Augmenting Online Algorithms with ε-Accurate Predictions

Anupam Gupta
Carnegie Mellon University
anupamg@cs.cmu.edu

Debmalya Panigrahi
Duke University
debmalya@cs.duke.edu

Bernardo Subercaseaux
Carnegie Mellon University
bsuberca@cs.cmu.edu

Kevin Sun
Duke University
ksun@cs.duke.edu

Abstract

A growing body of work in learning-augmented online algorithms studies how online algorithms can be improved when given access to ML predictions about the future. Motivated by ML models that give a confidence parameter for their predictions, we study online algorithms with predictions that are ε-accurate: namely, each prediction is correct with probability (at least) ε, but can be arbitrarily inaccurate with the remaining probability. We show that even with predictions that are accurate with a small probability and arbitrarily inaccurate otherwise, we can dramatically outperform worst-case bounds for a range of classical online problems including caching, online set cover, and online facility location. Our main results are an $O(\log(1/\varepsilon))$-competitive algorithm for caching, and a simple $O(1/\varepsilon)$-competitive algorithm for a large family of covering problems, including set cover, network design, and facility location, with ε-accurate predictions.

1 Introduction

The study of online algorithms with ML predictions has gained significant traction in the ML and algorithms research communities in recent years. The basic premise is that by having access to predictions about the future, one can design online algorithms that significantly outperform worst-case bounds. In practice, ML models producing such predictions often come with a *confidence* estimate which can be roughly interpreted as a probability ε of the prediction being accurate—we call these ε-*accurate* predictions. Thus, it is natural to ask: *can we design online algorithms whose performance is a function of the confidence parameter ε of ML predictions?* If $\varepsilon = 1$ and all predictions are correct, we would like the algorithm to approach optimal offline performance (*consistency*). If $\varepsilon = 0$, then the predictions have no guarantees and we would like to ensure that the algorithm is not much worse than an online algorithm without predictions (*robustness*). Between these two extremes, we would like the performance of the algorithm to *gracefully degrade* with the value of ε. In this paper, we present online algorithms augmented with ε-accurate predictions for fundamental problems such as caching, online set cover, online facility location, and online network design.

ε-accurate predictions. We consider ML predictions that are (independently) correct with probability at least ε. Note that ε represents only a *minimum guarantee* on the probability of a prediction being accurate; individual predictions are allowed to have higher confidence values. Moreover, we do not restrict what an incorrect prediction constitutes since this will depend on specific problem settings. But morally, a correct prediction reveals information about the future while an incorrect prediction does not. The design of online algorithms augmented with ε-accurate predictions has two main challenges: (i) that *information (correct predictions) is provided to the algorithm at a slow rate*, about

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

once every $1/\varepsilon$ steps, and (ii) the algorithm needs to *disambiguate information (correct predictions) from noise (incorrect predictions)*. We emphasize that we are interested in the "high noise" regime, i.e., small values of ε, where for every correct prediction, there are many incorrect ones. We will show that even in this regime, we can obtain online algorithms that are substantially better than the worst case. We also do not assume that the value of ε is known to the online algorithm; rather, our algorithms automatically adapt to the (unknown) value of ε in the underlying prediction model. In some cases, however, it will be easier to *describe* the algorithm and analysis if we assume that ε is known. If that is the case, we will first present the simpler algorithm that knows ε and then generalize it to the one that does not know ε.

Prediction error vs ε-accurate predictions. The trend in the online algorithms with predictions literature has been to use prediction error rather than prediction confidence to parameterize the performance of online algorithms. Note the difference in the two measures of prediction quality: an ML model with high confidence (i.e., that produces ε-accurate predictions for ε close to 1) can occasionally make highly inaccurate predictions and therefore have very high prediction error; conversely, an ML model that has a small but consistent bias can have $\varepsilon = 0$ but still have small prediction error. So, the results obtained in this paper are complementary (and incomparable) to the existing literature on online algorithms parameterized by prediction error. To the best of our knowledge, neither caching nor covering problems have been studied in the setting of ε-accurate predictions.

Prior to our work, variants of ε-accurate predictions that also model confidence estimates of predictions have been studied in a few different problem settings such as frequency estimation in data streams [13], k-means [11], secretary problems [10], and page migration [14]. In some cases such as for ski rental [23] and set cover [6], the confidence parameter λ does not represent a measure of correctness of the prediction, rather it is simply a hyperparameter that can be used to trade off the consistency and robustness bounds of the algorithm. In contrast to these, the goal in this paper is to understand the role of ε-accurate predictions in two central problem domains in online algorithms – caching and online covering problems.

Competitive ratio. As in previous literature in this area, we also use the classical performance metric of *competitive ratio* for online algorithms. The competitive ratio of an online algorithm is the worst-case ratio between the (expected) value of the objective in the algorithm's solution to that of the (offline) optimum across all input instances. In the ε-accurate predictions model, this expectation is taken over both the randomness of the algorithm (if any) and the randomness of the predictions. Our competitive ratios are expressed as a function of ε, and gracefully degrade as ε goes from 1 to 0. But, even if $\varepsilon = 0$, we do not lose robustness; using standard techniques (e.g., [4]), we can combine the best online algorithm without predictions and our algorithm with ε-accurate prediction to match (up to constants) the better of the two solutions.

1.1 Our contributions

First, we consider the problem of *caching* with ε-accurate predictions. In this problem, there is an underlying (slow) main memory of n pages; at any given point of time, some $k \ll n$ of these can be stored in a (fast) cache. The online input comprises a sequence of page requests $p_1, p_2, \ldots, p_T$, with page $p_t \in [n]$ being requested at time t. If page p_t is not in the cache at this time, the algorithm must perform a *page swap*: it must *evict* some page from the cache, and *load* page p_t. The goal of the algorithm is the minimize the number of page swaps over the entire sequence of requests.

Lykouris and Vassilvitskii [18] initiated the study of caching with ML predictions and gave deterministic and randomized algorithms whose competitive ratio gracefully degrades with prediction error. (See [7, 15, 24, 27] for subsequent work on this problem.) The general idea in these algorithms is to evict the page whose next request is predicted to be *furthest in the future*, but robustify this strategy by appropriately combining it with a worst-case *randomized marking* algorithm if the predictions turn out to be inaccurate.

In our paper, we consider the caching problem with ε-accurate predictions. At each time, the algorithm is given a prediction for the FIF page in the cache that is correct with probability at least ε; with the remaining probability, the prediction is just a random page in the cache and hence, does not reveal any information about the future. A natural algorithm is to simply follow the prediction, namely evict the predicted FIF page if the currently requested page is outside the cache. We call this the ONE-STRIKE algorithm and show that it achieves a competitive ratio of $O(1/\varepsilon)$. [1]

Our main result is to obtain an exponential improvement over this baseline. To achieve the better bound, we give a more nuanced algorithm that we call the TWO-STRIKES algorithm, and show that it achieves a competitive ratio of $O(\log(1/\varepsilon))$. We show this bound is tight by giving a matching lower bound of $\Omega(\log(1/\varepsilon))$ on the competitive ratio of any algorithm for caching with ε-accurate predictions when $\varepsilon = 1/k \log k$. (Note that as long as $\varepsilon > 0$, ε-accurate predictions are more accurate than predictions generated uniformly at random.)

Next, we consider the problem of solving online covering problems given ε-accurate predictions. Online covering is a general framework for many optimization problems such as set cover and network design. In this problem, the online algorithm has to maintain a solution on a fixed set of variables, and is only allowed to monotonically increase the variables over time. In each online step, the algorithm is presented with a new covering constraint on these variables, and must then augment its existing solution to satisfy the new constraint. The goal is to minimize a linear cost function defined on the variables.

The systematic study of online covering with predictions was initiated by Bamas, Maggiori, and Svensson [6], who used primal dual techniques to leverage a predicted optimal solution given to the online algorithm (see also [1] for subsequent work on online covering with predictions). We consider a model where in each online step, the algorithm gets a covering constraint and a predicted optimal way of satisfying the constraint. (Given an offline predicted solution, these online predictions can be generated on the fly.) However, these are only ε-accurate predictions. Namely, in each online step, the predicted solution to the covering constraint is part of an optimal solution with probability at least ε, and otherwise it is an *arbitrary* way of satisfying the constraint with the remaining probability. Note that in the latter case, the prediction does not reveal any information about the future. We give a simple algorithm for online covering with ε-accurate predictions that obtains a competitive ratio of $O(1/\varepsilon)$, and also a matching lower bound of $\Omega(1/\varepsilon)$. Our covering framework and algorithm are very flexible: they can even model and solve problems like facility location that are not covering problems in a strict sense.

Related work. There has been much recent work on incorporating ML predictions in online algorithms, such as in ad-allocation [19], auction pricing [20], page migration [14], flow allocation [17], scheduling [16, 22, 23], frequency estimation [13], speed scaling [5], Bloom filters [21], bipartite and secretary problems [3], and online linear optimization [9]. In particular, the caching problem was studied by [7, 15, 18, 24, 27], where the prediction model is that at each time t, the algorithm is given a prediction of when currently requested page p_t is *next* requested, and the prediction error η is defined as the ℓ_1 error between the predicted and actual request times. Antoniadis *et al.* [2] consider the more general problem of metrical task systems; their predictor gives a state p_t of the optimal solution, and the error η is the sum of distances between the predictions and actual states. The online set cover problem with predictions was studied by [5], where the (offline) prediction provides an entire feasible solution at the outset. Their algorithm uses the online primal-dual framework, and uses a hyperparameter $\lambda \in [0, 1]$ to obtain a trade off between the consistency and robustness bounds.

1.2 Preliminaries

For every positive integer n, let $[n] := \{1, 2, \ldots, n\}$. Let us formalize the prediction model: at each time t, the algorithm is given some *prediction* or *suggestion* s_t, whose nature depends on the problem statement. At each time t, the algorithm is given one of two possible suggestions: a "good" suggestion y_t, or a "bad" suggestion z_t. While the precise definitions of y_t and z_t depend on the specific problems, we require that the former reveals some information about the optimal solution, and make no such requirement about the latter. There is a fixed value $\varepsilon \in (0, 1]$ that is unknown to the algorithm. Now at each time step t, the ε*-accurate prediction* s_t is determined as follows, independently of the past:

$$s_t = \begin{cases} y_t & \text{with probability at least } \varepsilon \\ z_t & \text{otherwise.} \end{cases}$$

[1]More precisely, we can get $O(\min(1/\varepsilon, \log k)$ by combining ONE-STRIKE with Randomized Marking using standard techniques (see e.g., [2, 18]). As this applies to all our algorithms, we omit writing $O(\min(\cdot, \log k))$ for simplicity.

We aim to bound the competitive ratio of the algorithm in terms of the noise parameter ε. Note that the randomness that determines whether $s_t = y_t$ or $s_t = z_t$ is hidden from both the adversary, who generates the input sequence, and the algorithm.

2 Caching

In the *caching problem* there is an underlying (slow) main memory of n pages; at any given point of time, some $k \ll n$ of these can be stored in the (fast) cache. The online input comprises a sequence of page requests $p_1, p_2, \ldots, p_T$, with page $p_t \in [n]$ being requested at time t. If page p_t is already in the algorithm's cache at this time, the algorithm does not need to do anything, else it must perform a *page swap*: it must *evict* some page in the cache, and *load* the page p_t. The goal of the algorithm is the minimize the number of page swaps over the entire sequence of requests.

Given the request sequence up-front, a strategy that minimizes the number of page swaps is *Bélády's rule* [8]: at each time t, if the requested page p_t is not in the cache, evict the page that is requested the *furthest in the future* (FIF) among the pages in the cache. An online algorithm does not know the future requests, so this is not implementable; instead policies such as LRU [26] and *Randomized Marking* [12] are used to circumvent this lack of foresight by using past requests to predict the future.

In contrast to the recent work which assumes a prediction for the next-arrival time of the page requested at time t [7, 18, 25, 27], we consider a model where at every time t, the algorithm is provided with a prediction on the page in its cache that will be requested furthest in the future. The prediction s_t is a *noisy prediction*: with probability ε it is the FIF page, else it is a *random page* from the algorithm's cache. Using the notation from Section 1.2, y_t is the FIF page in the algorithm's cache, and z_t is a page chosen uniformly at random from the the algorithm's cache. This assumption of uniformity is only made for simplicity. We can in fact show (see Appendix A.2) that it suffices to assume that the probability that a page p is incorrectly predicted as the FIF page is at most $1/k\varepsilon^c$ where c is a constant.

Note the two extremes: if $\varepsilon = 1$ then we can follow Bélády's rule and be optimal; if $\varepsilon = 0$ then the predictions are just random pages in the cache (which can be generated without any knowledge about the future), and we get back the classic worst-case online setting.

Starting with the first request, we partition the input sequence into *phases*, where a phase is defined as a maximal contiguous subsequence of requests containing k distinct pages. (The last phase might contain fewer than k distinct pages because of the termination of the input sequence.) We note that our algorithms do not actually require suggestions to be precisely the FIF page; they work unchanged as long as the correct predictions provide a page that is not requested in the current phase. (The FIF page is just one such page.)

All our algorithms use the idea of a *cache reset* at the beginning of each phase (except the first): replace the contents of the cache with the k pages requested in the previous phase. Since each page brought back in due to a cache reset must have been evicted after it was last requested in the just-ended phase, we get the following lemma.

Lemma 2.1 (Cache Reset Overhead)**.** *If an algorithm that performs cache resets at the beginning of each phase, the number of page swaps in any cache reset is at most the number of page swaps performed in the previous phase.*

These cache resets allow us to *localize* the description and analysis of our algorithms to a single phase, because the state of the cache at the beginning of the phase only depends on the input sequence, and is now independent of the algorithm. Furthermore, we partition the (at most) k pages requested in each phase into two sets: *clean* pages are the ones that were not requested in the previous phase, and the rest are called *stale*. (All the k requested pages in the first phase are considered clean.) Let Δ_i denote the number of clean pages in the i^{th} phase.

2.1 ONESTRIKE: A Deterministic $O(1/\varepsilon)$-Competitive Algorithm

Our first algorithm **ONESTRIKE** using ε-accurate predictions is simple and deterministic, and obtains a competitive ratio of $O(1/\varepsilon)$. In the first phase, it fetches each requested page into the cache. After this point in time, the cache always remains full (with k pages). Each subsequent phase starts with a cache reset. Let C_t denote the algorithm's cache contents at the end of timestep t. During the phase, the algorithm is the following:

(a) If the requested page p_t is already in C_{t-1}, do nothing (i.e., it sets $C_t \leftarrow C_{t-1}$).

(b) Else if $p_t \notin C_{t-1}$, evict the predicted page s_t, i.e., set $C_t \leftarrow (C_{t-1} \setminus s_t) \cup \{p_t\}$.

Theorem 2.2. *The* **ONESTRIKE** *Algorithm is $O(1/\varepsilon)$-competitive.*

Remark. The **ONESTRIKE** algorithm and its analysis both apply assuming a weaker prediction model, in which the z_t predictions (i.e., the non-good ones) are chosen adversarially, rather than uniformly at random among the pages in the algorithm's cache. Note that in this model, an adversary still does not know when good predictions are given, but can adapt to the algorithm's behavior when specifying the z_t predictions.

2.2 TWOSTRIKES: A Randomized $O(\log {}^{1}\!/_{\varepsilon})$-Competitive Algorithm

We now use randomization to improve the competitive ratio to $O(\log(1/\varepsilon))$. Before describing the formal algorithm, we first give some intuition. The **ONESTRIKE** algorithm uses the prediction every time it must make a page swap. This places too much faith on the predictions, and incurs a large loss. Our first change is that the algorithm is more cautious with the predictions, and now views *two* predictions for the same page (and not just a single prediction) as a strong-enough signal to evict the page. We show that the probability that a page that remains the FIF page long enough is predicted twice far outweighs the probability that *any* non-FIF page is (erroneously) predicted twice during the course of the entire phase. As we now need two predictions before evicting a page, we need a fallback option if the requested page is not in the cache and no page has been predicted twice. We run a randomized marking algorithm (call it MARKER) for this purpose, and carefully combine these two algorithms to obtain the final algorithm.

However, this does not suffice: consider a situation where the length of the request sequence for which a page remains the FIF page in the cache is very short. (An extreme example is when pages are requested round-robin, in which case every page is the FIF page right after it is requested and remains so for only a single request.) In this case, none of the FIF pages maintain their FIF status long enough to be predicted twice, and the algorithm devolves to essentially being MARKER with a competitive ratio of $O(\log k)$. To handle this situation, we stop the algorithm once we are confident that we have seen at least Δ FIF pages in the predictions, where Δ is the number of clean pages requested in the phase. At this point, we switch to a different algorithm, which is again MARKER but run only on pages predicted earlier in the phase.

2.2.1 The TWOSTRIKES Algorithm

We now formally describe the **TWOSTRIKES** algorithm. (In this section we assume that we know the accuracy parameter ε: we remove this assumption later in the supplementary material.) As in **ONESTRIKE**, the first phase in **TWOSTRIKES** brings the first k requested pages into the cache and does no evictions. Every subsequent phase begins with a *cache reset* to ensure that the pages in the cache are precisely those requested in the previous phase. We now describe the behavior of the **TWOSTRIKES** algorithm for a single phase.

Epochs and Segments. Since the algorithm does not know Δ (the number of clean requests in this phase), it maintains a *guess* $\widehat{\Delta}$, which starts at 1 and is periodically updated. These updates break the phase into *epochs*: the first epoch starts at the beginning of the phase; each time we observe that the number of clean requests in the phase exceeds our guess $\widehat{\Delta}$, we double the value of $\widehat{\Delta}$, thereby ending the current epoch and starting a new one.

When an epoch starts, **TWOSTRIKES** first performs a cache reset, and then checks if $\varepsilon \leq (\widehat{\Delta}/k)^{1/5}$; if so, it simply runs randomized marking in the rest of the current epoch. Else if $\varepsilon > (\widehat{\Delta}/k)^{1/5}$, the epoch now is partitioned into an *explore segment* followed by an *exploit segment*. In the explore segment, the algorithm makes $\widehat{\Delta}^*$ good evictions of pages that have been predicted twice (for some $\widehat{\Delta}^* \leq \widehat{\Delta}$), and also learns a small candidate set of pages that contains at least $\widehat{\Delta} - \widehat{\Delta}^*$ pages which make for good evictions. In the following exploit segment, the algorithm then runs randomized marking on these candidate pages to actually make $\widehat{\Delta} - \widehat{\Delta}^*$ good evictions.

Before we give details about these two segments, let us give two procedures: MARKER and STRIKER. The MARKER procedure maintains a binary flag called MARK for every page; we say that page p is *marked* if MARK(p) = 1, else it is *unmarked*. We mark and unmark pages by changing the flag to 1 or 0 respectively. All pages are unmarked at the beginning of the epoch. The MARKER algorithm essentially runs the RANDOM MARKING algorithm, but since we may run it starting with only a few unmarked pages, we allow for the possibility of running out of unmarked pages—in which case we declare failure. The details of MARKER appear in Algorithm 1.

Algorithm 1: MARKER

let C be the set of pages in the cache
if *page* p_t *is requested at the current time* t **do**
  case $p_t \in C$ **do** do nothing
  case $p_t \notin C$ *and* $|C| < k$ **do** $C \leftarrow C \cup \{p_t\}$
  case $p_t \notin C$ *and* $|C| = k$ *and* C *has at least one unmarked page* **do**
    let q_t be a uniformly random unmarked page in C, and set $C \leftarrow (C \setminus \{q_t\}) \cup \{p_t\}$
  otherwise do declare FAIL
mark page p_t

The second procedure is STRIKER: it maintains a counter called STRIKE for every page, which takes on values in $\{0, 1, 2\}$. we say that page p is *striked* if STRIKE(p) $\in \{1, 2\}$, otherwise STRIKE(p) $= 0$ and page p is *unstriked*. If STRIKE(p) $= 2$, we say page p is *strike-evicted*. At the beginning of the epoch, all pages are unstriked. The STRIKER procedure operates in two modes: *active* and *passive*. The procedure is in the active mode when the cache contains k pages (i.e., it is full), and it is in the passive mode (and does nothing) otherwise. Like MARKER, STRIKER is not a stand-alone caching algorithm; instead, it is active when the cache is full to acknowledge the prediction, and possibly performs a preemptive eviction. The details appear in Algorithm 2.

Algorithm 2: STRIKER

let C be the set of pages in the cache
if $|C| < k$ **then** do nothing (we are in the passive mode)
else if *page* $s_t \in C$ *is the predicted page at the current time* t **then**
  STRIKE(s_t)++
  if STRIKE(s_t) = 2 **then** evict s_t, so $C \leftarrow C \setminus \{s_t\}$ (so that s_t is *strike-evicted*)

Switching to ONESTRIKE. In both the explore and exploit segments, it is possible for the algorithm to switch to the **ONESTRIKE** algorithm. When this happens, all marks and strikes are forgotten, and the algorithm simply evicts the predicted page whenever an eviction is required. The only state that remains is the count on the number of clean pages in the phase so far: whenever it exceeds $\widehat{\Delta}$, regardless of the state of the algorithm, we double the value of $\widehat{\Delta}$ and start a new epoch.

The Explore Segment. We now describe our algorithm for the explore segment. It uses a global counter called BAD-STRIKES, initialized to 0 at the beginning of each epoch, that counts the number of bad evictions made by the STRIKER procedure. Recall that an eviction at time t is good if the evicted page is not requested in the current phase after time t, and it is bad otherwise. Before serving each request, the algorithm first increments BAD-STRIKES as necessary, and then calls STRIKER and MARKER in that order. The formal description is in Algorithm 3. The explore segment ends if either (a) STRIKER makes at least $2\widehat{\Delta}$ evictions in the segment, or (b) STRIKER has been in active mode for $N := \widehat{\Delta}/\varepsilon^2$ requests.

Lemma 2.3. *In the explore segment, the* MARKER *procedure never declares* FAIL.

Proof. Pages are marked only when they are requested in the current epoch. So, if $p_t \notin C$ and C has k marked pages, then $k + 1$ distinct pages have been requested in the current epoch, and therefore also in the current phase. This contradicts the definition of a phase. □

The Exploit Segment. The exploit segment ignores all predictions. Instead it relies on the fact (proved in Lemma 2.10) that the set of striked pages at the end of the explore segment contains $\widehat{\Delta}$

Algorithm 3: Explore Segment

foreach *time* t **do**
  let p_t be the requested page and s_t the predicted page at time t
  if p_t *is strike-evicted (i.e.,* STRIKE(p_t) $= 2$*)* **then** increment BAD-STRIKES
  if BAD-STRIKES $= \widehat{\Delta}$ **then** run **ONESTRIKE** for rest of epoch **else** set STRIKE(p_t) $\leftarrow 0$
  call STRIKER
  call MARKER
  terminate explore segment if STRIKER evicts pages $\geq 2\widehat{\Delta}$ times, or STRIKER in active mode for $N := \widehat{\Delta}/\varepsilon^2$ requests

pages that would be good evictions (with good probability). The exploit segment now runs MARKER on these striked pages. Formally, see Algorithm 4.

Algorithm 4: Exploit Segment

mark all pages in the cache with no strikes; striked pages are left unmarked
foreach *time* t **do**
  let p_t be page requested at time t
  if p_t *is strike-evicted (i.e.,* STRIKE(p_t) $= 2$*)* **then** increment BAD-STRIKES
  if BAD-STRIKES $= \widehat{\Delta}$ **then** run **ONESTRIKE** for rest of epoch **else** set STRIKE(p_t) $\leftarrow 0$
  call MARKER
  if MARKER *returns* FAIL **then** run **ONESTRIKE** for the rest of the epoch

Since the exploit segment handles the possibility that MARKER may fail (and reverts to **ONESTRIKE** in that case), the algorithm is well-defined; it only remains to bound the expected number of evictions per epoch, and hence per phase. This is what we do next.

2.2.2 Competitive Ratio of the TWOSTRIKES Algorithm

Theorem 2.4. *The algorithm performs* $O(\widehat{\Delta} \log 1/\varepsilon)$ *evictions in expectation in an epoch.*

Proof. There are four types of evictions that happen during an epoch:

(i) evictions performed by STRIKER in the explore segment,
(ii) evictions performed by MARKER in the explore segment,
(iii) evictions performed by MARKER in the exploit segment, and
(iv) evictions due to the **ONESTRIKE** algorithm,

We show that the expected number of each of type of eviction is at most $O(\widehat{\Delta} \log 1/\varepsilon)$. The type (i) evictions are the easiest: there are at most $2\widehat{\Delta}$ of these, by the termination condition of the explore segment.

Next we bound the number of evictions of type (iii): consider running MARKER with some cache C_0, where some $k - r$ pages are pre-marked. Consider the r unmarked pages of C_0, and order them as $p_1, p_2, \ldots, p_r$ in *reverse chronological order* of their first request in this segment. The pages that are not requested at all are placed at the beginning of this sequence, but can be arbitrarily ordered relative to each other. Let D be the number of requests for pages outside the set C_0 received by the algorithm. The following property holds by induction, since only D pages are evicted, and each unmarked page is equally likely to be evicted at any time:

Lemma 2.5. *At any time t, suppose pages* $p_{i_t+1}, p_{i_t+2}, \ldots, p_r$ *have already been requested and pages* $p_1, p_2, \ldots, p_{i_t}$ *have not been requested yet. Then, for any* $i \leq i_t$*, the probability that page* p_i *is not in the cache of the* MARKER *procedure at time t is at most* $\min(D/i_t, 1)$.

Note that the property of this lemma is unconditional, in the sense that it does not depend on the set of pages among $p_{i_t+1}, p_{i_t+2}, \ldots, p_r$ that were evicted before time t. The following corollary follows by applying this lemma at the time of the first request for page p_i:

Corollary 2.6. *The probability that the cache of the* MARKER *procedure does not contain page p_i at the time of its first request is at most* $\min(D/i, 1)$.

The next lemma is proved using the above corollary:

Lemma 2.7. *The expected number of evictions of type (iii) in an epoch is* $O(\widehat{\Delta}\log(1/\varepsilon))$.

Proof. The explore segment terminates after it sees at most $N = \widehat{\Delta}/\varepsilon^2$ requests for which STRIKER is in active mode. Let S denote the set of striked pages; clearly, $|S| \le N$. Moreover, the number of distinct clean page requests is at most $\widehat{\Delta}$, else the epoch ends. By Corollary 2.6, the expected number of evictions in S is at most

$$\sum_{i=1}^{|S|} \min\left(\frac{\widehat{\Delta}}{i}, 1\right) \le \widehat{\Delta} + \sum_{i=\widehat{\Delta}}^{N} \frac{\widehat{\Delta}}{i} = \widehat{\Delta} + \widehat{\Delta}(H_N - H_{\widehat{\Delta}}) = O(\widehat{\Delta}\log(1/\varepsilon)). \qquad \square$$

Bounding evictions of type (ii) is a bit more involved, because we start off with k unmarked pages (so the naive application of Corollary 2.6 would give us $O(\widehat{\Delta}\log k)$); we need to use the interplay between MARKER and STRIKER to get our bound.

Lemma 2.8. *The expected number of evictions of type (ii) in an epoch is* $O(\widehat{\Delta}\log(1/\varepsilon))$.

Finally, to bound the number of type (iv) evictions, observe that we run **ONESTRIKE** if the counter BAD-STRIKES reaches $\widehat{\Delta}$, or if MARKER returns FAIL in the exploit segment. We claim that both of these events happen with probability at most ε. Now since **ONESTRIKE** makes $O(\widehat{\Delta}/\varepsilon)$ evictions in expectation the expected number of such evictions is at most $\varepsilon \cdot O(\widehat{\Delta}/\varepsilon) = O(\widehat{\Delta})$.

Lemma 2.9. $\Pr[\text{BAD-STRIKES} \ge \widehat{\Delta}] \le \varepsilon$.

Lemma 2.10. *The probability that* MARKER *declares* FAIL *in the exploit segment is at most* $O(\varepsilon)$.

We now summarize the bounds on the four types of evictions. In the explore segment, STRIKER performs at most $2\widehat{\Delta}$ evictions by design of the algorithm, and MARKER performs $O(\widehat{\Delta}\log(1/\varepsilon))$ evictions by Lemma 2.8. In the exploit segment, MARKER performs $O(\widehat{\Delta}\log(1/\varepsilon))$ evictions by Lemma 2.7. Finally, Lemmas 2.9 and 2.10 show that **ONESTRIKE** is called with probability $O(\varepsilon)$, and its cost is $O(\widehat{\Delta}/\varepsilon)$, so its expected contribution is $O(\widehat{\Delta})$ evictions. Combining everything together, the total expected number of evictions in any epoch is at most $O(\widehat{\Delta}\log 1/\varepsilon)$, which proves Theorem 2.4. $\square$

Since we double our guess for $\widehat{\Delta}$ each time, the total expected cost of a phase is $O(\Delta \log 1/\varepsilon)$, thereby proving the claimed competitive ratio for the known-ε case. The algorithm (and analysis) when we do not know the value of ε in advance are conceptually similar to the one above, but there are more details to consider. In particular, we maintain a guess $\hat{\varepsilon}$ for ε, and each time we square this guess. The real problem is that unlike the value of Δ, we do not get a clear signal that we have overestimated the value of ε. Our algorithm therefore needs to infer failure from our algorithm not performing as claimed; we defer this to the supplementary materials.

Comparison with [27]. We now show that under our prediction model, the algorithm given by Wei [27] performs poorly. That algorithm combines RANDOM MARKING and BlindOracle (which in our terminology is called **ONESTRIKE**). Since the BlindOracle algorithm just evicts the page suggested by the oracle, we take $n = k + 1$ pages and construct a sequence of phases. In phase i we request all pages except page i round-robin, and do this k times. The optimal strategy is to evict page i at the start of this phase. But the algorithm follows the oracle blindly, so it will evict random pages due to bad suggestions $1/\varepsilon$ times in expectation before getting a good suggestion and evicting page i. This happens in each phase, giving an expected cost $\Omega(1/\varepsilon)$ times the optimum. Since this is combined with RANDOM MARKING which has an $\Omega(\log k)$ lower bound, by setting $\varepsilon = 1/\log k$ and interleaving phases of the above lower bound sequence with phases of the lower bound for RANDOM MARKING, we get a sequence that causes Wei's algorithm to pay $\Omega(\log k)$ times the optimal cost, whereas our algorithm pays $O(\log(1/\varepsilon)) = O(\log\log k)$ times the optimal cost.

3 Set Cover and Generalized Covering Problems

We now show that for a large class of online covering problems, which includes set cover (and its multiset multicover variants), facility location, and network design problems such as Steiner tree and Steiner forest, we can obtain an $O(1/\varepsilon)$-competitive algorithm with ε-accurate predictions. We first present it for the case of online set cover, and then extend the ideas to more general settings.

3.1 The Set Cover Problem

In the SET COVER problem we are given a collection of m subsets $\mathcal{S} := \{S_1, \dots, S_m\}$ of a universe U containing $|U| = n$ elements. Each set has a cost $c(S) \geq 0$, and we want to pick a subcollection of least cost covering the universe. In the *online version* of the problem, we do not know the set system in advance: an element $e_t \in U$ is revealted at each time t, along with the names of the sets containing e_t. If none of these sets have already been chosen by the algorithm, one must be chosen at this time. Sets once chosen cannot be dropped (so the solutions are monotonically increasing), and the goal is to minimize the total cost of the chosen sets.

Our model of ε-accurate suggestions is the following: at each time, we are given an element e_t, along with a suggested set $s^t \in \mathcal{S}$, with the guarantee that $\Pr[s^t \in \mathsf{OPT} \mid \mathcal{H}^{t-1}] \geq \varepsilon$, where $\mathcal{H}^{t-1}$ is the history of all requests, predictions, and actions taken in previous steps, and OPT is the optimal offline solution.

> **SET-HEDGE**: If e_t is already covered, do nothing. Else let g^t be the minimum-cost set that covers e_t. Choose s^t with probability $c(g^t)/c(s^t)$, and choose g^t otherwise.

Since g^t is the cheapest set covering e_t, we have $c(g^t)/c(s^t) \leq 1$ and hence a valid probability.

Theorem 3.1. *Given ε-accurate predictions, the* **SET-HEDGE** *algorithm is $2/\varepsilon$-competitive.*

This result is tight up to constant factors: we show in the supplementary material that no algorithm for set cover can have competitive ratio better than $O(1/\varepsilon)$ in general. Moreover, we can run this algorithm in parallel with any other online set cover algorithm, say one that is α_{SC}-competitive, to get an algorithm that is $O(\min(1/\varepsilon, \alpha_{SC}))$-competitive.

3.2 Extension to Generalized Covering Problems

The simplicity of the algorithm allows us to extend to very general set of objective functions and constraints, which we call *generalized submodular-cost coverage* (GSCC). Consider the following:

1. The algorithm controls a point $x \in \mathbb{R}^d_{\geq 0}$. The initial point is $x^0 = \mathbf{0}$, the all-zeros vector. We require that x is monotone over time; i.e., $x^{t-1} \leq x^t$.
2. (Covering.) At each time t, a set $K_t \subseteq \mathbb{R}^d_{\geq 0}$ is revealed, and we want that $x \in \cap_{s \leq t} K_s$. We restrict ourselves to sets that are closed under taking component-wise maximums—i.e., $x, y \in K_t \implies (x \vee y) \in K_t$, where $(x \vee y)_i = \max(x_i, y_i)$.
3. (Monotonicity and Submodularity.) The objective function $c : \mathbb{R}^d \to \mathbb{R}_{\geq 0}$ is *monotone*: if $x \leq y$ then $c(x) \leq c(y)$. Moreover, it is *submodular*: $c(x \vee y \vee z) - c(x \vee z) \leq c(x \vee y) - c(x)$.

The ε-accurate suggestion model now says: at each time t, the suggestion $s^t \in K_t$, and moreover $\Pr[s^t \leq x^* \mid \mathcal{H}^{t-1}] \geq \varepsilon$, where x^* is the optimal offline solution. The **COVER-HEDGE** algorithm now extends the **SET-HEDGE** algorithm as follows:

> Let g^t be the minimum-cost increment—i.e., $g^t \leftarrow \arg\min\{c(x^{t-1} \vee g) \mid g \in K_t\}$. Then set $x^t \leftarrow x^{t-1} \vee s^t$ with probability $\frac{c(x^{t-1} \vee g^t) - c(x^{t-1})}{c(x^{t-1} \vee s^t) - c(x^{t-1})}$, and $x^t \leftarrow x^{t-1} \vee g^t$ otherwise.

In general, finding this miminum-cost increment may be computationally hard; we focus on the information-theoretic considerations for now, and defer the computational issues for later.

Theorem 3.2. *Given ε-accurate suggestions, the* **COVER-HEDGE** *algorithm is $2/\varepsilon$-competitive for any generalized covering problem.*

3.3 Applications

Beyond set cover, the general covering formulation above captures several interesting problems:

Network Design: In the *Survivable Network Design* problem (which generalizes *Steiner Tree* and *Steiner Forest*) we are given a set V of n points together with the distances d_{ij} between them $i, j \in V$. The goal is to connect k *pairs* of points $\{(s_\ell, t_\ell) \in V \times V \mid \ell \in [k]\}$ at minimum cost, where each pair (s_ℓ, t_ℓ) comes with a connectivity requirement of r_ℓ disjoint paths between them. This problem can be written as

$$\min \left\{ \sum_{i,j \in V} d_{ij} x_{ij} \mid \sum_{i \in S, j \notin S} x_{ij} \geq r_\ell \ \forall S, \ell \in [k] : s_\ell \in S, t_\ell \notin S; \ x \in \mathbb{R}^{n^2}_{\geq 0} \right\}.$$

This formulation satisfy the Covering, Monotonicity, and Submodularity properties. Moreover, the least-cost augmentation g^t can be obtained in polynomial time by a minimum cost flow algorithm.

Facility Location: This is not a covering program, yet it can be modeled using our framework. Given a point set V on n points and distances d_{ij} between points $i, j \in V$, and *opening costs* $f_i \geq 0$ for each $i \in V$, the goal is to designate some subset F of points as facilities, so that the total cost of $\sum_{i \in F} f_i + \sum_{j \in V} \min_{i \in F} d_{ij}$ is minimized. We can write this using Balinski's MILP formulation:

$$\min \left\{ \sum_{i \in V} f_i y_i + \sum_{j \in V} d_{ij} x_{ij} \mid \sum_i x_{ij} = 1 \ \forall j \in V, \ x_{ij} \leq y_i \ \forall i, j \in V, y \in \mathbb{Z}^n_{\geq 0}, x \in \mathbb{R}^{n^2}_{\geq 0} \right\}.$$

It can be verified that this formulation also satisfies the Covering, Monotonicity, and Submodularity properties, despite the non-covering constraints of the form $x_{ij} \leq y_i$. (Indeed, in the supplementary material, we show that the coordinate-wise maximum covering property allows us to go beyond standard covering programs.) Moreover, the least-cost augmentation g^t can be obtained in polynomial time using a simple greedy algorithm.

Covering Mixed-Integer Linear Programs (MILPs): given non-negative $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and $c, u \in \mathbb{R}^n$, and a subset $I \subseteq [n]$, we want to solve

$$\min\{c^\intercal x \mid Ax \geq b, 0 \leq x \leq u, x_i \in \mathbb{Z} \, \forall i \in I, x_i \in \mathbb{R} \forall i \notin I\}.$$

Each constraint is a half-space, and it can be verified that this formulation satisfies the Covering, Monotonicity, and Submodularity properties. However, since this problem is NP-hard in general, the least-cost augmentation g^t can be computed efficiently only in some cases such as the network design problems described above.

4 Conclusions

In this paper, we presented online algorithms augmented with ε-accurate predictions for several classic problems such as caching, set cover, facility location, Steiner tree, and generalizations. Can we show a a $\operatorname{poly}\log(1/\varepsilon)$ competitiveness, or even $f(\varepsilon)$ competitiveness for the k-server problem, which is an extension of our results for caching? Another direction would be to combine the ε-accuracy prediction model studied in this paper with some measure η of total prediction error namely, design online algorithms augmented by *probabilistically approximately correct* (or PAC) predictions. Finally, a third direction of future work would be a thorough experimental analysis of our framework and algorithms, to empirically compare the performance of our work with that of other learning-augmented algorithms.

Acknowledgments

AG and BS were supported in part by NSF awards CCF-1955785, CCF-2006953, and CCF-2224718. DP and KS were supported in part by NSF grants CCF-1750140 (CAREER) and CCF-1955703 and ARO grant W911NF2110230.

References

[1] Keerti Anand, Rong Ge, Amit Kumar, and Debmalya Panigrahi. Online algorithms with multiple predictions. In *International Conference on Machine Learning*. PMLR, 2022.

[2] Antonios Antoniadis, Christian Coester, Marek Elias, Adam Polak, and Bertrand Simon. Online metric algorithms with untrusted predictions. In *International Conference on Machine Learning*, pages 345–355. PMLR, 2020.

[3] Antonios Antoniadis, Themis Gouleakis, Pieter Kleer, and Pavel Kolev. Secretary and online matching problems with machine learned advice. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *NeurIPS 2020*, 2020.

[4] Yossi Azar, Andrei Z. Broder, and Mark S. Manasse. On-line choice of on-line algorithms. In Vijaya Ramachandran, editor, *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas, USA*, pages 432–440. ACM/SIAM, 1993.

[5] Étienne Bamas, Andreas Maggiori, Lars Rohwedder, and Ola Svensson. Learning augmented energy minimization via speed scaling. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *NeurIPS 2020*, 2020.

[6] Étienne Bamas, Andreas Maggiori, and Ola Svensson. The primal-dual method for learning augmented algorithms. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *NeurIPS 2020*, 2020.

[7] Nikhil Bansal, Christian Coester, Ravi Kumar, Manish Purohit, and Erik Vee. Learning-augmented weighted paging. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 67–89. SIAM, 2022.

[8] Laszlo A. Belady. A study of replacement algorithms for virtual-storage computer. *IBM Syst. J.*, 5(2):78–101, 1966.

[9] Aditya Bhaskara, Ashok Cutkosky, Ravi Kumar, and Manish Purohit. Online learning with imperfect hints. In *International Conference on Machine Learning*, pages 822–831. PMLR, 2020.

[10] Paul Dütting, Silvio Lattanzi, Renato Paes Leme, and Sergei Vassilvitskii. Secretaries with advice. In Péter Biró, Shuchi Chawla, and Federico Echenique, editors, *EC '21: The 22nd ACM Conference on Economics and Computation, Budapest, Hungary, July 18-23, 2021*, pages 409–429. ACM, 2021.

[11] Jon Ergun, Zhili Feng, Sandeep Silwal, David P. Woodruff, and Samson Zhou. Learning-augmented k-means clustering. *CoRR*, abs/2110.14094, 2021.

[12] Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel Dominic Sleator, and Neal E. Young. Competitive paging algorithms. *J. Algorithms*, 12(4):685–699, 1991.

[13] Chen-Yu Hsu, Piotr Indyk, Dina Katabi, and Ali Vakilian. Learning-based frequency estimation algorithms. In *International Conference on Learning Representations*, 2019.

[14] Piotr Indyk, Frederik Mallmann-Trenn, Slobodan Mitrović, and Ronitt Rubinfeld. Online page migration with ml advice. *arXiv preprint arXiv:2006.05028*, 2020.

[15] Zhihao Jiang, Debmalya Panigrahi, and Kevin Sun. Online algorithms for weighted paging with predictions. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[16] Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1859–1877. SIAM, 2020.

[17] Thomas Lavastida, Benjamin Moseley, R. Ravi, and Chenyang Xu. Learnable and instance-robust predictions for online matching, flows and load balancing, 2020.

[18] Thodoris Lykouris and Sergei Vassilvtiskii. Competitive caching with machine learned advice. In *International Conference on Machine Learning*, pages 3296–3305. PMLR, 2018.

[19] Mohammad Mahdian, Hamid Nazerzadeh, and Amin Saberi. Allocating online advertisement space with unreliable estimates. In Jeffrey K. MacKie-Mason, David C. Parkes, and Paul Resnick, editors, *Proceedings 8th ACM Conference on Electronic Commerce (EC-2007), San Diego, California, USA, June 11-15, 2007*, pages 288–294. ACM, 2007.

[20] Andrés Muñoz Medina and Sergei Vassilvitskii. Revenue optimization with approximate bid predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1856–1864, 2017.

[21] Michael Mitzenmacher. A model for learned bloom filters, and optimizing by sandwiching. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 462–471, 2018.

[22] Michael Mitzenmacher. Scheduling with predictions and the price of misprediction. In *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[23] Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ML predictions. In *Advances in Neural Information Processing Systems*, pages 9661–9670, 2018.

[24] Dhruv Rohatgi. Near-optimal bounds for online caching with machine learned advice. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1834–1845. SIAM, 2020.

[25] Dhruv Rohatgi. Near-optimal bounds for online caching with machine learned advice. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1834–1845. SIAM, 2020.

[26] Daniel D Sleator and Robert E Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[27] Alexander Wei. Better and simpler learning-augmented online caching. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 4, where we list the problems we consider and directions for future work.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] Most proofs are in the supplementary material.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [N/A]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [N/A]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [N/A]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Organization. Appendix A and Appendix B contain the missing proofs for Section 2, namely:

- Appendix A.1 contains the proofs for the **ONESTRIKE** algorithm from Section 2.1.
- Appendix A.2 contains the proofs required for Section 2.2.
- Appendix B describes and analyzes the **TWOSTRIKESUA** algorithm, which is an extension of the **TWOSTRIKES** to the case where the ε is not known.
- Appendix B.2 proves the $\Omega(\log(1/\varepsilon))$ randomized lower bound.

Appendix C presents the missing proofs for Section 3, namely:

- Appendix C.1 contains the proofs of Theorem 3.1 and Theorem 3.2.
- Appendix C.2 contains the proof of Theorem 3.2 for the case where the objective function is separable.
- Appendix C.3 shows a syntactic characterization of generalized covering problems over half-spaces, which allows the extension of our results to problems such as facility location, as mentioned in Section 3.3.
- Appendix C.4 proves the $\Omega(1/\varepsilon)$ randomized lower bound.

A Missing Proofs from Section 2

First, we prove Lemma 2.1 (Cache Reset Overhead):

Proof of Lemma 2.1. Every page that is being brought into the cache during the cache reset must have been swapped out in the previous phase after serving its request. The lemma follows. ☐

A.1 Proofs for the ONESTRIKE algorithm

Next, we prove Theorem 2.2, the competitive ratio of the **ONESTRIKE** algorithm.

First, we introduce some helper lemmas. The first lemma is a standard property of caching that helps lower bound the cost of any feasible solution:

Lemma A.1 (Lower Bound on the Optimal Cost)**.** *The number of page swaps in any feasible solution (and therefore, in an optimal solution) is at least $\sum_{i\geq 2} {}^{\Delta_i}/_2$, where Δ_i is the number of clean pages requested in phase i.*

Proof. For any $i \geq 2$, a total of $k + \Delta_i$ distinct pages are requested across phases $i-1$ and i, which means that any feasible solution must perform at least Δ_i page swaps across these two phases. The lemma follows by summing over all the values of i. ☐

At any time t, recall that a *good eviction* is one where the evicted page is not requested in the current phase after time t. Once evicted, such a page does not return to the cache for the rest of the phase. This allows us to relate good evictions to clean requests:

Lemma A.2 (Good Evictions vs. Clean Requests)**.** *The number of good evictions in phase i is at most Δ_i, the number of clean pages requested in the phase.*

Proof. Note that a total of $k + \Delta_i$ pages are requested across phases $i-1$ and i. After Δ_i good evictions in phase i, all subsequent requests in phase i are for the remaining k pages. The cache can hold all these k pages, so there cannot be any further page swaps in phase i. ☐

Finally, we relate good evictions to FIF pages:

Lemma A.3 (Good Evictions vs. FIF Pages)**.** *At any time t, if the cache contains k pages and the* FIF *page is evicted to serve a request for a page not currently in the cache, then this is necessarily a good eviction.*

Proof. Excluding the currently requested page, at most $k-1$ other pages can be requested after time t in the current phase. Now, note that the $k-1$ pages in the cache other than the FIF page are requested before the FIF page after time t. The lemma follows. $\square$

Now, we are ready to prove Theorem 2.2:

Proof of Theorem 2.2. By Lemma 2.1 and Lemma A.1, it suffices to prove that the expected number of page evictions of the algorithm in a given phase is at most Δ/ε, where Δ is the number of clean pages requested in the phase. By Lemma A.3 and the ε-accuracy condition, every eviction is a good eviction with probability at least ε. It follows by Lemma A.2 that the expected number of page evictions made by the **ONESTRIKE** algorithm in the current phase is at most Δ/ε. $\square$

A.2 Proofs for the TWOSTRIKES algorithm

Now we provide the missing proofs for establishing the competitive ratio of the **TWOSTRIKES** algorithm.

Proof of Lemma 2.8. Suppose $p_1, p_2, \ldots, p_k$ are the unmarked pages in the cache at the beginning of the epoch in reverse chronological order of their first requests in this segment. The pages that are not requested at all are placed at the beginning of this sequence, but can be arbitrarily ordered relative to each other. As in Lemma 2.7, the number of clean requests for pages is at most $\widehat{\Delta}$, else the epoch ends. So by Corollary 2.6, the expected number of evictions of type (ii) among pages $p_1, p_2, \ldots, p_{2N}$ is again at most $O(\widehat{\Delta}\log(1/\varepsilon))$, using the same argument as in Lemma 2.7.

But what about evictions of type (ii) among the remaining pages $p_{2N+1}, \ldots, p_k$? Suppose at time t, the first request for page p_{i_t} causes a type (ii) eviction of a page in $p_{2N+1}, \ldots, p_k$. At this point in time, the cache must be full (else we would not have evicted a page), so the STRIKER algorithm is active at the end of time t. By Lemma 2.5, the probability that some page p_j has been evicted by MARKER by time t is at most $\widehat{\Delta}/i_t \le \widehat{\Delta}/2N = \varepsilon^2/2$ for any $j \in \{2N+1, \ldots, k\}$. By the union bound, none of the pages p_j for $j \in \{2N+1, \ldots, k\}$ satisfying $j \in [i_t - 1/\varepsilon^2, i_t)$ have been evicted by MARKER by time t, with probability at least $1/2$.

Suppose this event happens. Since STRIKER is now active, one of the following events must happen in the next $1/\varepsilon^2$ requests:

- $\mathcal{E}_a$: There is a type (i) eviction in these $1/\varepsilon^2$ requests, or
- $\mathcal{E}_b$: The STRIKER procedure is active for all these $1/\varepsilon^2$ requests.

In either case, we can charge to the corresponding event. By the first termination condition for the explore segment, event $\mathcal{E}_a$ happens at most $2\widehat{\Delta}$ times. By second termination for the explore segment, event $\mathcal{E}_b$ happens at most $\widehat{\Delta}$ times. Putting these together, we conclude that the expected number of type (ii) evictions for pages $p_{2N+1}, \ldots, p_k$ is $O(\widehat{\Delta})$, and the total expected type (ii) evictions are at most $O(\widehat{\Delta}\log(1/\varepsilon))$. $\square$

Proof of Lemma 2.9. We show that $\mathbb{E}[\text{BAD-STRIKES}] \le \varepsilon\widehat{\Delta}$, whereupon Markov's inequality implies the claimed probability bound. Indeed, fix any page p, and consider the expected number of times that the counter BAD-STRIKES is incremented due to p. For any such increment, p must be predicted twice when it is not the FIF page in a cache containing k pages. This is because if p is the FIF page in any of these predictions, then by Lemma A.3, p is a good eviction. By the termination condition for STRIKER, it is active mode for at most $N = \widehat{\Delta}/\varepsilon^2$ steps. So the expected contribution of p to BAD-STRIKES is at most

$$\sum_{i\ge 1} i \cdot \binom{N}{2i} \cdot \frac{1}{k^{2i}} \le \sum_{i\ge 1} \frac{1}{2^i} \cdot \left(\frac{N}{k}\right)^{2i} \le \left(\frac{N}{k}\right)^2 < \left(\frac{\widehat{\Delta}}{\varepsilon^2 k}\right)^2 \quad \text{since } N = \frac{\widehat{\Delta}}{\varepsilon^2}.$$

By linearity of expectation, the expected value of the counter BAD-STRIKES is at most

$$\left(\frac{\widehat{\Delta}}{\varepsilon^2 k}\right)^2 \cdot k = \widehat{\Delta} \cdot \frac{1}{\varepsilon^4} \cdot \frac{\widehat{\Delta}}{k} < \varepsilon\widehat{\Delta}, \qquad \text{since } \frac{\widehat{\Delta}}{k} < \varepsilon^5.$$

$\square$

Relaxing the prediction model. The proof of Lemma 2.9 assumes that the probability that a page p is predicted when it is not the FIF page in a cache containing k pages is at most $1/k$. Now we show that the lemma still holds assuming this probability is at most $1/k\varepsilon^c$ for any $c > 0$. The proof is essentially the same, but we need to change a few constants. Now, the expected contribution of p to BAD-STRIKES is at most

$$\sum_{i \geq 1} i \cdot \binom{N}{2i} \cdot \frac{1}{(k\varepsilon^c)^{2i}} \leq \sum_{i \geq 1} \frac{1}{2^i} \cdot \left(\frac{N}{k\varepsilon^c}\right)^{2i} \leq \left(\frac{N}{k\varepsilon^c}\right)^2.$$

By linearity of expectation, it suffices to show that this quantity is at most $\varepsilon\widehat{\Delta}/k$. This holds if we set $N = \widehat{\Delta}/\varepsilon^d$ and assume $\widehat{\Delta} < \varepsilon^{2c+2d+1}k$. (In the proof of Lemma 2.9, we assumed $c = 0$ and set $d = 2$.) Under these modifications, all of the other proofs still hold; this lemma was the only one relying on an upper bound on the probability of a page being incorrectly predicted as FIF.

Proof of Lemma 2.10. We call a strike *good* if it applies to a page that will not be requested in the rest of the phase. Let S denote the number of pages struck by at least one good strike during the explore segment. It suffices to show $S \geq \widehat{\Delta}$ with probability $1 - O(\varepsilon)$. If this happens, then at the beginning of the exploit phase, there are at least $\widehat{\Delta}$ good pages that have either been struck or evicted, hence we would not run out of unmarked pages before the epoch ends.

Suppose the explore segment ends due to the first termination condition: STRIKER has made $2\widehat{\Delta}$ evictions. By Lemma 2.9, with probability $1 - \varepsilon$, at most $\widehat{\Delta}$ of these evictions are bad, so at least $\widehat{\Delta}$ of them are good, as desired.

Now suppose the explore segment ends due to the second termination condition: STRIKER has been in active mode for $N = \widehat{\Delta}/\varepsilon^2$ requests. Whenever STRIKER is in active mode, it strikes the predicted page (and possibly evicts it). Conditioned on the past, each strike is good with probability at least ε, so the expected number of good strikes is at least $\varepsilon N = \widehat{\Delta}/\varepsilon$. Some of these good strikes apply to the same page, but STRIKER evicts any page with two strikes, so among the good strikes, at most $\widehat{\Delta}$ of them apply to pages that already have a good strike. Thus, we have $\mathbb{E}[S] \geq \widehat{\Delta}/\varepsilon - \widehat{\Delta}$.

We now finish with a concentration bound; we want to show that $S < \widehat{\Delta}$ with probability $O(\varepsilon)$. Letting $\delta = 1 - 1/(1/\varepsilon - 1)$ and $\mu = \widehat{\Delta}/\varepsilon - \widehat{\Delta}$, by a standard Chernoff bound, we have

$$\Pr(S < \widehat{\Delta}) = \Pr(S < (1-\delta)\mu) \leq \exp(-\delta^2\mu/2) \leq \exp((4 - 1/\varepsilon)/2) = O(\varepsilon),$$

assuming ε is sufficiently small. □

B Caching with ε-Accurate Predictions for Unknown ε

In this section, we formally describe the **TWOSTRIKESUA** algorithm for unknown accuracy ε. At a high level, the algorithm is similar to the one described for known ε: each phase begins with a cache reset to ensure that the pages in the cache are precisely those requested in the previous phase. Within a phase, there is an outer loop that iterates over *epochs*, and an inner loop that iterates over *blocks*. Each epoch has a fixed value of $\widehat{\Delta}$, which is initialized to 1 at the beginning of the phase. When the number of clean pages requested in an epoch exceeds $\widehat{\Delta}$, we end the epoch, perform a cache reset, and start a new epoch after doubling the value of $\widehat{\Delta}$. Each block fixes the value of $\hat{\varepsilon}$, which is initialized to $1/2$ (or any constant strictly less than 1) at the beginning of an epoch. The condition for ending a block and starting a new one is more complicated than that for an epoch because there is no direct way for the algorithm to detect if $\varepsilon < \hat{\varepsilon}$. We will describe this condition later as part of the internals of a block.

Within a block starts, **TWOSTRIKESUA** first checks if $\hat{\varepsilon} \leq (\widehat{\Delta}/k)^{1/5}$; if so, it simply runs randomized marking in the rest of the current epoch. Else if $\hat{\varepsilon} > (\widehat{\Delta}/k)^{1/5}$, the block now is partitioned into an *explore segment* followed by an *exploit segment*. In the explore segment, the algorithm makes $\widehat{\Delta}^*$ good evictions of pages that have been predicted twice (for some $\widehat{\Delta}^* \leq \widehat{\Delta}$), and also learns an additional candidate set of pages that contains at least $\widehat{\Delta} - \widehat{\Delta}^*$ pages which would be good evictions. In the following exploit segment, the algorithm runs randomized marking on these candidate pages to actually make $\widehat{\Delta} - \widehat{\Delta}^*$ good evictions.

The algorithm requires the procedures MARKER and STRIKER. Both are identical to their respective counterparts from Section 2, where ε was known. The explore and exploit segments are largely identical to their counterparts as well, with minor changes that we now describe.

The Explore Segment. The explore segment is almost identical to its counterpart from Section 2. The only difference is the following: If BAD-STRIKES (initially 0 at the beginning of each block) reaches $\widehat{\Delta}$, then we end the current block, perform a cache reset, and start a new block after squaring the value of $\hat{\varepsilon}$. The details are in Algorithm 5.

Algorithm 5: Explore Segment

---

foreach *time* t **do**
- **let** p_t be the requested page and s_t the predicted page at time t
- **if** p_t *is strike-evicted (i.e.,* STRIKE$(p_t) = 2$*)* **then** increment BAD-STRIKES
- **if** BAD-STRIKES $= \widehat{\Delta}$ **then** square $\hat{\varepsilon}$ and start a new block **else** set STRIKE$(p_t) \leftarrow 0$
- **call** STRIKER
- **call** MARKER
- **terminate** explore segment if STRIKER evicts pages $\geq 2\widehat{\Delta}$ times, or STRIKER in active mode for $N := \widehat{\Delta}/\hat{\varepsilon}^2$ requests

---

The Exploit Segment. The exploit segment is also almost identical to its counterpart from Section 2. As was the case for the explore segment, the only difference is that if BAD-STRIKES reaches $\widehat{\Delta}$, then we end the current block, perform a cache reset, and start a new block after squaring the value of $\hat{\varepsilon}$. The details are in Algorithm 6.

Algorithm 6: Exploit Segment

---

let $S \leftarrow$ striked pages from the explore segment
mark all pages in the cache not in S; pages in S are unmarked
if p_t *requested at time* t **then**
- **if** p_t *is strike-evicted (i.e.,* STRIKE$(p_t) = 2$*)* **then** increment BAD-STRIKES
- **if** BAD-STRIKES $= \widehat{\Delta}$ **then** square $\hat{\varepsilon}$ and start a new block **else** set STRIKE$(p_t) \leftarrow 0$
- **call** MARKER **if** MARKER *returns* FAIL **then** square $\hat{\varepsilon}$ and start a new block

---

Note that the **TWOSTRIKESUA** algorithm, unlike **TWOSTRIKES** from Section 2, does not call **ONESTRIKE** at any point. Instead, whenever BAD-STRIKES reaches $\widehat{\Delta}$, it squares $\hat{\varepsilon}$ and starts a new block.

B.1 Competitive ratio of the TWOSTRIKESUA algorithm

In any block, there are three types of page evictions in the randomized algorithm:

(i) evictions performed by STRIKER in the explore segment,
(ii) evictions performed by MARKER in the explore segment, and
(iii) evictions performed by MARKER in the exploit segment

We can bound all three types using the same proofs from from Section 2, except we replace ε with $\hat{\varepsilon}$. Furthermore, the page swaps during the cache reset at the beginning of a block can be charged to the page evictions in the previous block. So, we have arrived at the following bound:

Lemma B.1. *The expected number of evictions in a block is* $O(\widehat{\Delta} \cdot \log(1/\hat{\varepsilon}))$.

Now we address the main difference between **TWOSTRIKESUA** and **TWOSTRIKES**, which is the block structure due to our guess $\hat{\varepsilon}$ of ε. At a high level, Lemma B.1 allows us to bound the cost incurred in blocks where $\hat{\varepsilon} > \varepsilon$, because squaring $\hat{\varepsilon}$ creates a geometric series over these blocks. For blocks where $\hat{\varepsilon} < \varepsilon$, we now show that the probability of starting a new block is at most $2\hat{\varepsilon}$. This will in fact allow us to bound the total cost over these blocks by $O(\widehat{\Delta})$.

Lemma B.2. $\Pr[\text{BAD-STRIKES} \geq \widehat{\Delta}] \leq \hat{\varepsilon}$.

Proof. The proof is identical to that of Lemma 2.9, except with $\hat{\varepsilon}$ instead of ε. ☐

Lemma B.3. *In any block where* $\hat{\varepsilon} \leq \varepsilon$*, the probability that* MARKER *declares* FAIL *is* $O(\hat{\varepsilon})$.

Proof. The proof is identical to that of Lemma 2.10, except with $\hat{\varepsilon}$ instead of ε. More specifically, suppose the explore segment ends due to the first termination condition (i.e., STRIKER has made $2\widehat{\Delta}$ evictions). Then by Lemma B.2, with probability $1-\hat{\varepsilon}$, at most $\widehat{\Delta}$ of these evictions are bad so at least $\widehat{\Delta}$ of them are good, as desired.

Now suppose the explore segment ends due to the second termination condition: STRIKER has been in active mode for $N = \widehat{\Delta}/\hat{\varepsilon}^2$ requests. Whenever STRIKER is in active mode, it strikes the predicted page (and possibly evicts it). Conditioned on the past, each strike is good with probability at least ε, so the expected number of good strikes is at least $\varepsilon N = \varepsilon\widehat{\Delta}/\hat{\varepsilon}^2 \geq \widehat{\Delta}/\hat{\varepsilon}$. The rest of the proof is identical to that of Lemma 2.10, except with $\hat{\varepsilon}$ instead of ε. ☐

Lemma B.4. *If $\hat{\varepsilon} \leq \varepsilon$, then the probability that the algorithm starts a new block is at most $O(\hat{\varepsilon})$.*

Proof. There are two ways for the algorithm to start a new block: if BAD-STRIKES $= \widehat{\Delta}$, or MARKER returns FAIL due to a request to a page not in the cache when the cache has k marked pages. By Lemma B.2 the former occurs with probability at most $\hat{\varepsilon}$, and by Lemma B.3, the latter occurs with probability $O(\hat{\varepsilon})$. The lemma follows by a union bound. ☐

Finally, we are ready to prove the competitive ratio of the algorithm.

Lemma B.5. *The competitive ratio of* **TWOSTRIKESUA** *is $O(\log(1/\varepsilon))$.*

Proof. We need to show that the expected number of evictions in a phase is $O(\Delta \log(1/\varepsilon))$. Since $\widehat{\Delta} \leq 2\Delta$ in each epoch and doubles between consecutive epochs in the same phase, it suffices to show that the expected number of page swaps in an epoch is $O(\widehat{\Delta}\log(1/\varepsilon))$. By Lemma B.1, the expected number of evictions in all blocks of a single epoch satisfying $\hat{\varepsilon} \geq \varepsilon^2$ is bounded by $O(\widehat{\Delta}\log(1/\varepsilon))$. By Lemma B.4, for any block with $\hat{\varepsilon} < \varepsilon$, the probability that the algorithm starts a new block is $O(\hat{\varepsilon})$. Thus, the expected number of swaps due to all blocks satisfying $\hat{\varepsilon} < \varepsilon^2$ is at most

$$O(\widehat{\Delta}) \cdot \sum_{i=1}^{\infty} \varepsilon^{2^i} \cdot \log \frac{1}{\varepsilon^{2^{i-1}}} \leq O(\widehat{\Delta}) \cdot \sum_{i=1}^{\infty} (2\varepsilon^2)^i \log \frac{1}{\varepsilon} = O(\widehat{\Delta}).$$ ☐

B.2 Caching Lower Bound

Theorem B.6. *Any (randomized) algorithm for caching with ε-accurate suggestions is $\Omega(\log(1/\varepsilon))$-competitive.*

In our lower bound construction, we will consider the following prediction model: at each time t, with probability ε (independently of the past), the algorithm receives a good prediction (i.e., it is told the FIF page in its cache). With probability $1-\varepsilon$, it does not receive any prediction at all. This prediction model is stronger than ε-accurate suggestions, because any algorithm, given the former, can generate the latter by choosing a page uniformly at random from its cache in the $1-\varepsilon$ case. Thus, a lower bound in this model also holds for the ε-accurate prediction model.

Proof. Let $n = k+1$ and consider the sequence that generates each request uniformly at random among the $k+1$ pages. Set $\varepsilon = \frac{1}{k \ln k}$. A *phase* is defined as a maximal contiguous subsequence of requests that contains exactly k distinct pages. Consider an arbitrary phase and let X be the random variable denoting its length. Note that $\mathbb{E}[X]$ is the expected number of times we would need to sample from a uniform random variable over a space of size $k+1$ until we obtain k distinct outcomes. By a slight modification of the coupon collector analysis, we can show that $X = \Theta(k \log k)$ with constant probability.

Now partition the input into groups of 3 consecutive phases (starting at the beginning), and consider any such group. Notice that every phase contains all but one of the $k+1$ pages, and that missing page is the first page requested in the subsequent phase. So for each page p suggested to the algorithm during the first phase, p is either requested in the second phase, or p is the first request of the third phase.

Furthermore, the probability that the algorithm does not receive any good suggestions during the second or third phase is $(1-\varepsilon)^{\Theta(k \log k)} = \Theta(1)$. So overall in this group, with constant probability, the algorithm only receives good suggestions in the first phase, and these suggestions do not reveal anything about the third phase (except for possibly the first request). So when serving the third phase of the group, the algorithm incurs a miss at every step with probability $1/(k+1)$, for an expected cost of $\Omega(\log k)$ to serve the group. On the other hand, the optimal solution can serve each phase by evicting the single page that does not appear in that phase, thereby incurring $O(1)$ cost per group. $\square$

C Missing Proofs from Section 3

C.1 Analysis of Set-Hedge and Cover-Hedge

Proof of Theorem 3.1. Fix any instance of online set cover and an optimal solution OPT for it. Let ALG^t be the solution of the **Set-Hedge** algorithm after covering e_t. Consider the potential function

$$\Phi^t := (2/\varepsilon) \cdot \textstyle\sum_{S \in \mathsf{OPT} \setminus \mathsf{ALG}^t} c(S),$$

the cost of optimal sets not already picked by the algorithm by time t (and scaled by $2/\varepsilon$). If $\Delta\Phi^t$ and $\Delta c(\mathsf{ALG}^t)$ denote the change in potential and the algorithm's cost due to sets picked at time t, we claim

$$\mathbb{E}[\Delta\Phi^t + \Delta c(\mathsf{ALG}^t)] \leq 0; \tag{1}$$

and because Φ^t is always non-negative, this shows that the total expected cost of sets picked by the algorithm is at most $\Phi^0 = (2/\varepsilon) \cdot c(\mathsf{OPT})$.

To prove (1), condition on $\mathcal{H}^{t-1}$: if e_t is already covered by ALG^{t-1} we have $\Delta\Phi^t = \Delta c(\mathsf{ALG}^t) = 0$. Else if e_t is not previously covered, $\Delta c(\mathsf{ALG}^t) = \frac{c(g^t)}{c(s^t)} \cdot c(s^t) + (1 - \frac{c(g^t)}{c(s^t)}) \cdot c(g^t) \leq 2\, c(g^t)$. The potential change is

$$\mathbb{E}[\Delta\Phi^t \mid \mathcal{H}^{t-1}] = -(2/\varepsilon) \cdot \sum_{S \in \mathsf{OPT} \setminus \mathsf{ALG}^{t-1} : e_t \in S} c(S) \cdot \Pr[s^t = S \mid \mathcal{H}^{t-1}] \cdot \frac{c(g^t)}{c(S)}.$$

By the ε-accuracy condition, the probability values sum to at least ε. Hence $\mathbb{E}[\Delta\Phi^t \mid \mathcal{H}^{t-1}] \leq -(2/\varepsilon) \cdot c(g^t) \cdot \varepsilon$. So (1) holds irrespective of $\mathcal{H}^{t-1}$ (and hence unconditionally). $\square$

Proof of Theorem 3.2. The proof closely mimics that of Theorem 3.1; the potential is now

$$\Phi^t := (2/\varepsilon) \cdot \left[c(x^* \vee x^t) - c(x^t)\right].$$

This is clearly non-negative (by monotonicity), and starts off at $\Phi^0 \leq (2/\varepsilon) \cdot c(x^*)$. Again it suffices to show that $\mathbb{E}[\Delta\Phi^t + \Delta c(\mathsf{ALG}^t)] \leq 0$. First, a direct calculation shows

$$\mathbb{E}[\Delta\, c(\mathsf{ALG}^t)] \leq 2 \cdot (c(x^{t-1} \vee g^t) - c(x^{t-1})).$$

To bound the expected potential decrease, for any vector v, setting $x^t \leftarrow x^{t-1} \vee v$ decreases the potential by

$$(2/\varepsilon) \cdot \left[c(x^* \vee x^{t-1}) - c(x^{t-1}) - (c(x^* \vee x^{t-1} \vee v) - c(x^{t-1} \vee v))\right],$$

which is non-negative by submodularity. Let $\mathcal{E}^t$ be the event that $p^t \leq x^*$, and let $\mathcal{F}^t$ be the event that we make the choice of setting $x^t \leftarrow x^{t-1} \vee p^t$. If both events happen, we have $c(x^* \vee (x^{t-1} \vee p^t)) = c(x^* \vee x^{t-1})$, and the expression above becomes

$$(2/\varepsilon) \cdot (c(x^{t-1} \vee p^t) - c(x^{t-1})).$$

Hence

$$\mathbb{E}[\Delta\Phi^t \mid \mathcal{H}^{t-1}] \leq -(2/\varepsilon) \cdot (c(x^{t-1} \vee p^t) - c(x^{t-1})) \cdot \Pr[\mathcal{E}^t \cap \mathcal{F}^t \mid \mathcal{H}^{t-1}].$$

Moreover, the two events are independent. By the ε-accuracy property, we know that $\Pr[\mathcal{E}^t \mid \mathcal{H}^{t-1}] \geq \varepsilon$. Finally, substituting the probability of $\mathcal{F}^t$ from the algorithm description, we get $\mathbb{E}[\Delta\Phi^t + \Delta c(\mathsf{ALG}^t) \mid \mathcal{H}^{t-1}] \leq 0$, which proves the claim. $\square$

C.2 Separable Objective Functions

We consider the same setting as Section 3.2, with the only difference being that instead of requiring that the objective function $c : \mathbb{R}^d \to \mathbb{R}_{\geq 0}$ is submodular, in this section, we require it to be separable. In particular, we assume there exist functions $c_1, \dots, c_d : \mathbb{R} \to \mathbb{R}_{\geq 0}$ such that every c_i is monotone, and for all $x \in \mathbb{R}^d$, $c(x) = \sum_i c_i(x_i)$. Note that this implies c itself is monotone.

Recall the **COVER-HEDGE** algorithm, given a subset K_t and suggestion p^t:

> Let g^t be the minimum-cost increment—i.e., $g^t \leftarrow \arg\min\{c(x^{t-1} \vee g) \mid g \in K_t\}$. Then set $x^t \leftarrow x^{t-1} \vee p^t$ with probability $\frac{c(x^{t-1} \vee g^t) - c(x^{t-1})}{c(x^{t-1} \vee p^t) - c(x^{t-1})}$, and $x^t \leftarrow x^{t-1} \vee g^t$ otherwise.

We claim that it is still $2/\varepsilon$-competitive in this setting. Note that when c is separable, for any vectors $x, v \in \mathbb{R}^d$, we have $c(x \vee v) = \sum_i c_i(x_i \vee v_i)$.

Theorem C.1. *Given ε-accurate suggestions, the* **COVER-HEDGE** *algorithm is $2/\varepsilon$-competitive for any generalized covering problem where the objective is separable.*

Proof. Like that of Theorem 3.2, this proof closely mimics that of Theorem 3.1. The potential is the same as the one used to prove Theorem 3.2:

$$\begin{aligned}
\Phi^t &:= \left(\frac{2}{\varepsilon}\right) \cdot \left[c(x^* \vee x^t) - c(x^t)\right] \\
&= \left(\frac{2}{\varepsilon}\right) \cdot \left[\sum_{i=1}^{d} c_i(x_i^* \vee x_i^t) - c_i(x_i^t)\right] \\
&= \left(\frac{2}{\varepsilon}\right) \cdot \left[\sum_{i : x_i^* \geq x_i^t} c_i(x_i^*) - c_i(x_i^t)\right].
\end{aligned}$$

This is non-negative (by monotonicity) and starts off at $\Phi^0 \leq (2/\varepsilon) \cdot c(x^*)$. Again it suffices to show that $\mathbb{E}[\Delta\Phi^t + \Delta c(\mathsf{ALG}^t)] \leq 0$. To bound the expected potential decrease, for any vector v, setting $x^t \leftarrow x^{t-1} \vee v$ decreases the potential by

$$\begin{aligned}
&(2/\varepsilon) \cdot \left[c(x^* \vee x^{t-1}) - c(x^{t-1}) - (c(x^* \vee x^{t-1} \vee v) - c(x^{t-1} \vee v))\right] \\
&= (2/\varepsilon) \cdot \left[\sum_{i : x_i^* \geq x_i^{t-1}} c_i(x_i^*) - c_i(x_i^{t-1}) - \left(\sum_{i : x_i^* \geq x_i^{t-1} \vee v_i} c_i(x_i^*) - c_i(x_i^{t-1} \vee v_i)\right)\right],
\end{aligned}$$

which is non-negative since $x_i^* \geq x_i^{t-1} \vee v_i$ implies $x_i^* \geq x_i^{t-1}$, and every c_i is monotone. The remainder of the proof is identical to that of Theorem 3.2. $\square$

C.3 Properties of Generalized Covering Problems

The component-wise maximum property is satisfied for covering programs (i.e., where the constraints are $a^\intercal x \geq b$, and all entries of a, b are non-negative). However, we now show that the property is also satisfied when a has one negative coordinate.

Fact C.2. *Suppose $a \in \mathbb{R}^n, b \in \mathbb{R}$, and consider the set $K = \{x : a^\intercal x \geq b, x \geq 0\}$. Assume K contains at least one non-zero vector. Then K is closed under max if and only if a has at most one negative coordinate.*

Proof. We first prove the backward direction. Let $x, y \in \mathbb{R}^n$ satisfy $a^\intercal x \geq b$ and $a^\intercal y \geq b$, let $z = x \vee y$. If a has no negative coordinates, then clearly $a^\intercal z \geq b$. Now suppose there exists i such that $a_i < 0$. Then we have

$$-a_i z_i + b \leq \max\left(\sum_{j \neq i} a_j x_j, \sum_{j \neq i} a_j y_j\right) \leq \sum_{j \neq i} a_j z_j,$$

which implies $a^\intercal z \geq b$, as desired.

For the forward direction, if $n = 1$, the statement is trivial. So for contradiction, without loss of generality, we assume $a_1, a_2 < 0$. Let $\alpha_i = -a_i$, $N = \{i : a_i < 0\}$, and $P = \{i : a_i > 0\}$. Now the condition $a^\intercal x \geq b$ is equivalent to

$$\alpha_1 x_1 + \alpha_2 x_2 + \sum_{i \in N \setminus \{1,2\}} \alpha_i x_i + b \leq \sum_{i \in P} a_i x_i. \tag{2}$$

Consider the following cases:

1. $P = \emptyset$: In this case, if $b \geq 0$, then the zero vector is the only one that could possibly satisfy (2), contradicting the assumption that K contains at least one non-zero vector. If $b < 0$, then consider $u, v \in \mathbb{R}^n$ with all zeroes except $u_1 = -b/\alpha_1$ and $v_2 = -b/\alpha_2$. Then $w = u \vee v$ does not satisfy (2), since the left-hand side is $-b - b + b = -b > 0$ while the right-hand side is 0.

2. $P \neq \emptyset$: Define $u, v \in \mathbb{R}^n$ such that the following conditions hold: $u_i = v_i = 0$ for all $i \in N \setminus \{1, 2\}$, $u_i = v_i$ for all $i \in P$, $(u_1, u_2) = (\varepsilon/\alpha_1, 0)$, $(v_1, v_2) = (0, \varepsilon/\alpha_2)$, and $\sum_{i \in P} a_i u_i = \sum_{i \in P} a_i v_i = b + \varepsilon$ for some arbitrary $\varepsilon > |b|$ (so $b + \varepsilon > 0$). Then $w = u \vee v$ does not satisfy (2), since the left-hand side is $\varepsilon + \varepsilon + b = b + 2\varepsilon$ while the right-hand side is $b + \varepsilon$ and $\varepsilon > 0$. □

C.4 Lower Bound for Set Covering

Theorem C.3. *Any algorithm for online set cover with ε-accurate suggestions is $\Omega(1/\varepsilon)$-competitive, even for the fractional case (or allowing randomization).*

As we did for the proof of Theorem B.6, we will consider the following prediction model: at each time t (independently of the past), the algorithm receives a good prediction (i.e., a set included in the optimal solution) with probability ε. With probability $1 - \varepsilon$, it does not receive any prediction at all. Again, the algorithm can generate predictions on its own in the $1 - \varepsilon$ case.

Proof. Within $1/\varepsilon$ steps, there is a constant probability that the algorithm receives no suggestion. So we define a sequence of $1/\varepsilon$ requested elements as follows: the first element e_1 is contained in $2^{1/\varepsilon}$ sets. Each subsequent element e_t is contained in exactly half of the sets that contain e_{t-1}, chosen uniformly at random. In expectation, the algorithm spends a total of $1/2$ on sets that contain e_t and do not contain $e_{t'}$ for any $t' > t$. Since these spent amounts are disjoint, over $1/\varepsilon$ steps, the algorithm spends $\log(2^{1/\varepsilon})/2 = \varepsilon/2$ in expectation, while an optimal solution spends 1 by picking the single set containing $1/\varepsilon$ elements.

We can scale this construction by making disjoint copies of this instance, and in each copy, there is a constant probability that the algorithm spends $\Omega(1/\varepsilon)$ while the optimal solution spends 1. □