Proceedings of Machine Learning Research vol 117:1–20, 2020 31st International Conference on Algorithmic Learning Theory

# Interactive Learning of a Dynamic Structure

**Ehsan Emamjomeh-Zadeh*** EHSANEZ@FB.COM
*Facebook, Seattle, WA USA*

**David Kempe** DAVID.M.KEMPE@GMAIL.COM
*Department of Computer Science, University of Southern California, Los Angeles, CA USA*

**Mohammad Mahdian** MAHDIAN@GOOGLE.COM
*Google Research, New York, NY USA*

**Robert E. Schapire** SCHAPIRE@MICROSOFT.COM
*Microsoft Research, New York, NY USA*

**Editors:** Aryeh Kontorovich and Gergely Neu

## Abstract

We propose a general framework for interactively learning combinatorial structures, such as (binary or non-binary) classifiers, orderings/rankings of items, or clusterings, when the underlying structure changes over time. Inspired by Angluin's equivalence query model, the algorithm proposes a structure in each round, and it either learns that its proposal is the true structure in this round, or it observes a specific mistake in the proposal. The feedback is correct only with probability $1-p$, and adversarially incorrect with probability $p$. The algorithm's goal is to minimize its number of mistakes over the course of $R$ rounds.

Our general framework is based on a graph representation of the structures and feedback in a static environment, proposed by Emamjomeh-Zadeh and Kempe (2017). To be able to learn efficiently, it is sufficient that there be a graph $G$ whose nodes are the candidate structures and whose (weighted) edges capture the possible feedback, satisfying a certain natural shortest paths property.

To model the evolution of the underlying structure, we consider two natural models, which we term the Shifting Target model and Drifting Target model. In the former, the true structure always belongs to a small pool of candidate structures. In the latter, the structure can change only by transitioning along the edges of a known evolution graph. In order to achieve non-trivial results, we bound the total number of times the underlying structure can change, denoted by $B$. We provide upper and lower bounds on the number of mistakes, which depend on the total number of changes $B$, the total number of structures $n$, and natural measures of complexity of the dynamic models: the size of the pool of candidate structures in the Shifting Target model, and the maximum degree of the evolution graph in the Drifting Target model.

## 1. Introduction

We consider the task of interactively learning a combinatorial structure, such as a classifier, ranking, or clustering, when the structure itself evolves over time. The point of departure for our work is the equivalence query model of Angluin (1988), and its recent generalization

* Participated in this work while being at Department of Computer Science, University of Southern California, Los Angeles, CA, USA.

to learning combinatorial structures by Emamjomeh-Zadeh and Kempe (2017). In this generalized framework, the learner repeatedly proposes a structure, and receives feedback in the form of a "local correction" that makes the structure more similar to the ground truth, called *target structure* by Emamjomeh-Zadeh and Kempe (2017). The goal in the static model is to minimize the number of rounds of interactions (which is equal to the number of mistakes) until the learner has identified the target structure.

As a concrete example, consider a search engine interacting repeatedly with a user or population of users. With each interaction (in the form of a concrete query), the search engine proposes a ranking of a fixed set of items (with respect to the query); it then receives feedback in the form of a user clicking on a lower-ranked page instead of (or before) clicking on a higher-ranked one. The implication of this interaction is that the item which was clicked preferentially should be ranked ahead of all the items the user skipped (Joachims, 2002; Radlinski and Joachims, 2005); this constitutes a "local" correction that would improve the learner's ranking. In the general model of Emamjomeh-Zadeh and Kempe (2017) (described in more detail in Section 2), there is a known graph $G$ whose nodes are all structures (i.e., the search space), and whose edges capture the possible responses. For the concrete example of the search engine, the nodes are all rankings (permutations) of the pages, and there is an edge from a permutation $\pi$ to $\mu$ if and only if $\mu$ can be obtained by moving one element of $\pi$ ahead by one or more positions.

Emamjomeh-Zadeh and Kempe (2017) show that as long as the graph $G$ satisfies a certain shortest-path property with respect to the responses the learner may receive, a learner can interactively find the target structure in a number of rounds that is logarithmic in the number of possible candidate structures. This result holds even when each response is, with probability $p < \frac{1}{2}$, adversarially incorrect. Subsequently, Dereniowski et al. (2019) generalized this result to fully adversarial noise. Applied to the concrete example of rankings, these results imply that under the given feedback model, a learner can identify the target ranking in $O(n \log n)$ rounds, even in the presence of random or adversarial noise. In addition, the framework naturally recovers past results on learning a classifier or a clustering (Balcan and Blum, 2008; Awasthi and Zadeh, 2010; Awasthi et al., 2017).

The main novel aspect of our work is that we consider the general interactive learning problem in a setting where the target may *evolve* over time. Such changes in the target may naturally occur due to a number of reasons.

First, the learner may be interacting with a *population* of users (such as multiple people sharing the same computer and thus the same search engine or recommendation system) rather than a single user. In this case, there will be a (small) set of target structures, and each user response — when not noisy — is consistent with one of the structures, but the specific structure may change between rounds. We call the resulting model (defined formally in Section 2.2.1) the *Shifting Target* Model. As we discuss more in Section 6, this model resembles the Shifting Expert Advice problem (see, e.g., Bousquet and Warmuth (2002)).

Second, the target structure may gradually evolve. For instance, a user's preferred ranking of web pages or pieces of music may change over time. While the overall set of candidate structures may be large, in each round, the structure will only change slightly. We can model such a gradual evolution by assuming that there is an *evolution graph* $G'$ on all structures; from one round to the next, the target structure can only change from a

structure to a neighboring structure in $G'$. We call this model (defined in Section 2.2.2) the *Drifting Target* Model.

### 1.1. Our Contribution

We provide learning algorithms for both the Shifting Target and Drifting Target models, in the general interactive learning framework of Emamjomeh-Zadeh and Kempe (2017). The algorithms should minimize the number of mistakes over the $R$ rounds, and ideally, each prediction should be computed efficiently in the size of the graph (independently of the number of rounds).

Here, we informally state the main results of our work; see Sections 4 and 5 for formal statements and proofs. In the theorem statements, $n$ is the number of candidate structures, $B$ is an upper bound on the number of times that the underlying target structure changes, and $p$ is the probability of receiving incorrect answers to queries.

**Theorem 1 (Informal)** *Consider the Shifting Target model in which the target structure can shift only within an (unknown) set of at most $k$ structures. There exists an "almost efficient" deterministic learning algorithm*[1] *that makes at most $\frac{1}{1-\mathrm{H}(p)} \cdot \Big(k \cdot \log n + B \cdot \log k + R \cdot \mathrm{H}(B/R)\Big)$ mistakes in expectation, where $\mathrm{H}(p) = p \log \frac{1}{p} + (1-p) \log \frac{1}{1-p}$ denotes the entropy.*

*In the worst case, every algorithm for this problem (including inefficient or randomized algorithms) makes at least $\frac{1}{1-\mathrm{H}(p)} \cdot \Big(k \cdot \log n + B \cdot \log k\Big)$ mistakes in expectation.*

**Theorem 2 (Informal)** *Consider the Drifting Target model in which the target structure moves along the edges of an evolution graph $G'$ with maximum degree $\Delta$. There exists an efficient deterministic algorithm that makes at most $\frac{1}{1-\mathrm{H}(p)} \cdot \Big(\log n + B \cdot \log \Delta + R \cdot \mathrm{H}(B/R)\Big)$ mistakes in expectation. Every algorithm for this problem (even inefficient or randomized ones) makes at least $\frac{1}{1-\mathrm{H}(p)} \cdot \Big(\log n + B \cdot \log \Delta\Big)$ mistakes in expectation.*

In order to prove our lower bounds, in Appendix A, we show a lower bound on the *expected* query complexity of the classic binary search problem on a sorted array. While Ben-Or and Hassidim (2008) show a lower bound that holds with high probability, that bound does not exactly provide the kind of bounds that we need.

### 1.2. Related Work

The model of Emamjomeh-Zadeh and Kempe (2017), and by extensions ours, is based on the equivalence query model of Angluin (1988). The equivalence query model is known to be equivalent to the Online Learning model of Littlestone (1988). Both models focus on learning a (binary) classifier; a large amount of follow-up work has studied this problem.

Interactive learning of a combinatorial structure can be considered as a version of the Multi-Armed Bandit problem or the Expert Advice problem. Each candidate structure corresponds to one arm (or expert). In every round, the loss associated with the target

1. The running time is exponential in $k$, but no other parameters.

structure is 0, while the loss of every other arm is 1. Then, the total loss of any algorithm is the number of mistakes it makes. In our model, the learner does not receive full information about all the experts; rather, it observes the loss of its prediction as well as "directional" feedback to the good expert of this round, both through a noisy channel.

A model with superficially similar flavor, which lies between the Expert Advice problem and the Multi-Armed Bandit problem, is the *Graph Feedback* model (see, e.g., (Alon et al., 2015, 2017)). In this model, the arms/experts are also the nodes of a graph. In each round, the learner observes the loss of the chosen arm as well as the loss of each of its neighbors. Notice that the graph plays very different roles in the Graph Feedback model and in our model: in the Graph Feedback model, it captures observability of rewards, while in our model, it provides directional information.

Active learning of a permutation which evolves throughout the learning process has been studied in a closely related model by Anagnostopoulos et al. (2011); Besa Vial et al. (2018b,a). In their models, the learner *actively* chooses pairs of items to compare. In contrast, in our model, the mistake in the permutation that is revealed to the learner is chosen by an adversary. Furthermore, we treat the problem in the more general setting of learning a general combinatorial structure.

A model very similar to our Shifting Target model was considered by Deligkas et al. (2017). In their model, however, a target is chosen randomly from a small pool with respect to a fixed distribution. They show that if one node is more likely to be the target than all other nodes combined, one can identify this specific target in a logarithmic number of rounds. Their focus is on learning the identity of one target (or more targets, under extra assumptions), rather than minimizing the number of mistakes over a long sequence.

## 2. Preliminaries, Definitions, and Models

We let $\mathrm{H}(\mathcal{X})$ denote the entropy of the random variable $\mathcal{X}$. We also overload this notation and for every $0 < \alpha < 1$, we define $\mathrm{H}(\alpha) = \alpha \log \frac{1}{\alpha} + (1-\alpha) \log \frac{1}{1-\alpha}$ as the entropy of a coin flip with probability $\alpha$. For convenience, we define $\mathrm{H}(0) = \mathrm{H}(1) = 0$. All logarithms are base 2 unless explicitly stated otherwise.

Weighted graphs (directed or undirected) are denoted by $G = (V, E, \omega)$, where $\omega : E \to \mathbb{R}^+$ denotes the edge weights, which will always be strictly positive here. For unweighted graphs, we omit $\omega$.

For every undirected graph $G$ and every node $v$ in $G$, $N_G(v)$ denotes the set of neighbors of $v$ in $G$. If $G$ is a directed graph, $N^{\text{in}}_G(v)$ and $N^{\text{out}}_G(v)$ are the set of nodes with an edge to $v$ and the set of nodes with an edge from $v$, respectively. For an undirected connected and possibly weighted graph $G = (V, E, \omega)$ and a pair of adjacent nodes $u, v$ in $G$, $S_G(u, v)$ denotes the set of all nodes in $G$ with a shortest path from $u$ that starts with the edge $(u, v)$. More formally, $S_G(u, v) = \{w \in V \mid d_G(u, w) = \omega_G(u, v) + d_G(v, w)\}$, where $d_G(u, v)$ denotes the distance from $u$ to $v$ in $G$. For notational convenience, we define $S_G(u, u) = \{u\}$. When the graph $G$ is clear from the context, we will omit the subscript $G$ from the preceding notation.

### 2.1. The Basic Interactive Learning Framework

Our algorithms and analysis are cast in the framework of Emamjomeh-Zadeh and Kempe (2017), which generalizes Angluin's Equivalence Query model (Angluin, 1988) from learning a classifier to learning more general discrete structures, such as permutations, clusterings, and others. We begin with a description of this framework for the case of a static structure and no noise.

In the framework of Emamjomeh-Zadeh and Kempe (2017), all candidate structures comprise the vertices of an undirected, connected, and possibly weighted, graph $G = (V, E, \omega)$. The edges $E$ exactly capture the possible corrections that a learner can receive in response to a prediction, and the weights $\omega$ can be chosen by the algorithm designer. The graph $G$ is known to the learner, but the structure to be learned, called the *target* and denoted by $t$, is an unknown node in $G$. In each round $r$, the learner *predicts* (or *queries*)[2] a node $q_r$. If the prediction is not the target, the learner observes the first edge of *one of the shortest paths* from $q_r$ to $t$. If there are multiple shortest paths from $q_r$ to $t$, the response is chosen adversarially. More formally, the response in round $r$ is a node $z_r \in \{q_r\} \cup N(q_r)$ such that $t \in S(q_r, z_r)$. We say that a node $v$ is *consistent* with this response in round $r$ if $v \in S(q_r, z_r)$, i.e., the response alone does not rule out the possibility that $v$ is the target.

In order to apply this framework to a concrete learning problem, the algorithm designer gets to design the weights $\omega$ such that the "natural" responses that a user would provide satisfy the preceding definition of the process. Emamjomeh-Zadeh and Kempe (2017) show how to define suitable weights for learning classifiers, permutations, or clusterings, under several natural models of interaction. As a particularly straightforward illustration, consider Angluin's original task of learning a binary classifier of $m$ items (Angluin, 1988), when in each round, the learner observes a label correction for exactly one wrongly classified item. Then, a suitable graph structure is the unweighted $m$-dimensional hypercube: each correction exactly corresponds to an edge of the hypercube along the corresponding dimension, and lies on one of the shortest paths to the fully corrected classifier.

A more interesting application of the framework of Emamjomeh-Zadeh and Kempe (2017) is learning a ranking, as discussed in Section 1. Consider a recommendation system that presents a list of $m$ items to a user in each round. If the user skips an item and clicks on the next one, the learner infers that the two items are in incorrect order. Emamjomeh-Zadeh and Kempe (2017) showed that the following undirected and unweighted graph satisfies the shortest-path property: the nodes of $G$ are the permutations over the $m$ items, and two nodes are connected if one permutation is obtained from the other one by an adjacent swap. The graph structure can be naturally extended, using non-uniform edge weights, to an interaction model where a user may be skipping more than one item.

Let $\mathcal{L} : V \to \mathbb{R}^{\geq 0}$ be a function that assigns a non-negative "likelihood" to every node.[3] Let $\Lambda = \sum_{v \in V} \mathcal{L}(v)$ be the total likelihood of all nodes. A median of the graph $G$ with respect to $\mathcal{L}$ is a node $u$ that minimizes $\sum_{v \in V} \mathcal{L}(v) d(u, v)$. Because $G$ is connected and all edge weights are positive, there always is at least one median. The following proposition is immediate from the definition.

2. We use the two terms interchangeably.
3. Informally speaking, the likelihoods indicate the learner's belief about the target, but they do not necessarily capture likelihood in a precise mathematical sense.

**Proposition 3** *If $\mathcal{L}(u) > \frac{\Lambda}{2}$ for a node $u$, then $u$ is the unique median.*

The following Lemma 4 is the key result in (Emamjomeh-Zadeh et al., 2016) (utilized by Emamjomeh-Zadeh and Kempe (2017)). The lemma basically states that if a median of the graph is queried and the response is one of its neighbors, then the total likelihood of the nodes consistent with this response is at most $\frac{\Lambda}{2}$.

**Lemma 4 (Lemma 4 of (Emamjomeh-Zadeh et al., 2016))** *Let $u$ be a median of $G$ and $v \neq u$ a neighbor of $u$. Then $\sum_{w \in S(u,v)} \mathcal{L}(w) \leq \frac{\Lambda}{2}$.*

### 2.2. The Learning Model for Dynamic Structures

In the (static) model of Emamjomeh-Zadeh and Kempe (2017), the target node $t$ never changes. The main modeling contribution in the present work is that we allow the target to change. The learning process proceeds over the course of $R$ rounds. For every round $1 \leq r \leq R$, there is a target node $t_r \in V$. We assume that during the $R$ rounds, the target changes at most $B$ times. More formally, $|\{1 \leq r < R \mid t_{r+1} \neq t_r\}| \leq B$.

As in the static model, the graph $G$ is known to the learner, but the targets are not. In every round $r$, the learner predicts/queries a node $q_r \in V$. In response, it either learns that its prediction in this round is correct (i.e., $q_r = t_r$) or, if this is not the case, it observes the first edge of a shortest path from $q_r$ to $t_r$ in $G$. As in the static model, if there are multiple edges incident on $q_r$ that lie on shortest paths from $q_r$ to $t_r$, any of them could be returned; in particular, the response can be adversarial. Formally, the response in round $r$, denoted by $z_r$, is an adversarial node in $\{q_r\} \cup N_G(q_r)$ such that $t_r \in S_G(q_r, z_r)$. If $q_r \neq t_r$, we say that the algorithm has *made a mistake* in round $r$.

If we place no assumptions on the way in which the target can move in $G$, only trivial bounds on the number of mistakes are possible. We therefore next present two natural restrictions on the moves of $t_r$, which will allow us to prove stronger mistake bounds.

#### 2.2.1. The Shifting Target Model

The *Shifting Target* model aims to model scenarios where the learner interacts with a *small* population of users (or a population with only a small number of types of users), each of whom has a possibly different structure. The learner does not know between consecutive rounds whether it is still interacting with the same user or a different one.

More formally, there exists an unknown set $T \subseteq V$ of (known) size at most $k$, such that $t_r \in T$ for every $1 \leq r \leq R$. In other words, the target only moves within the set $T$.

#### 2.2.2. The Drifting Target Model

The *Drifting Target* model models scenarios where there is only one user (or the users share the same structure), but the structure slowly evolves over time. As an example, think about a ranking capturing a user's music preferences or preferred order of search results; either is wont to evolve over time.

We model the slow evolution as follows. There exists a known directed unweighted *evolution graph* $G' = (V, E')$ (on the same set of nodes $V$), such that for every $1 < r \leq R$, $t_r \in \{t_{r-1}\} \cup N^{\text{out}}_{G'}(t_{r-1})$. In other words, the target can only move along the edges of $G'$.

$G'$ need not be connected. We assume that $G'$, like $G$, is known to the learner; however, we explicitly allow for $G' \neq G$.[4] We write $\Delta = \max_{v \in V} |\{N^{\text{out}}_{G'}(v)\} \cup \{v\}|$ for the maximum out-degree of any node in $G'$ (implicitly assuming that every node has at least a self-loop).

### 2.3. Noise

To model the possibility that a user simply provides a wrong answer in response to a query, we follow Emamjomeh-Zadeh and Kempe (2017) and allow for noisy responses. Each query response is incorrect with probability $p$, independently of the responses for all other rounds. In the case when the response is (randomly) chosen to be incorrect, the *actual* response is adversarial. That is: *whether* a query response is noisy or not is random, but conditioned on being incorrect, the query response itself is adversarially altered. We assume that $p$ is a constant known to the learner. It is easy to see (as pointed out by Emamjomeh-Zadeh et al. (2016)) that if $p \geq \frac{1}{2}$, it is impossible to achieve any non-trivial result, even if the target does not move. Hence, we assume throughout that $p < \frac{1}{2}$.

## 3. A Generic Mistake Bound

In this section, we present a generic result that is applicable to both the Shifting Target and Drifting Target models, as well as more general models of moving targets. The generic learning algorithm, however, is not computationally efficient. In later sections, we show how the more specific Shifting Target and Drifting Target models facilitate efficient implementations of the algorithm.

Let $G = (V, E, \omega)$ be an undirected connected and weighted graph and $a^* = \langle t_1, \ldots, t_R \rangle$ the unknown true sequence of targets throughout the $R$ rounds. Recall that $p$ denotes the probability of incorrect responses.

**Theorem 5** *Let $A = V^R$ be the set of all node sequences of length $R$. Let $\lambda : A \to \mathbb{R}^{\geq 0}$ be a function that assigns non-negative weights to these sequences, such that $\sum_{a \in A} \lambda(a) \leq 1$.*

*There is an online learning algorithm which makes at most $\frac{1}{1-\mathrm{H}(p)} \cdot \log \frac{1}{\lambda(a^*)}$ mistakes in expectation.*

Each sequence $a \in A$ can be considered as "recommending" a node to query in each of the $R$ rounds. Inspired by the classic Hedge algorithm, we consider each sequence as a *meta-expert* and keep track of a weight for each. However, in contrast to the standard Hedge analysis, simply drawing a random meta-expert in every round according to their weights and outputting its recommendation does not guarantee the bound of Theorem 5. Instead, we utilize the "directional" information in $G$ provided by the shortest-path pointers which the algorithm receives. We adapt an idea from (Emamjomeh-Zadeh et al., 2016) to guarantee that after each mistake the algorithm makes, the relative weight of $a^*$ increases significantly. Unlike the Hedge algorithm, our algorithm is deterministic.

A direct adaptation of the analysis of Emamjomeh-Zadeh et al. (2016) results in a mistake bound of $\frac{1}{\lambda(a^*)} + R \cdot \mathrm{H}(B/R)$ for our problem. We modify this analysis in order to prove the bound of Theorem 5; this bound is tight in general (e.g., for the case when the target does not move), as pointed out by Emamjomeh-Zadeh et al. (2016).

4. The case $G' = G$ raises a lot of interesting questions for future work, and is discussed in Section 6.

**Proof** For every $1 \leq r \leq R+1$, the algorithm assigns a weight $\lambda_r(a)$ to every sequence $a \in A$. Initially, $\lambda_1(a) = \lambda(a)$ as given in the input. Define $\Lambda_r = \sum_{a \in A} \lambda_r(a)$ to be total weight of all sequences in round $r$. For the purpose of analysis, we define $\hat{\lambda}_r(a) = \log \lambda_r(a)$ for every sequence $a \in A$ and similarly, $\hat{\Lambda}_r = \log \Lambda_r$. Moreover, we assign a "likelihood" $\mathcal{L}_r(v)$ to every node $v$ in every round $r$, which is essentially the total weight of the sequences which recommend $v$ in round $r$. Formally, $\mathcal{L}_r(v) = \sum_{a \in A}(\lambda_r(a) \cdot \mathbb{1}[a_r = v])$.

In every round $1 \leq r \leq R$, the algorithm's prediction is a median of $G$ with respect to $\mathcal{L}_r(\cdot)$. Let $q_r$ and $z_r$ denote the prediction of the algorithm and the response it receives, respectively. The algorithm then computes the weights of the sequences for the next round as follows. If the recommendation of the sequence $a$ in round $r$ is consistent with the response $z_r$, the weight of $a$ is multiplied by $1-p$. Otherwise, it is multiplied by $p$. More formally, $\lambda_{r+1}(a) = (1-p) \cdot \lambda_r(a)$ if $a_r \in S(q_r, z_r)$, and $\lambda_{r+1}(a) = p \cdot \lambda_r(a)$ otherwise. Recall that if $q_r = z_r$, then $q_r$ itself is the only node that is consistent with the response, i.e., $S(v, v) = \{v\}$.

**Lemma 6** *For every* $1 \leq r \leq R$, $\mathbb{E}[\hat{\lambda}_{r+1}(a^*) - \hat{\lambda}_r(a^*)] \geq -\mathrm{H}(p)$.

**Proof.** In every round $r$, the response is correct with probability at least $1-p$. In this case, the recommendation of $a^*$ is consistent with the response; hence, $\lambda_{r+1}(a^*) = (1-p) \cdot \lambda_r(a^*)$, which implies that $\hat{\lambda}_{r+1}(a^*) = \hat{\lambda}_r(a^*) + \log(1-p)$. In the other case, due to the noise, the recommendation of $a^*$ may or may not be consistent with the response. Because at worst, the weight is multiplied with $p$, we obtain the bound that $\lambda_{r+1}(a^*) \geq p \cdot \lambda_r(a^*)$, i.e., $\hat{\lambda}_{r+1}(a^*) \geq \hat{\lambda}_r(a^*) + \log p$. By combining both cases, we get that

$$\mathbb{E}[\hat{\lambda}_{r+1}(a^*)] \geq (1-p) \cdot \Big(\hat{\lambda}_r(a^*) + \log(1-p)\Big) + p \cdot \Big(\hat{\lambda}_r(a^*) + \log p\Big) = \hat{\lambda}_r(a^*) - \mathrm{H}(p). \quad \blacksquare$$

Now, summing over all $R$ rounds, we obtain that

$$\mathbb{E}[\hat{\lambda}_{R+1}(a^*)] \geq \log \lambda(a^*) - R \cdot \mathrm{H}(p). \tag{1}$$

Having obtained a lower bound on the weight of $a^*$ at the end of the $R$ rounds, we now prove an upper bound on $\hat{\Lambda}_{R+1}$. For each round $1 \leq r \leq R$, one of the following three scenarios happens.

(a) In round $r$, each node $v$ has $\mathcal{L}_r(v) \leq \frac{\Lambda_r}{2}$. By Lemma 4 and because of the way the algorithm updates the weights (as in the analysis of Emamjomeh-Zadeh et al. (2016)), $\Lambda_{r+1} \leq \frac{\Lambda_r}{2}$ and thus, $\hat{\Lambda}_{r+1} - \hat{\Lambda}_r \leq -1$.

(b) In round $r$, there is a node $v \neq a_r^*$ with $\mathcal{L}_r(v) > \frac{\Lambda_r}{2}$. By Proposition 3, $v$ is the unique median and hence the algorithm's prediction. However, $v \neq a_r^*$, so $v$ is not the correct target in this round. Define $\beta = \mathcal{L}_r(v)/\Lambda_r$. By the assumption, $\frac{1}{2} < \beta \leq 1$. With probability $(1-p)$, the response is correct. In this case, $v$ is not consistent with the response, and therefore

$$\Lambda_{r+1} \leq p \cdot \mathcal{L}_r(v) + (1-p) \cdot (\Lambda_r - \mathcal{L}_r(v)) = \Lambda_r \cdot \Big(p \cdot \beta + (1-p) \cdot (1-\beta)\Big).$$

Taking logarithms, this implies that $\hat{\Lambda}_{r+1} - \hat{\Lambda}_r \leq \log\Big(p \cdot \beta + (1-p) \cdot (1-\beta)\Big)$. With the remaining probability, the response is adversarially incorrect. Importantly, it is impossible that both node $v$ and some other node in the graph are consistent with the response. Because both (1 - $p$) and $\beta$ are larger than $\frac{1}{2}$, the smallest reduction in $\Lambda$ is achieved by having $v$ be consistent with the response. Hence, in this case, we obtain the following upper bound:

$$\Lambda_{r+1} \leq (1-p) \cdot \mathcal{L}_r(v) + p \cdot (\Lambda_r - \mathcal{L}_r(v)) = \Lambda_r\Big((1-p) \cdot \beta + p \cdot (1-\beta)\Big).$$

By taking logarithms, we obtain the bound $\hat{\Lambda}_{r+1} - \hat{\Lambda}_r \leq \log\Big((1-p) \cdot \beta + p \cdot (1-\beta)\Big)$. Combining the two cases, we get that

$$\mathbb{E}[\hat{\Lambda}_{r+1} - \hat{\Lambda}_r] \leq (1-p) \cdot \log\Big(p \cdot \beta + (1-p) \cdot (1-\beta)\Big) + p \cdot \log\Big((1-p) \cdot \beta + p \cdot (1-\beta)\Big).$$

For every $0 < p < \frac{1}{2}$ and $\frac{1}{2} \leq \beta \leq 1$, the derivative with respect to $\beta$ is always negative, meaning that the expression is maximized when $\beta = \frac{1}{2}$. Plugging this value of $\beta$ into the inequality, we get that $\mathbb{E}[\hat{\Lambda}_{r+1} - \hat{\Lambda}_r] \leq -1$.

(c) The final case is that in round $r$, $\mathcal{L}_r(a_r^*) > \frac{\Lambda_r}{2}$. By Proposition 3, $a_r^*$ is the unique median and hence, the algorithm's prediction. In this case, the algorithm does not make a mistake (i.e., its prediction is the true target). Define $\beta = \mathcal{L}_r(a_r^*)/\Lambda_r$. Similar to scenario (b), we consider two cases, based on whether the response is correct or not. With probability $1-p$, the response is correct; it then confirms that the algorithm's prediction is correct, and we have

$$\Lambda_{r+1} = (1-p) \cdot \mathcal{L}_r(a_r^*) + p \cdot (\Lambda_r - \mathcal{L}_r(a_r^*)) = \Lambda_r \cdot \Big((1-p) \cdot \beta + p \cdot (1-\beta)\Big).$$

If the response is incorrect, we can again reason as in the scenario (b), and obtain that

$$\Lambda_{r+1} \leq p \cdot \mathcal{L}_r(a_r^*) + (1-p) \cdot (\Lambda_r - \mathcal{L}_r(a_r^*)) = \Lambda_r\Big(p \cdot \beta + (1-p) \cdot (1-\beta)\Big).$$

Taking logs and combining both cases, we get that

$$\mathbb{E}[\hat{\Lambda}_{r+1} - \hat{\Lambda}_r] \leq p \cdot \log\Big(p \cdot \beta + (1-p) \cdot (1-\beta)\Big) + (1-p) \cdot \log\Big((1-p) \cdot \beta + p \cdot (1-\beta)\Big).$$

This time, for every $0 < p < \frac{1}{2}$ and $\frac{1}{2} \leq \beta \leq 1$, the derivative with respect to $\beta$ is positive; therefore, the expression is maximized at $\beta = 1$. Thus, $\mathbb{E}[\hat{\Lambda}_{r+1} - \hat{\Lambda}_r] \leq -\mathrm{H}(p)$.

Let $M$ denote the number of rounds in which either scenario (a) or (b) happens. $M$ is an upper bound on the number of mistakes of the algorithm, and our preceding analysis shows that

$$\mathbb{E}[\hat{\Lambda}_{R+1}] \leq -M - (R-M) \cdot \mathrm{H}(p). \tag{2}$$

Combining Inequalities (1) and (2) and using the fact that $\hat{\Lambda}_{R+1} \geq \hat{\lambda}_{R+1}(a^*)$ (hence, $\mathbb{E}[\hat{\Lambda}_{R+1}] \geq \mathbb{E}[\hat{\lambda}_{R+1}(a^*)]$), we can now complete the proof by bounding

$$-M - (R-M) \cdot \mathrm{H}(p) \geq \log \lambda(a^*) - R \cdot \mathrm{H}(p),$$

which can be rearranged to $M \leq \frac{1}{1-\mathrm{H}(p)} \log \frac{1}{\lambda(a^*)}$. ■

The algorithm we presented in this section explicitly keeps track of weights for all sequences in $A$. If the target does not move, the number of valid sequences cannot exceed the number of the nodes, which allows for an efficient implementation. However, with a moving target, there may be exponentially many sequences. In Sections 4 and 5, we discuss how to implement this algorithm more efficiently for the Shifting Target and Drifting Target models, by carefully choosing the function $\lambda(\cdot)$. Here, we say that an algorithm is *computationally efficient* if it computes its prediction in each round computationally efficiently.

## 4. Results for the Shifting Target Model

In this section, we provide a more efficient implementation of the learning algorithm for the Shifting Target Model, as well as a lower bound on the number of mistakes that any learning algorithm must make in this model.

**Theorem 7** *Under the Shifting Target model, there is a deterministic algorithm that runs in time $O(n^k \operatorname{poly}(n))$ and makes at most $\frac{1}{1-\mathrm{H}(p)} \cdot \Big(k \log n + (B+1) \log k + R \cdot \mathrm{H}(B/R)\Big)$ mistakes in expectation.*

**Proof** We adapt the algorithm of Theorem 5. Let $\mathcal{S}_k$ denote the collection of all subsets of $V$ of size $k$. Consider the following random procedure to generate a sequence $a$ of length $R$. In the following description, $b \in [0,1]$ is a constant, to be determined momentarily.

1. Pick a set $X$ in $\mathcal{S}_k$ uniformly at random.
2. Pick an initial node $a_1 \in X$ uniformly at random.
3. For every $2 \leq r \leq R$, let $a_r = a_{r-1}$ with probability $(1-b)$; with the remaining probability, $a_r$ is chosen uniformly at random from $X$.

For a sequence $a$, let $\Theta(a) \subseteq \mathcal{S}_k$ denote the collection of all sets of (exactly) $k$ nodes that are supersets of the support of $a$. If the support size of $a$ is larger than $k$, then $\Theta(a) = \emptyset$; if it is $k$, then $|\Theta(a)| = 1$. If the sequence's support is smaller than $k$, then $|\Theta(a)| > 1$. Notice that the random procedure can only generate $a$ when it chooses $X \in \Theta(a)$.

As in Theorem 5, let $A = V^R$. For every sequence $a \in A$, let $\lambda(a)$ be the probability that $a$ is generated by the random procedure described above. In particular, if $a$ has support larger than $k$, the probability is $\lambda(a) = 0$. Because $\lambda(\cdot)$ is a probability distribution, $\sum_{a \in A} \lambda(a) = 1$. And because the true sequence $a^*$ shifts at most $B$ times, $\lambda(a^*) \geq \binom{n}{k}^{-1} \cdot \frac{1}{k} \cdot (\frac{b}{k})^B \cdot (1-b)^{R-B}$. Letting $b = \frac{B}{R}$ and applying Theorem 5 gives us the mistake bound of Theorem 7.

In order to bound the running time, we present a more efficient way to implement the algorithm of Theorem 5. The key insight is that in order to run the algorithm, one only needs

to compute, in each round, the likelihood of each node. Fix some set $U \in \mathcal{S}_k$ of $k$ nodes, and let $A_U$ denote the set of all length-$R$ sequences with support $U$. For every node $u \in U$ and every round $1 \le r \le R$, let $\mathcal{L}_{r,U}(u) = \sum_{a \in A_U} \frac{\lambda_r(a)}{|\Theta(a)|} \mathbb{1}[a_r = u]$ be the "contribution" of sequences in $A_U$ in round $r$ to the likelihood of node $u$. Our faster implementation of the algorithm does not keep track of $\lambda_r(a)$ for every sequence $a \in A$. Instead, in every round $1 \le r \le R$, it computes $\mathcal{L}_{r,U}(u)$ for every $U \in \mathcal{S}_k$ and $u \in U$. This is sufficient to compute the cumulative likelihood $\mathcal{L}_r(u) = \sum_{U \ni u} \mathcal{L}_{r,U}(u)$ for every node $u$. In turn, the $\mathcal{L}_r(u)$ are sufficient for computing the median of the graph.

We now show how to inductively compute the $\mathcal{L}_{r,U}(u)$ for a fixed set $U$ and every $u \in U$. For the base case, because the first node is picked uniformly at random, $\mathcal{L}_{1,U}(u) = \binom{n}{k}^{-1} \cdot \frac{1}{k}$. Now assume that for some round $r$, the algorithm has already computed $\mathcal{L}_{r,U}(u)$ for all nodes $u$. As explained above, it can then compute the median $q_r$. As before, let $z_r$ denote the response. For each $u \in U$, if $u$ is consistent with the response, then, according to the description of the algorithm, the weight of every sequence $a \in A_U$ with $a_r = u$ is multiplied by $1-p$. Otherwise, the weight of every such sequence is multiplied by $p$. Define intermediate variables $\mathcal{L}'_{r,U}(u) = \mathcal{L}_{r,U}(u) \cdot \Big((1-p) \cdot \mathbb{1}[u \in S(q_r, z_r)] + p \cdot \mathbb{1}[u \notin S(q_r, z_r)]\Big)$. By the definition of the random generative process for sequences, each sequence stays at the same node with probability $1-b$, and shifts to a uniformly random node in $U$ with the remaining probability $b$. Therefore, the likelihood of the nodes for the next round can be computed as follows: $\mathcal{L}_{r+1,U}(u) = (1-b) \cdot \mathcal{L}'_{r,U}(u) + \sum_{v \in U} \frac{b}{k} \cdot \mathcal{L}'_{r,U}(v)$. This implementation of the algorithm requires memory of size $\binom{n}{k} \in O(n^k)$, and the running time is $O(n^k \operatorname{poly} n)$, as claimed. ■

**Remark 8** *Notice a perhaps interesting feature of our analysis. While the shifts of the target are entirely adversarial (subject to the bound on the support size and the total number of shifts), our analysis essentially "pretends" that the target performs a random walk over a uniformly random target set of size $k$. Despite this "incorrect" shifting model, it achieves the same mistake bound as Theorem 5.*

**Remark 9** *This algorithm needs to know $k$ ahead of time in order to properly keep track of the likelihoods. When $k$ is not known ahead of time, there is a slightly different random procedure for generating random sequences which results in* essentially *the same bound as that of Theorem 7. This random procedure is similar to the one we explained in this section. Instead of picking $X$ from $\mathcal{S}_k$, $X$ is a random set such that each node belongs to $X$ independently with probability $\frac{1}{n}$. The naïve implementation of our algorithm using this new random procedure, however, requires explicitly enumerating all sequences of length $R$. We do not know of any more computationally efficient implementation at this time.*

Next, we state a lower bound on the number of mistakes for any algorithm (deterministic or randomized, efficient or not) in the Shifting Target model. The proof of Theorem 10 is presented in Appendix A.

**Theorem 10** *For every pair of positive integers $k$ and $m$, there exists an undirected, connected and unweighted graph $G$ on a set of $n = km$ nodes such that under the Shifting Target*

*model, every (possibly randomized and/or computationally inefficient) algorithm makes at least*

$$\min\{R - o(R), \frac{1}{1-\mathrm{H}(p)} \cdot \Big(k \log n + (B - 2k + 1) \cdot (\log k)\Big) - o(\log n) - B \cdot o(\log k)\}$$

*mistakes, in expectation.*

The lower bound of Theorem 10 does not quite match the upper bound of Theorem 7; there is an additive gap of essentially $\frac{R \cdot \mathrm{H}(B/R)}{1-\mathrm{H}(p)}$. This gap is most relevant when $k$ is *small*. Whether our upper bound is tight or not remains open.

## 5. Results for the Drifting Target Model

In this section, we state a theorem regarding an improved implementation of the learning algorithm from Section 3 for the Drifting Target Model. The theorem is proved in Appendix B. Unlike the implementation in Section 4, here, we actually achieve a polynomial-time implementation. Recall that $\Delta$ is the maximum degree of the evolution graph $G'$.

**Theorem 11** *Under the Drifting Target model, there is a polynomial-time deterministic learning algorithm which makes at most* $\frac{1}{1-\mathrm{H}(p)} \cdot \Big(\log n + B \cdot \log \Delta + R \cdot \mathrm{H}(B/R)\Big)$ *mistakes in expectation.*

Similar to Section 4, we also state a lower bound, which leaves an additive gap of $\frac{R \cdot \mathrm{H}(B/R)}{1-\mathrm{H}(p)}$. The proof is given in Appendix A.

**Theorem 12** *For every pair of positive integers* $\Delta$ *and* $m$*, there exist undirected unweighted graphs* $G$ *and* $G'$ *over the same set of* $n = \Delta \cdot m$ *nodes such that* $G$ *is connected, the evolution graph* $G'$ *is* $\Delta$*-regular (with a self-loop for every node), and every (possibly randomized or inefficient) algorithm under the Drifting Target model makes at least* $\min\{R - o(R), \frac{1}{1-\mathrm{H}(p)} \cdot \Big(\log n + B \cdot \log \Delta\Big) - o(\log n) - B \cdot o(\log \Delta)\}$ *mistakes in expectation.*

## 6. Conclusions and Future Work

In this work, we presented algorithms for learning a structure (such as a classifier, permutation, or clustering) which evolves over time. For the Shifting Target and the Drifting Target models, we showed how to produce more efficient implementations. Under both models, our bounds are off from each other by an additive gap of essentially $R \cdot \mathrm{H}(B/R)$. Closing this gap is an interesting direction for future work. We conjecture that the upper bound is tight.

The computational complexity of the algorithm for the Shifting Target model is not polynomial in the size of the graph, and it is an interesting direction to find an efficient implementation. The algorithm of Theorem 7 is similar to the "direct" algorithm discussed by Bousquet and Warmuth (2002) (attributed there to Yoav Freund). It is worth mentioning that in the noiseless setting, we can develop a polynomial-time algorithm by essentially reducing this problem to a two-player game in which the first player's choices are the nodes

and the second player's choices are the edges (technical details of the reduction and the algorithm are skipped). For the noisy setting, however, it is more complex: the "loss" of the learner depends on whether it queries the target or not; a predicate whose true value is not observed by the learner.

In this work, we focused on 0-1 loss: the algorithm's loss is 1 in a round if it does not guess the correct target. One can consider other natural loss functions. For example, a natural measure of loss would be the distance $d_G(q_r, t_r)$ in $G$ from $q_r$ to the target. This loss function is well-motivated in the context of interactive learning: the further the guess is from the true target, the more unsatisfied a user will be. It is not clear whether our techniques carry over to this or other loss functions.

One particularly interesting direction for future work arises when the two graphs $G$ and $G'$ in the Drifting Target model are the same; that is, the notion of "similarity" governing a user's responses is the same as the notion of "similarity" governing the structure's evolution over time. If we relax the notion of a "mistake" to count a guess as correct whenever it is within some small distance (such as $O(\log n)$) of the target, one can achieve mistake bounds that are independent of the total number of rounds: once the target has been approximately found, the algorithm can always follow the responses it receives and stay within the same distance. However, it is not clear how quickly an algorithm can get within distance $O(\log n)$ of a moving target for the first time. Designing an algorithm for this case is a very intriguing question that may have to draw on deeper insights into the graph's structure.

Acknowledgments

EE and DK were supported in part by NSF grant IIS-1619458 and ARO MURI grant 72924-NS-MUR. Thanks to Haipeng Luo for several very helpful discussions.

## References

Noga Alon, Nicolo Cesa-Bianchi, Ofer Dekel, and Tomer Koren. Online learning with feedback graphs: Beyond bandits. In *Journal of Machine Learning Research*, volume 40, 2015.

Noga Alon, Nicolo Cesa-Bianchi, Claudio Gentile, Shie Mannor, Yishay Mansour, and Ohad Shamir. Nonstochastic multi-armed bandits with graph-structured feedback. *SIAM Journal on Computing*, 46(6):1785–1826, 2017.

Aris Anagnostopoulos, Ravi Kumar, Mohammad Mahdian, and Eli Upfal. Sorting and selection on dynamic data. *Theoretical Computer Science*, 412(24):2564–2576, 2011.

Dana Angluin. Queries and concept learning. *Machine Learning*, 2:319–342, 1988.

Pranjal Awasthi and Reza Bosagh Zadeh. Supervised clustering. In *Proc. 24th Advances in Neural Information Processing Systems*, pages 91–99, 2010.

Pranjal Awasthi, Maria-Florina Balcan, and Konstantin Voevodski. Local algorithms for interactive clustering. *Journal of Machine Learning Research*, 18:1–35, 2017.

Maria-Florina Balcan and Avrim Blum. Clustering with interactive feedback. In *Proc. 19th Intl. Conf. on Algorithmic Learning Theory*, pages 316–328, 2008.

Michael Ben-Or and Avinatan Hassidim. The bayesian learner is optimal for noisy binary search (and pretty good for quantum as well). In *Proc. 49th IEEE Symp. on Foundations of Computer Science*, pages 221–230, 2008.

Juan José Besa Vial, William E. Devanny, David Eppstein, Michael T. Goodrich, and Timothy Johnson. Optimally sorting evolving data. In *Proc. 45th Intl. Colloq. on Automata, Languages and Programming*, pages 81:1–81:13, 2018a.

Juan José Besa Vial, William E. Devanny, David Eppstein, Michael T. Goodrich, and Timothy Johnson. Quadratic time algorithms appear to be optimal for sorting evolving data. In *2018 Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 87–96, 2018b.

Olivier Bousquet and Manfred K. Warmuth. Tracking a small set of experts by mixing past posteriors. *Journal of Machine Learning Research*, 3:363–396, 2002.

Argyrios Deligkas, George B. Mertzios, and Paul G. Spirakis. Binary search in graphs revisited. In *Mathematical Foundations of Computer Science*, pages 20:1–20:14, 2017.

Dariusz Dereniowski, Daniel Graf, Stefan Tiegel, and Przemysław Uznański. A framework for searching in graphs in the presence of errors. In *Symposium on Simplicity in Algorithms*, 2019.

Ehsan Emamjomeh-Zadeh and David Kempe. A general framework for robust interactive learning. In *Proc. 31st Advances in Neural Information Processing Systems*, pages 7085–7094, 2017.

Ehsan Emamjomeh-Zadeh, David Kempe, and Vikrant Singhal. Deterministic and probabilistic binary search in graphs. In *Proc. 48th ACM Symp. on Theory of Computing*, pages 519–532, 2016. ISBN 978-1-4503-4132-5.

Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proc. 8th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 133–142, 2002.

Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

Filip Radlinski and Thorsten Joachims. Query chains: Learning to rank from implicit feedback. In *Proc. 11th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 239–248, 2005.

## Appendix A. Lower Bound Proofs

In this section, we prove Theorems 10 and 12. Both proofs are based on the lower bound of Theorem 13 for the noisy classic binary search problem with a uniform target.

### A.1. A Lower Bound on Noisy Standard Binary Search

In the classic binary search problem, the algorithm has to find a target $t \in \{1, \ldots, m\}$ using binary comparisons. Formally, we define the *noisy standard binary search* problem as follows. A target $t$ is chosen uniformly at random from the set $\{1, \ldots, m\}$, and the learner's goal is to identify $t$. In each round $r$, the learner queries an element $q_r$. If $q_r = t$, the process ends. Otherwise, the response is a single bit, stating whether $q_r > t$ or not. The response is noisy, meaning that the bit is correct with probability $1 - p$, and incorrect with the remaining probability.

We now prove a theorem which we utilize later to prove Theorems 10 and 12.

**Theorem 13** *Every (possibly randomized) algorithm for the noisy standard binary search problem requires at least* $\frac{\log m}{1-\mathrm{H}(p)} - o(\log m)$ *queries in expectation.*

**Proof** Because the input and responses are not adversarial, but drawn from a known distribution, there is a deterministic optimal algorithm, so it suffices to prove the theorem for deterministic algorithms.

Let $\mathcal{A}'$ be a deterministic algorithm for the noisy standard binary search problem, and assume that it finds the target in at most $Q - 1$ rounds in expectation. ($Q$ may depend on $p$ and $m$, but the dependencies are omitted to simplify the notation.) We first show that at a small additional cost, we can assume that the algorithm *never* uses too many queries.

**Lemma 14** *If there is a deterministic algorithm* $\mathcal{A}'$ *that uses at most* $Q - 1$ *rounds in expectation, then there is a deterministic algorithm* $\mathcal{A}$ *that finds the target in at most* $Q$ *rounds in expectation and never uses more than* $(Q + 1)m$ *queries.*

**Proof** The algorithm $\mathcal{A}$ consists of running $\mathcal{A}'$ for no more than $Qm$ rounds. If the target has not been found yet, then $\mathcal{A}'$ is terminated, and $\mathcal{A}$ queries all the elements in $\{1, \ldots, m\}$ sequentially.

This algorithm never uses more than $Qm + m$ queries. Moreover, the probability that $\mathcal{A}'$ is terminated is, by Markov's inequality, no more than $\frac{1}{m}$. Thus, the query complexity of $\mathcal{A}$ is bounded by $Q - 1 + \frac{1}{m} \cdot m = Q$. ■

In order to prove Theorem 13, we show that for every fixed $\epsilon > 0$, if $m$ is large enough, then $\mathcal{A}$ requires at least $\frac{1-\epsilon}{1-\mathrm{H}(p)} \log m$ queries in expectation. Throughout this proof, $p$ is a fixed constant. Moreover, we assume that $m$ is a (large enough) fixed constant and (without loss of generality) a power of 2. We later discuss how large $m$ should be (as a function of $\epsilon$).

We reduce the standard noisy binary search problem to a coding problem. Let $T_1$ and $T_2$ be two positive integers, depending on $m$ and $p$, which we will specify later. Consider the following random procedure $\mathcal{R}$ which generates a sequence of length $T_1 + T_2$: The first $T_1$ elements of the sequence are drawn independently and uniformly at random from the

set $\{1, \ldots, m\}$. The final $T_2$ elements of the sequence are drawn i.i.d. from $\{0, 1\}$; each of them is 1 with probability $p$. The entropy of the sequence generated from this procedure is $T_1 \cdot \log m + T_2 \cdot \mathrm{H}(p)$, implying Lemma 15.

**Lemma 15** *Every algorithm that constructs a binary encoding of the string generated from the random procedure $\mathcal{R}$ has to use at least $T_1 \log m + T_2 \mathrm{H}(p)$ bits, in expectation.*

We show that using a deterministic algorithm $\mathcal{A}$ for standard noisy binary search, we can construct a good binary encoding algorithm $\mathcal{B}$ for sequences generated by $\mathcal{R}$. By Lemma 14, we can assume w.l.o.g. that $\mathcal{A}$ finds the target in at most $Q$ rounds in expectation and never uses more than $(Q+1)m$ queries. The construction works as follows: Let $s = \langle t_1, \ldots, t_{T_1}, b_1, \ldots, b_{T_2} \rangle$ be a sequence generated from the random procedure $\mathcal{R}$, i.e., the $t_i$ for $1 \leq i \leq T_1$ are drawn independently uniformly at random from the set $\{1, \ldots, m\}$, and each $b_i$ for $1 \leq i \leq T_2$ is 1 independently with probability $p$ and 0 otherwise.

For every $1 \leq i \leq T_1$, the algorithm $\mathcal{B}$ simulates the algorithm $\mathcal{A}$, assuming that the target is $t_i$. The $i^{\text{th}}$ simulation of $\mathcal{A}$ is as follows. In each round, if $\mathcal{A}$ queries $t_i$, $\mathcal{B}$ terminates; otherwise, it uses a fresh bit among $b_1, \ldots, b_{T_2}$ sequentially to determine whether to feed an incorrect or correct response to $\mathcal{A}$. Notice that the algorithm terminates if and when the response to a query confirms that the queried node is the target $t_i$. Up until that event, the response to each query is either "left" or " right." Therefore, the responses can be encoded into a binary string $c_i$. By the assumption on $\mathcal{A}$, $\mathbb{E}[|c_i|] \leq Q$, and $|c_i| \leq (Q+1)m$ always. Let $c_i'$ be a binary string encoding of the length of $c_i$. Thus, $|c_i'| \leq 1 + \log |c_i|$. Define a binary string $c_i''$ of length $|c_i''| = 2|c_i'|$, by adding a 1 after the last digit of $c_i'$ and adding a 0 after each of its other digits.

Finally, define the binary string $\overline{c_i}$ as the concatenation of $c_i''$ with $c_i$. From any string with prefix $\overline{c_i}$, one can first extract $c_i''$, by finding the first digit '1' in an even position. From this, one obtains $c_i'$ from the leading odd digits; after learning the length of $c_i$, one can uniquely reconstruct $c_i$. In turn, from $c_i$, one can uniquely recover the target $t_i$ because $\mathcal{A}$ is deterministic. Moreover, having $c_i$ and $t_i$, it is uniquely determined which responses were noisy. Because $\mathbb{E}[|c_i|] \leq Q$ and log is a concave function, $\mathbb{E}[|c_i''|] \leq 2(\log Q + 1) \leq O(\log Q)$.

Define $c = \overline{c_1} \cdots \overline{c_{T_1}}$ as the binary string obtained by sequentially concatenating $\overline{c_1}, \ldots, \overline{c_{T_1}}$. Based on the previous paragraph, given $c$, one can recover every $t_i$ as well as the bits $b_1, \ldots, b_L$, where $L = |c| = \sum_{1 \leq i \leq T_1} |c_i|$. Notice that $\mathbb{E}[L] \leq T_1 \cdot Q$ and $L \leq T_1 \cdot (Q+1) \cdot m$ always.

Let $\epsilon_1, \epsilon_2 > 0$ be real numbers, which we will let go to 0 later. Let $T_1$ be large enough so that $\Pr[L \geq T_1 Q \cdot (1+\epsilon_1)] < \epsilon_2$. The existence of $T_1$ follows from Chebyshev's Inequality because $\mathbb{E}[|c_i|] \leq Q$, and the variance of $|c_i|$ is bounded by its maximum, which is at most $(Q+1) \cdot m$. Let $T_2 = T_1 \cdot Q \cdot (1+\epsilon_1)$. By definition of $T_2$, with probability at least $1 - \epsilon_2$, the $T_2$ random bits in the second part of the sequence $s$ are sufficient for the $T_1$ executions of $\mathcal{A}$. In this case, the binary string $c$, concatenated with the bits $b_{L+1}, \ldots, b_{T_2}$ which had been left unused in the simulations of $\mathcal{A}$, is sufficient to uniquely determine $s$. The length of this binary string is

$$(T_2 - L) + \sum_{i=1}^{T_1} |\overline{c_i}| \leq T_2 + \sum_{i=1}^{T_1} O(\log |\overline{c_i}|).$$

With the remaining probability, i.e., with probability at most $\epsilon_2$, the $T_2$ bits at the end of $s$ may not be sufficient for $\mathcal{B}$'s $T_1$ simulations of $\mathcal{A}$ In this case, $\mathcal{B}$ simply encodes $s$ by encoding each $t_i$ using $\log m$ bits, and appends the final $T_2$ bits. In this case, the length of the binary code is $T_1 \log m + T_2$. The expected length of the code, denoted by $Q'$, is upper-bounded as follows:

$$\begin{aligned}\mathbb{E}[Q'] &\leq \epsilon_2 \cdot (T_1 \log m + T_2) + (1-\epsilon_2) \cdot (T_1 \cdot O(\log Q) + T_2) \\ &\leq T_1 \cdot (\epsilon_2 \log m + O(\log Q)) + T_2 \\ &\leq T_1 \cdot (\epsilon_2 \log m + O(\log Q) + Q(1+\epsilon_1))\,.\end{aligned}$$

Lemma 15 implies that

$$\mathbb{E}[Q'] \geq T_1 \log m + T_2 \cdot \mathrm{H}(p) \;=\; T_1 \cdot (\log m + Q \cdot \mathrm{H}(p) \cdot (1+\epsilon_1)).$$

Combining both inequalities and canceling out the $T_1$ factor,

$$\log m + Q \cdot \mathrm{H}(p) \cdot (1+\epsilon_1) \leq \epsilon_2 \log m + O(\log Q) + Q \cdot (1+\epsilon_1).$$

Rearranging this inequality gives us that

$$\frac{1-\epsilon_2}{1+\epsilon_1} \cdot \frac{1}{1-\mathrm{H}(p)} \cdot \log m \leq Q + O(\log Q).$$

By letting $\epsilon_1, \epsilon_2 \to 0$, we obtain the claimed lower bound on the expected number of rounds $Q$ for any algorithm to find a random target. ■

### A.2. Proof of Theorem 10

We next show how to use Theorem 13 to prove Theorem 10, which we restate here for convenience.

**Theorem 10** *For every pair of positive integers $k$ and $m$, there exists an undirected, connected and unweighted graph $G$ on a set of $n = km$ nodes such that under the Shifting Target model, every (possibly randomized and/or computationally inefficient) algorithm makes at least*

$$\min\{R - o(R), \frac{1}{1-\mathrm{H}(p)} \cdot \Big(k \log n + (B - 2k + 1) \cdot \log k\Big) - o(\log n) - B \cdot o(\log k)\}$$

*mistakes, in expectation.*

**Proof** Let the graph $G$ be a $k \times m$ grid. Formally, $V = \{(i,j) \mid 1 \leq i \leq k, 1 \leq j \leq m\}$ is the set of $n = k \cdot m$ nodes. Two nodes $(i,j)$ and $(i',j')$ are connected if and only if $|i - i'| + |j - j'| = 1$. Visualizing the grid, we call the first coordinate of a node its *row number* and the second coordinate its *column number*.

For each row $i$, the adversary draws a column $j_i$ independently and uniformly at random. Let $t^{(i)} = (i, j_i)$ be the resulting node in row $i$. These $k$ nodes $t^{(1)}, \ldots, t^{(k)}$ are the targets.

The first stage consists of $k$ phases, each lasting until the learner has identified the target in a particular row. In phase $i$, the adversary lets $t^{(i)}$ be the target. Identifying $t^{(i)}$ is

tantamount to identifying $j_i$, and therefore equivalent to the classic binary search problem on $m$ integers. Therefore, by Theorem 13, every learner must make at least $\frac{\log m}{1-\mathrm{H}(p)} - o(\log m)$ mistakes in expectation until it identifies the target $t^{(i)}$ for the first time. As soon as the learner does, phase $i$ ends, and phase $i+1$ begins. In total over the $k$ phases, the learner must thus make at least $\frac{k \cdot \log m}{1-\mathrm{H}(p)} - k \cdot o(\log m)$ mistakes in expectation.

The second stage consists of $B-k-1$ phases. In each phase, the adversary picks one of the $k$ targets uniformly at random, independently of past choices. A phase ends when the learning algorithm identifies the target correctly. Consider any one phase in the second stage. Unless the learner queried the correct row, the adversary will always provide feedback in the vertical direction, i.e., it identifies whether the target is above or below the queried node. Then, the learner's task in each phase is equivalent to identifying the target's row, and therefore to the classic binary search problem on a path of length $k$. By Theorem 13, it takes the learner at least $\frac{\log k}{1-\mathrm{H}(p)} - o(\log k)$ rounds to identify the target in expectation. The total number of mistakes across all $B-k-1$ phases of the second stage is therefore at least $\frac{(B-k-1)\cdot(\log k)}{1-\mathrm{H}(p)} - B \cdot o(\log k)$ in expectation.

Combining both stages, we obtain the claimed bound. Notice that if $R$ is "small," the adversary may not have enough time to finish this strategy. However, the same strategy results in $R - o(R)$ mistakes in expectation, corresponding to the first term in $\min\{\ldots\}$. ■

### A.3. Proof of Theorem 12

Next, we prove Theorem 12, also restated here for convenience.

**Theorem 12** *For every pair of positive integers $\Delta$ and $m$, there exist undirected unweighted graphs $G$ and $G'$ over the same set of $n = \Delta \cdot m$ nodes such that $G$ is connected, the evolution graph $G'$ is $\Delta$-regular (with a self-loop for every node), and every (possibly randomized or inefficient) algorithm under the Drifting Target model makes at least* $\min\{R - o(R), \frac{1}{1-\mathrm{H}(p)} \cdot \Big(\log n + B \cdot \log \Delta\Big) - o(\log n) - B \cdot o(\log \Delta)\}$ *mistakes in expectation.*

**Proof** Let $G$ be the $\Delta \times m$ grid. Formally, $V = \{(i,j) \mid 1 \le i \le \Delta, 1 \le j \le m\}$ is the set of $n = \Delta \cdot m$ nodes. Two nodes $(i,j)$ and $(i',j')$ are connected in $G$ if and only if $|i-i'| + |j-j'| = 1$. The undirected graph $G'$ is defined on the same set of nodes: in $G'$, two nodes $(i,j)$ and $(i',j')$ are connected if and only if $j = j'$, i.e., they are in the same column. In other words, $G'$ is a disjoint (and disconnected) collection of complete graphs, each comprising the nodes of one of the grid's columns. Thus, each node of $G'$ has degree $\Delta$, counting the self-loop.

The adversary's strategy proceeds in $B+1$ phases $s = 1, \ldots, B+1$. Each phase $s$ has a designated target $t_s$ which stays fixed for the duration of the phase.[5] Phase $s$ ends when the learner queries a node in the same row as $t_s$. Until then, the adversary only reveals vertical information, i.e., it reveals an edge pointing up or down, depending on whether the learner's guess was below or above the target. The initial target $t_1$ is a uniformly random node in $G$. Whenever a new phase $s+1$ starts, the target $t_{s+1}$ is chosen as a uniformly random node in the same column as $t_s$ (and thus also $t_1, \ldots, t_{s-1}$). Because the learner had

5. Notice that we slightly change the meaning of the notation $t_s$, which referred to the target for a given *round* $s$ throughout the body of the paper.

only learned the row of $t_s$, when the row is changed to a new uniformly random one, the target is again uniformly random as far as the learner is concerned.

By Theorem 13, each phase lasts for at least $\frac{\log \Delta}{1-\mathrm{H}(p)} - o(\log \Delta)$ rounds in expectation. Once the adversary has exhausted the $B$ moves, the target stays fixed. At this point, in order to identify the target's column (about which nothing has been revealed so far), the learner still requires at least $\frac{\log m}{1-\mathrm{H}(p)} - o(\log m)$ queries in expectation. Hence, we obtain a lower bound of $(B+1) \cdot \frac{\log \Delta}{1-\mathrm{H}(p)} + \frac{\log m}{1-\mathrm{H}(p)} - (B+1) \cdot o(\log \Delta) - o(\log m)$. Substituting that $m = n/\Delta$ now gives the claimed bound. As in the proof of Theorem 10, if $R$ is "small," the adversary may not have enough time to finish this strategy. However, the same strategy results in $R - o(R)$ mistakes in expectation, corresponding to the first term in $\min\{\ldots\}$. ■

## Appendix B. Proof of Theorem 11

In this section, we prove Theorem 11, restated here for convenience.

**Theorem 11** *Under the Drifting Target model, there is a polynomial-time deterministic learning algorithm which makes at most* $\frac{1}{1-\mathrm{H}(p)} \cdot \Big( \log n + B \cdot \log \Delta + R \cdot \mathrm{H}(B/R) \Big)$ *mistakes in expectation.*

**Proof** As in the proof of Theorem 7, we will prescribe specific choices of weights $\lambda(a)$ for sequences $a$; these weights facilitate efficient computation of the median node to query. As before, let $A = V^R$ denote the set of all sequences of length $R$. Consider a uniformly random walk on $G'$ with stalling probability $1-b$ (which will be determined below), starting from a uniformly random vertex $a_1 \in V$. That is, in each round $r$, with probability $1-b$, the walk stays at $a_{r+1} = a_r$; with the remaining probability, it moves to a uniformly random neighbor of $a_r$ in $G'$. Without loss of generality, we assume that every node has a self-loop in $G'$. For each sequence $a \in A$, define $\lambda(a)$ to be the probability that $a$ occurs as the result of this random walk process.

Because $\lambda(\cdot)$ is a probability distribution, $\sum_{a \in A} \lambda(a) = 1$. And because the ground truth sequence $a^*$ moves at most $B$ times, it has initial weight $\lambda(a^*) \geq \frac{1}{n} \cdot (\frac{b}{\Delta})^B \cdot (1-b)^{R-B}$. Setting $b = \frac{B}{R}$, we apply Theorem 5, which gives a mistake bound of at most $\frac{-1}{1-\mathrm{H}(p)} \cdot \log \Big( \frac{1}{n} \cdot (\frac{b}{\Delta})^B \cdot (1-b)^{R-B} \Big)$, which is exactly the claimed bound.

It remains to show how to implement this algorithm to run efficiently. We again define "likelihoods" $\mathcal{L}_r(u)$ — this time for individual nodes $u$ — and show how to inductively compute them. For the base case, because the first node is chosen uniformly at random, $\mathcal{L}_1(u) = \frac{1}{n}$ for every node $u$. Let $q_r$ and $z_r$ denote the prediction of the algorithm and the response in round $r$, respectively. Consider a node $u$.

- If $u$ is consistent with the response, i.e., $u \in S_G(q_r, z_r)$, then the weight of every sequence $a$ which predicts $u$ in round $r$ (i.e., $a_r = u$) is multiplied by $(1-p)$; as a result, so is the likelihood of $u$.
- Similarly, if $u$ is inconsistent with the response, i.e., $u \notin S_G(q_r, z_r)$, then the weight of every sequence $a$ which predicts $u$ in round $r$ is multiplied by $p$, and so is the likelihood of $u$.

Similar to the proof of Theorem 7, we define intermediate variables

$$\mathcal{L}'_r(u) = \mathcal{L}_r(u) \cdot \Big( (1-p) \cdot \mathbb{1}[u \in S(q_r, z_r)] + p \cdot \mathbb{1}[u \notin S(q_r, z_r)] \Big).$$

Because the weights of sequences correspond to the random walk with stalling probability $1-b$, the new likelihoods are $\mathcal{L}_{r+1}(u) = (1-b) \cdot \mathcal{L}'_r(u) + \sum_{v \in S} \frac{b}{|N^{\text{out}}_{G'}(v)|} \cdot \mathcal{L}'_r(v)$. Notice that the algorithm only needs to keep track of one variable per node and per round, and the computations are straightforward linear combinations. Hence, we obtain an efficient implementation. ■