot (k-1)$ as shown above. We also showed that the number of undeleted vertices with prediction 1 is at most one. If G is expanded to G', no additional vertices with prediction 1 are added and, since the prediction $u(G')$ is correct, the vertices with prediction 1 encode an optimal solution. Hence, $\text{cost}(\textsc{Opt}(G')) \leq i + 1$. Also, $\text{cost}(\textsc{Alg}^{u(G')}(G')) \geq \text{cost}(\textsc{Alg}^{u(G)}(G)) = x$, so

$$\frac{\text{cost}(\textsc{Alg}^{u(G')}(G'))}{\text{cost}(\textsc{Opt}(G'))} \geq \frac{x}{i+1} \geq \frac{x}{x - m(k-1) + 1}\,. \tag{3}$$

We know that $k > 1$, $m \cdot w_{\text{ALG}} \geq x \geq m \cdot k$ and $m > 0$. It follows that

$$\frac{x}{x + 1 - m \cdot (k-1)} \geq \frac{m \cdot w_{\text{ALG}}}{m \cdot w_{\text{ALG}} - m \cdot (k-1) + 1} \geq \frac{w_{\text{ALG}}}{w_{\text{ALG}} - k + 1 + 1/m}\,. \tag{4}$$

From Equations (3) and (4) it follows that for any fixed $\varepsilon > 0$, for sufficiently large m,

$$\frac{\text{cost}(\textsc{Alg}^{u(G')}(G'))}{\text{cost}(\textsc{Opt}(G'))} > \frac{w_{\text{ALG}}}{w_{\text{ALG}} - k + 1} - \varepsilon\,.$$

Since the prediction $u(G')$ is correct and, additionally, $\text{cost}(\textsc{Opt}(G')) \geq \text{cost}(\textsc{Opt}(G)) = m$ can be chosen arbitrarily large, this implies that

$$r_{\text{ALG}} > \frac{w_{\text{ALG}}}{w_{\text{ALG}} - k + 1} - \varepsilon$$

for any $\varepsilon > 0$, and thus $r_{\text{ALG}} \geq w_{\text{ALG}}/(w_{\text{ALG}} - k + 1)$. As $w_{\text{ALG}} \geq k$, we now substitute $w_{\text{ALG}} = k + \lambda/(1-\lambda)$ with $\lambda \in [0, 1)$, yielding

$$r_{\text{ALG}} \geq \frac{w_{\text{ALG}}}{w_{\text{ALG}} - k + 1} = \frac{k + \frac{\lambda}{1-\lambda}}{\frac{\lambda}{1-\lambda} + 1} = \frac{k(1-\lambda) + \lambda}{\lambda + 1 - \lambda} = k - \lambda \cdot (k-1)\,. \qquad \blacktriangleleft$$

We finally consider a simple family of graphs that do not satisfy the conditions of Theorem 10, since they are not 2-vertex connected: the paths P_k of constant length $k \geq 3$ (the path P_2, which corresponds to the Delayed Vertex Cover Problem, is in fact 2-connected). Nevertheless, we can prove an equivalent result for paths P_k with $k \geq 5$. The proof is similar to that of Theorem 10.

▶ **Theorem 11.** *For $H = P_k$ and $k \geq 5$, there does not exist any deterministic algorithm solving the* Delayed H-Node-Deletion Problem *that has both a consistency of less than $k - \lambda \cdot (k-1)$ and a robustness of less than $k + \lambda/(1-\lambda)$ for any $\lambda \in [0, 1)$.*

Proof. The reason the proof of Theorem 10 fails for paths is not the number of deletions that is enforced. Each gadget is still able to enforce k vertex deletions, but when the gadgets are combined to G, they lose the property that they only require one optimal vertex deletion per gadget. This happens because new induced copies of the path may appear between the combined gadgets.

We therefore tweak the gadgets from Theorem 10 such that every vertex of gadget g^{i_j} shares an edge with every vertex which is not part of gadget g^{i_j}. Note that the reused vertex v is both part of g^{i_j} and g^{i_j+1} and that for any two different gadgets, there is at most one reused vertex which is part of both gadgets. We now show that there can be no induced path that is not contained in a single gadget: Suppose there is an induced copy of P_k in G that includes at least two vertices that are not part of the same gadget. Denote these vertices by vertex v_1^x, which is part of the gadget x, and vertex v_1^y, which is part of the gadget y. Furthermore, we denote the set of $k \geq 5$ vertices that form this induced copy of P_k by V_P. As $v_1^x \in V_P$ and $v_1^y \in V_P$ are not part of the same gadget, there is an edge between v_1^x and v_1^y. There cannot be a vertex in V_P which is not part of gadget x or gadget y because it would share an edge with v_1^x and v_1^y, creating a triangle, which cannot be part of any induced P_k. Since there is at most one vertex that is part of both gadgets x and y, there must be two other vertices in V_P which are part of gadgets x and y but not part of both. Suppose those two vertices are part of the same gadget. Without loss of generality, let them be part of gadget x. Therefore, they both share an edge with v_1^y which leads to vertex v_1^y having a degree of at least three in the subgraph induced by V_P, which is not possible if that subgraph is a path. Thus, there must be a vertex $v_2^y \in V_P$, which is part of gadget y and not part of gadget x, and a vertex $v_2^x \in V_P$, which is part of gadget x and not part of gadget y. Thus, there is a cycle of length 4 in the subgraph induced by V_P, namely $v_1^x, v_1^y, v_2^x, v_2^y, v_1^x$, which is again a direct contradiction. The rest of the proof is analogous to the proof of Theorem 10. ◀

6 Conclusion

We have presented tight lower bounds for the Delayed H-Node-Deletion Problem for a large family of forbidden subgraphs. In addition, we presented a family of algorithms which we proved to be Pareto-optimal for a subset of this family, including cliques, cycles, and paths of length at least 5, in a prediction model where each vertex comes with a single bit that predicts whether or not it should be deleted. Further work would be required to generalize these results to further graph classes or even non-induced forbidden subgraphs.

Our results imply tight bounds for the Delayed Vertex Cover Problem, which corresponds to the forbidden subgraph $H = P_2$. To cover other prominent vertex deletion problems such as Feedback Vertex Set, where no cycle of any length may appear, our results would have to be generalized to (potentially infinite) sets $\mathcal{F}$ of forbidden subgraphs, opening up an interesting field of research, both for algorithms with and without predictions. In the model of advice complexity, both Chen et al. [9], and Berndt and Lotze [3] have already considered this case.

Additionally, future research might examine other models of predictions for the problem, which could be considered more realistic in practice than the encoding of an entire optimal solution.

References

1 Spyros Angelopoulos, Christoph Dürr, Shendan Jin, Shahin Kamali, and Marc Renault. Online computation with untrusted advice. In *Proceedings of the 11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ITCS.2020.52.

2 Spyros Angelopoulos, Christoph Dürr, Shendan Jin, Shahin Kamali, and Marc P. Renault. Online computation with untrusted advice. *Journal of Computer and System Sciences*, 144:103545, 2024. doi:10.1016/J.JCSS.2024.103545.

3 Niklas Berndt and Henri Lotze. Advice complexity bounds for online delayed $\mathcal{F}$-node-, H-node- and H-edge-deletion problems. In *Proceedings of the 34th International Workshop on Combinatorial Algorithms (IWOCA 2023)*, pages 62–73. Springer, 2023. doi:10.1007/978-3-031-34347-6_6.

4 Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královič, Richard Královič, and Tobias Mömke. On the advice complexity of online problems. In *Proceedings of the 20th International Symposium on Algorithms and Computation (ISAAC 2009)*, pages 331–340. Springer, 2009. doi:10.1007/978-3-642-10631-6_35.

5 Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královič, Richard Královič, and Tobias Mömke. Online algorithms with advice: The tape model. *Information and Computation*, 254:59–83, 2017. doi:10.1016/j.ic.2017.03.001.

6 Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

7 Joan Boyar, Lene M Favrholdt, Christian Kudahl, Kim S Larsen, and Jesper W Mikkelsen. Online algorithms with advice: A survey. *ACM Computing Surveys (CSUR)*, 50(2):1–34, 2017. doi:10.1145/3056461.

8 Joan Boyar, Lene M Favrholdt, Christian Kudahl, and Jesper W Mikkelsen. Advice complexity for a class of online problems. In *Proceedings of the 32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.STACS.2015.116.

9 Li-Hsuan Chen, Ling-Ju Hung, Henri Lotze, and Peter Rossmanith. Online node- and edge-deletion problems with advice. *Algorithmica*, 83(9):2719–2753, 2021. doi:10.1007/S00453-021-00840-9.

10 Stefan Dobrev, Rastislav Královič, and Dana Pardubská. Measuring the problem-relevant information in input. *RAIRO-Theoretical Informatics and Applications*, 43(3):585–613, 2009. doi:10.1051/ITA/2009012.

11 Yuval Emek, Pierre Fraigniaud, Amos Korman, and Adi Rosén. Online computation with advice. In *Proceedings of the 36th International Colloquium on Automata, Languages, and Programming (ICALP 2009)*, pages 427–438. Springer, 2009. doi:10.1007/978-3-642-02927-1_36.

12 Yuval Emek, Pierre Fraigniaud, Amos Korman, and Adi Rosén. Online computation with advice. *Theoretical Computer Science*, 412(24):2642–2656, 2011. doi:10.1016/J.TCS.2010.08.007.

13 Juraj Hromkovič, Rastislav Královič, and Richard Královič. Information complexity of online problems. In *Proceedings of the 35th International Symposium on Mathematical Foundations of Computer Science (MFCS 2010)*, pages 24–36. Springer, 2010. doi:10.1007/978-3-642-15155-2_3.

14 Dennis Komm. *An Introduction to Online Computation: Determinism, Randomization, Advice*. Springer, 2016. doi:10.1007/978-3-319-42749-2.

15 Dennis Komm, Rastislav Královič, Richard Královič, and Christian Kudahl. Advice complexity of the online induced subgraph problem. In *Proceedings of the 41st International Symposium on Mathematical Foundations of Computer Science (MFCS 2016)*, volume 58, page 59. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.MFCS.2016.59.

16 Alexander Lindermayr and Nicole Megow. ALPS — Algorithms with Predictions. https://algorithms-with-predictions.github.io. [Accessed 23-06-2026].

17 Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. In *International Conference on Machine Learning*, pages 3296–3305. PMLR, 2018. URL: https://proceedings.mlr.press/v80/lykouris18a.html.

18 Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. *Journal of the ACM*, 68(4):24:1–24:25, 2021. doi:10.1145/3447579.

19 Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ML predictions. *Advances in Neural Information Processing Systems*, 31, 2018.

20 Peter Rossmanith. On the advice complexity of online edge- and node-deletion problems. In Hans-Joachim Böckenhauer, Dennis Komm, and Walter Unger, editors, *Adventures Between Lower Bounds and Higher Altitudes - Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday*, volume 11011 of *Lecture Notes in Computer Science*, pages 449–462. Springer, 2018. doi:10.1007/978-3-319-98355-4_26.

21 Mihalis Yannakakis. Node-and edge-deletion NP-complete problems. In *Proceedings of the 10th annual ACM symposium on Theory of computing (STOC 1978)*, pages 253–264, 1978. doi:10.1145/800133.804355.