Dynamic Set Cover: Improved Algorithms and Lower Bounds

Amir Abboud
IBM Almaden Research Center
amir.abboud@ibm.com

Raghavendra Addanki
University of Massachusetts Amherst
raddanki@cs.umass.edu

Fabrizio Grandoni
IDSIA, USI-SUPSI
fabrizio@idsia.ch

Debmalya Panigrahi
Duke University
debmalya@cs.duke.edu

Barna Saha
University of Massachusetts Amherst
barna@cs.umass.edu

Abstract

We give new upper and lower bounds for the *dynamic* set cover problem. First, we give a $(1+\varepsilon)f$-approximation for fully dynamic set cover in $O(f^2 \log n/\varepsilon^5)$ (amortized) update time, for any $\epsilon > 0$, where f is the maximum number of sets that an element belongs to. In the decremental setting, the update time can be improved to $O(f^2/\varepsilon^5)$, while still obtaining an $(1+\varepsilon)f$-approximation. These are the first algorithms that obtain an approximation factor linear in f for dynamic set cover, thereby almost matching the best bounds known in the offline setting and improving upon the previous best approximation of $O(f^2)$ in the dynamic setting.

To complement our upper bounds, we also show that a linear dependence of the update time on f is necessary unless we can tolerate much worse approximation factors. Using the recent distributed PCP-framework, we show that any dynamic set cover algorithm that has an amortized update time of $O(f^{1-\varepsilon})$ must have an approximation factor that is $\Omega(n^{\delta})$ for some constant $\delta > 0$ under the Strong Exponential Time Hypothesis.

CCS Concepts

• **Theory of computation** → **Online algorithms**.

Keywords

online algorithm, dynamic algorithm, randomized algorithm, set cover, competitive ratio.

ACM Reference Format:
Amir Abboud, Raghavendra Addanki, Fabrizio Grandoni, Debmalya Panigrahi, and Barna Saha. 2019. Dynamic Set Cover: Improved Algorithms and Lower Bounds. In *Proceedings of the 51st Annual ACM SIGACT Symposium on the Theory of Computing (STOC '19), June 23–26, 2019, Phoenix, AZ, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3313276.3316376

1 Introduction

Suppose, we need to solve a combinatorial optimization problem where the input to the problem changes over time. In such a dynamic setting, recomputing the solution from scratch after every update can be prohibitively time consuming, and it is natural to seek dynamic algorithms that provide faster updates. In the last few decades, efficient dynamic algorithms have been discovered for many combinatorial optimization problems, particularly in graphs such as shortest paths [4, 17, 19, 28], connectivity [5, 25, 26, 37], maximal independent set and coloring [7, 11, 34]. For many of these problems, maintaining exact solutions is prohibitively expensive under various complexity conjectures [2, 3, 23, 30], and thus the best approximation bounds are sought. In their seminal work [33], Onak and Rubinfeld proposed an algorithm for *matching* and *vertex cover* that maintains $O(1)$-approximate solutions to the maximum matching and minimum vertex cover in the graph. The algorithm runs in $t \cdot \text{polylog}(n)$ time for any sequence of t edge insertions and deletions in an n-vertex graph, i.e., in $O(\text{polylog}(n))$ time when *amortized* over all the updates. This has led to a flurry of activity in dynamic algorithms for matching and vertex cover [8–10, 13, 14, 22, 32, 36], and more recently, for the more general set cover problem [10, 12, 21] that we study in this paper.

In the set cover problem, we are given a universe X of n elements and a family $\mathcal{S}$ of m sets on these elements. The goal is to find a minimum-cardinality subfamily of sets $\mathcal{F} \subseteq \mathcal{S}$ such that $\mathcal{F}$ covers all the elements of X. The two traditional lines of inquiry for this problem are via greedy and primal dual algorithms, and have respectively led to a $\ln n$- and an f-approximation. Here, f is the maximum number of sets that an element belongs to in the set system $\mathcal{S}$. Both these results are known to be tight under appropriate complexity-theoretic assumptions [18, 29]. In the dynamic setting, the set system $\mathcal{S}$ is fixed, but the set of elements that needs to be covered in X changes over time. In particular, after the *insertion* of a new element, or the *deletion* of an existing one, the solution has to be updated to maintain feasibility and the approximation guarantee. The time taken to perform these updates is called the *update time* of the algorithm, and is often stated amortized over any fixed prefix of updates.

As in the case of the offline problem, dynamic algorithms for set cover have also followed two lines of inquiry. The first is to use greedy-like techniques, which were recently shown to yield an $O(\log n)$-approximation in $O(f \log n)$ update time by Gupta, Kumar, Krishnaswamy, and Panigrahi [21].[1] The second is to use a primal-dual framework, which was employed by Bhattacharya, Henzinger, and Italiano [12] to give an $O(f^2)$-approximation in $O(f \log (m+n))$ update time. Gupta *et al.* [21] and Bhattacharya, Chakrabarty, and Henzinger [10] also obtained a different but incomparable result using the primal-dual technique, which improves the update time

STOC '19, June 23–26, 2019, Phoenix, AZ, USA

ACM ISBN 978-1-4503-6705-9/19/06…$15.00
https://doi.org/10.1145/3313276.3316376

[1] All update times stated in this paper are amortized, unless stated otherwise.

to $O(f^2)$ thereby removing the dependence on n and m, but at the cost of a weaker approximation bound of $O(f^3)$. What stands out in these results is that:

- While dynamic and offline approximation factors match at $O(\log n)$, there is no $O(f)$-approximation known for the dynamic setting. Indeed, the only previous algorithm we are aware of that achieves this bound is one that recomputes the offline f-approximation after every update.
- The update times of these algorithms depend on $\log n$ and f. While the dependence on $\log n$ is not required, at least if we settle for an $O(f^3)$ approximation [10, 21], it is not clear if the polynomial dependence on f is fundamental. For instance, might it be possible to design a dynamic set cover algorithm whose update time only has a logarithmic dependence on f?

1.1 Our Results

Our first result closes the gap between offline and dynamic approximation for the set cover problem: **for any $\varepsilon > 0$, we give a $(1+\varepsilon)f$-approximation algorithm for dynamic set cover with an update time of $O(f^2 \log n/\varepsilon^5)$**. Previous algorithms for dynamic set cover heavily rely on deterministically maintaining a greedy-like or primal-dual structure on the set cover solution. Instead, our algorithm is based on the observation that a simple offline algorithm for the set cover problem achieves a $(1+\varepsilon)f$-approximation in $O(f/\varepsilon)$ expected update time when the elements are deleted in a random order. We switch this statement around by transferring the randomness to the algorithm in order to handle an arbitrary sequence of deletions (and insertions). As a result, our algorithm is randomized, and our update time bound holds in expectation. (The approximation bound holds deterministically.)

In the *decremental* setting where elements can only be deleted but not inserted, a simplification of the above algorithm yields the same **approximation factor of $(1+\varepsilon)f$ in amortized update time $O(f^2/\varepsilon^5)$**. This can be compared with the result of Gupta *et al.* [21] which achieves a (larger) $O(f^3)$-approximation with (roughly) the same update time, but in the fully dynamic case. As far as we know, the approximation bounds of [10, 21] do not change when considering the decremental setting, which has been extensively studied in the past for other problems [15, 24, 28].[2]

Finally, we turn to the problem of determining the dependence of the update time on f. Using the recently introduced framework of distributed PCP [1] from fine-grained complexity theory, **we show that under the *Strong Exponential Time Hypothesis (SETH)*, any dynamic set cover algorithm that has an (amortized) update time of $O(f^{1-\varepsilon})$ for any fixed $\varepsilon > 0$ must have an approximation factor of $(n/\log f)^{\Omega(1)}$**. Since a polynomial dependence on n in the approximation factor is rather weak, this result essentially states that any dynamic set cover algorithm must have a linear dependence on f in the update time[3]. This shows the update time bound of [21] to achieve $O(\log n)$ approximation is essentially tight within a $\log n$ factor. Our lower bound holds even if the algorithm is allowed a preprocessing stage with arbitrary polynomial runtime, and it also applies to the set-updates model where the elements are fixed but sets get inserted and deleted. This model is much more popular in the streaming setting [6, 31], especially when there are only insertions (see the work by Indyk et al. [27] and the many references therein). This is a novel application of the growing area of fine-grained complexity theory to show hardness of approximation of an NP-Hard problem.

1.2 Our Techniques

A natural starting point for our work is to use the deterministic greedy or primal dual techniques for dynamic set cover from [10, 12, 21]. An alternative strategy is to generalize previous randomized approaches for dynamic vertex cover [8, 36]. At a very high level, all these algorithms derive their results from maintaining, either explicitly or implicitly, a very structured dual solution that lower bounds the cost of the algorithm. Indeed, the algorithm of [21] for dynamic set cover can be thought of as a derandomization of the dynamic vertex cover algorithm of [36]. In order to improve the approximation factor to $O(f)$, these dual solutions must only violate the dual packing constraints by a constant factor (as against an $\Omega(f)$ violation in the previous results), but this requirement is too strict for the analysis framework of these papers that effectively rely on an f-discretization of the dual space.

Hence, we need a significantly new approach to improve the approximation factor to $O(f)$. We start with the following folklore algorithm for *offline* set cover. Initially, all elements are uncovered and the algorithm has an empty solution. Pick an arbitrary uncovered element and call it a pivot p. Then, include all sets containing p in the solution and mark all elements in those sets as covered. Repeat this process until all elements get covered. This algorithm runs in $O(nf)$ time and achieves an f-approximation, since no two pivots share a common set and the algorithm picks at most f sets for each pivot. We call this the *deterministic covering* algorithm.

Now, consider a decremental setting where elements are deleted over time, but in a *uniform random* order. A small modification to the deterministic covering algorithm gives a $(1+O(\varepsilon))f$-approximation in $O(f/\varepsilon)$ update time in this setting. Initially, we run deterministic covering to produce a feasible cover. During the deletion phase, the approximation bound may no longer hold because pivots are being deleted. To restore the bound, we re-run the deterministic covering algorithm whenever an ε-fraction of the pivots have been deleted. Since the number of undeleted pivots forms a lower bound on the optimal solution, it follows that this algorithm maintains an $f/(1-\varepsilon) = (1+O(\varepsilon))f$-approximation.

Let us now consider the update time. Clearly, deterministic covering takes $O(nf)$ time in every run. So, the question is: how frequently do we run it? Because of the random deletion order, we expect to delete an ε-fraction of all elements before an ε-fraction of pivots gets deleted. This suggests an informal amortized update bound of $O(nf/(\varepsilon n)) = O(f/\varepsilon)$. It turns out that this informal idea can be made formal, but we skip the details here since we are going to use this only for intuitive purposes.

More interesting for us is to transition from a random deletion order to an adversarial deletion order. The same update rule gives a

[2] For the incremental setting, where elements can be inserted but not deleted, the offline set cover algorithm itself gives an f-approximation in $O(f)$ update time.

[3] In our model, the elements and sets are fixed, as well as their membership relations, and the updates can change which of the elements are "active" in the instance. Thus, an element insertion can be specified with $O(\log n)$ bits, rather than the $\Omega(f)$ bits that may be required to list all its membership relations. This makes an $\Omega(f)$ lower bound on the update time non-trivial.

$(1+\varepsilon)f$-approximation, but now, the bound on update time may no longer hold. For instance, if all the pivots are deleted before other elements, the amortized update time is clearly much higher when the first ε-fraction of pivots gets deleted.

Our main idea, at this juncture, is to transfer the randomization from the deletion sequence to the algorithm itself. More specifically, instead of picking a pivot arbitrarily from the uncovered elements in each step, let us select it uniformly at random. We call this the *random covering* algorithm. Our hope is that an (oblivious) adversary deleting a single element will be able to pick a specific pivot with probability no higher than $1/n$. This would ensure that in expectation, an ε-fraction of the elements will have to be deleted before an ε-fraction of pivots is, as in the random deletion scenario.

However, this intuition is not quite correct. While the first pivot is indeed uniformly distributed over all elements, the subsequent pivots are not. To see this, consider the following example: suppose the sets represent edges of a graph containing $(n-2)/f$ cliques on f vertices each, and an isolated edge. For a vertex on the isolated edge to be chosen as the second pivot, it must not be covered by sets containing the first pivot and should be selected as the second pivot; the probability for this event is given by: $(1-2/n)\cdot(1/(n-f))$. Clearly, this probability exceeds $1/n$ for $f > 2$. As a consequence, the expected number of element deletions after which we need to run random covering might be smaller than εn. To overcome this bottleneck, we employ a more fine-grained update procedure: instead of running random covering over the entire undeleted instance, we run it only for a subset of elements. We maintain sufficient structure in the solution to still claim a $(1+\varepsilon)f$-approximation, while improving the update time to $O(f^2/\varepsilon^5)$ for the decremental setting, and $O(f^2 \log n/\varepsilon^5)$ for the fully dynamic setting where elements can be inserted in addition to deletions.

2 The Decremental Set Cover Algorithm

In this section, we give a dynamic set cover algorithm for the decremental setting. We denote the initial set system by $(X, \mathcal{S})$, where $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ is a collection of subsets of the ground set X that contains n elements. The maximum number of subsets that an element belongs to is denoted f:

$$f = \max_{x \in X} |\{i : x \in S_i\}|.$$

The elements are deleted in a fixed sequence, independent of the randomness of the algorithm, that is represented by $X = \{x_1, x_2, \ldots, x_n\}$.

2.1 The Algorithm

The description of the algorithm comprises two phases: the *initial phase* where the algorithm selects a feasible solution at the outset, and the *update phases* where the algorithm changes its solution in response to the deletion of elements. The feasible solution that the algorithm maintains dynamically is denoted by $\mathcal{F}$. Recall that the goal is to ensure that the cost of $\mathcal{F}$ is at most $(1+\varepsilon)f$ times that of an optimal solution for the set of undeleted elements at all times.

Both the initial and the update phases use a common subroutine that we call the *random cover* subroutine. We describe this subroutine first.

The Random Cover Subroutine. The random cover subroutine takes as input a set system $(X', \mathcal{S}')$ and outputs a feasible set cover solution $\mathcal{F}'$ for this set system. The algorithm is iterative, where each iteration starts with a set of *uncovered* elements $Y \subseteq X'$, adds a collection of sets $\mathcal{F}^+ \subseteq \mathcal{S}'$ to the solution $\mathcal{F}'$, and removes all the elements covered by the sets in $\mathcal{F}^+$ from the set of the uncovered elements Y for the next iteration. Initially, all elements in X' are uncovered, i.e., $Y = X'$, and the solution $\mathcal{F}'$ is empty, i.e., $\mathcal{F}' = \emptyset$. It only remains to describe an iteration, or more precisely, the sets $\mathcal{F}^+$ added to the solution $\mathcal{F}'$ in an iteration. The selection of $\mathcal{F}^+$ has three steps. First, the algorithm picks the set in $\mathcal{S}'$ that covers the maximum number of uncovered elements, breaking ties arbitrarily. Let us call this set Z, i.e., $Z = \arg\max_{S \in \mathcal{S}'} |S \cap Y|$. Next, the algorithm chooses an element in $Z \cap Y$, i.e., an uncovered element in the chosen set, *uniformly at random*, and calls this element the *pivot* for the current iteration. Let us call this pivot $p \in_{\text{u.a.r.}} Z \cap Y$. Finally, all sets in $\mathcal{S}'$ that contain the pivot are added to the solution, i.e., $\mathcal{F}^+ = \{S \in \mathcal{S}' : p \in S\}$. The random cover subroutine ends when all elements in X' are covered by the solution $\mathcal{F}'$, i.e., $Y = \emptyset$. This algorithm can be implemented in $O(f|X'|)$ deterministic time (details in Section 4).

The above completes the description of the random cover subroutine. However, it will be convenient to introduce some additional notation for this process that we will use later. Each iteration is characterized by its pivot p. We map the pivot to the set $S(p) := Z \cap Y$ from which it is chosen. If $|S(p)| \in [2^i, 2^{i+1})$, we say that p is a level-i pivot, and denote $\ell(p) = i$. Note that by the definition of the random cover subroutine, the pivots chosen in successive iterations have monotonically non-increasing levels, i.e., if pivot p is chosen in an earlier iteration and pivot p' in a later iteration, then $\ell(p) \geq \ell(p')$. Finally, if the sets $\mathcal{F}^+$ are added to the solution $\mathcal{F}'$ in an iteration with pivot p, then we denote $\mathcal{F}(p) = \mathcal{F}^+$. The set of previously uncovered elements that are covered by $\mathcal{F}(p)$ is denoted $\mathcal{X}(p)$.

Initial Phase. In the initial phase, the random cover subroutine is run on the the entire input set system $(X, \mathcal{S})$. This produces the initial solution $\mathcal{F}$.

In the algorithm, we also maintain sets P, D, and U that respectively represent *all*, *deleted*, and *undeleted* pivots. At the end of the initial phase, all the pivots in $\mathcal{F}$ are added to P and U, and D is empty. When an element e is deleted, if e is in P, then we move e from U to D, i.e., change its status from undeleted to deleted. Importantly, we keep this element in P. Changes to P are done only at the end of an update phase that we describe below.

Update Phase. An update phase is triggered when the number of deleted pivots exceeds an ε-fraction of the total number of pivots, i.e., $|D| \geq \varepsilon \cdot |P|$. In an update phase, the algorithm first fixes a level ℓ using a process that we describe later called the *level fixing process*. Having fixed this level ℓ, the algorithm discards all sets $\mathcal{F}(p)$ from $\mathcal{F}$ that were added by pivots p at levels ℓ or lower, i.e., where $\ell(p) \leq \ell$. Correspondingly, these pivots p are also removed from P and from either D or U depending on whether they are deleted or undeleted. As a result of this change to $\mathcal{F}$, some elements become uncovered in $\mathcal{F}$; this set is denoted by X'. The algorithm now runs the Random Cover subroutine on the instance $(X', \mathcal{S}')$ induced by X', where $\mathcal{S}' = \{S \cap X' : S \in \mathcal{S}, S \cap X' \neq \emptyset\}$. The resulting sets $\mathcal{F}'$ are added to the overall solution $\mathcal{F}$. Correspondingly, the newly selected pivots are also added to P and U. We say that levels ℓ and below have been updated in the current update phase. (Note that the newly selected

pivots will be at level ℓ or below.) Clearly, this restores feasibility of the solution $\mathcal{F}$. We already argued that the call to the Random Cover subroutine can be performed in $O(f|X'|)$ deterministic time. The same upper bound holds for the remaining operations related to the construction of the instance $(X', \mathcal{S}')$ and to the update of the approximate solution (see Section 4 for the details).

The level fixing process. We now describe the level fixing process. Let $\{0, 1, \ldots, L = \lfloor \log_2 n \rfloor\}$ be the set of levels. Let P_j, D_j, and U_j respectively denote the current total set of pivots, deleted pivots, and undeleted pivots at a given level j. This process finds a level ℓ with the following property: for every level $i \leq \ell$, $\sum_{j=i}^{\ell} |D_j| \geq \varepsilon \cdot \sum_{j=i}^{\ell} |P_j|$. In other words, the fraction of deleted pivots in levels $i, i+1, \ldots, \ell$ is at least an ε-fraction of the total number of pivots in these levels. We say that level ℓ is *critical*. The next lemma claims that at least one critical level exists whenever the number of deleted pivots is an ε-fraction of the total number of pivots.

LEMMA 1. *If $\sum_{j=0}^{L} |D_j| \geq \varepsilon \cdot \sum_{j=0}^{L} |P_j|$, then there exists at least one critical level.*

PROOF. Suppose $\sum_{j=0}^{L} |D_j| \geq \varepsilon \cdot \sum_{j=0}^{L} |P_j|$. Assume by contradiction that the claim is not true. Hence for each level ℓ, there exists a level $\text{FAIL}(\ell) \leq \ell$ (in case of ties, take the lowest such level) such that the condition does not hold, namely $\sum_{j=\text{FAIL}(\ell)}^{\ell} |D_j| < \varepsilon \cdot \sum_{j=\text{FAIL}(\ell)}^{\ell} |P_j|$. We next define a sequence of levels $\ell_1, \ldots, \ell_q$ as follows. Set $\ell_1 = L$. Given ℓ_i, halt if $\text{FAIL}(\ell_i) = 0$, else set $\ell_{i+1} = \text{FAIL}(\ell_i) - 1$ and continue with ℓ_{i+1}. Observe that the intervals $[\text{FAIL}(\ell_i), \ell_i]$ are disjoint and span $[0, L]$. We have

$$\sum_{j=0}^{L} |D_j| = \sum_{i=1}^{q} \sum_{j=\text{FAIL}(\ell_i)}^{\ell_i} |D_j| < \sum_{i=1}^{q} \varepsilon \cdot \sum_{j=\text{FAIL}(\ell_i)}^{\ell_i} |P_j| = \varepsilon \cdot \sum_{j=0}^{L} |P_j|.$$

This contradicts the assumption. □

The next lemma, shown in Section 4, establishes the time complexity of the above algorithm.

LEMMA 2. *Suppose we perform an update at critical level ℓ. Let p_i be the total number of pivots at levels $i \leq \ell$ right before this update. Then, the total time taken for this update phase is $O(\sum_{i \leq \ell} f^2 p_i 2^i)$.*

2.2 Analysis of the Competitive Ratio

LEMMA 3. *The competitive ratio of the algorithm is at most $f/(1-\varepsilon)$.*

PROOF. Consider the data structure right before the t-th deletion. Let P^t be the total set of pivots and U^t the set of undeleted pivots at that time. We also let OPT^t and $\mathcal{F}^t$ be the optimal and approximate solution at that time.

Observe that $|\mathcal{F}^t| \leq f \cdot |P^t|$ by construction. We claim that $|OPT^t| \geq |U^t|$. This implies the claim since by construction $|P^t| \leq |U^t|/(1-\varepsilon)$ at any time.

To see that, let us show by a simple induction that, for any two distinct $p, p' \in U^t$, there is no set $S \in \mathcal{F}$ covering both p and p'. Thus OPT^t needs to include a distinct set for each element of U^t. The Random Cover subroutine applied to X' never selects a pivot p' that is covered by sets selected due to a previous pivot p. This implies that the property holds after the initialization step, where $X' = X$.

Assume the property holds up to step $t \geq 1$, and suppose that at step $t+1$ an update happens at critical level ℓ, involving elements X'. By inductive hypothesis and the properties of Random Cover, the claim holds for any pair of pivots p, p' that are both contained in X' or in its complement $X \setminus X'$. Furthermore by construction X' is disjoint from any set S that covers a pivot $p \in X \setminus X'$, hence S cannot cover any pivot $p' \in X'$. □

2.3 Analysis of the Amortized Update Time

Our goal is to show that, after t deletions, the expected time taken by the algorithm is $O\left(fn + \frac{f^2}{\varepsilon^5} \cdot t\right)$. In particular, over a sequence of n deletions, the expected amortized cost per deletion is $O\left(\frac{f^2}{\varepsilon^5}\right)$.

Suppose we perform an update at critical level ℓ. Let P_i be the total set of pivots at level $i \leq \ell$ right before this update, of which D_i denotes the set of deleted pivots. Let also $p_i = |P_i|$ and $d_i = |D_i|$. Recall that by Lemma 2, the total time taken for this update phase is $O(\sum_{i \leq \ell} f^2 p_i 2^i)$. We call a level $i \leq \ell$ *charged* if $d_i \geq \frac{\varepsilon}{2} p_i$ and *uncharged* otherwise. We denote by $\mathcal{L} = (\ell_q, \ell_{q-1}, \ldots, \ell_1)$ the decreasingly ordered sequence of charged levels $j \leq \ell$. Observe that, by the definition of critical level, $\ell_q = \ell$. Also let D be the set of deleted pivots in the charged levels. The following lemma creates a useful mapping between D and $P := \cup_{i \leq \ell} P_i$.

LEMMA 4. *There exists a b-matching M between D and P such that:*

- *Each element of P is matched to exactly one element of D and each element of D to at most $b = 2/\varepsilon$ elements of P;*
- *If $d \in D$ is matched to $p \in P$, then $\ell(d) \geq \ell(p)$.*

PROOF. Let us define $\ell_0 = 0$. For every $k = 1, \ldots, q$, by definition

$$\sum_{j=\ell_{k-1}+1}^{\ell_{k-1}} d_j \leq \frac{\varepsilon}{2} \sum_{j=\ell_{k-1}+1}^{\ell_{k-1}} p_j. \tag{1}$$

Let us define $\mathcal{P}_{\ell_k} := \cup_{j=\ell_{k-1}+1}^{\ell_k} P_j$. Therefore, for every $h = 1, \ldots, q$, we have

$$\begin{aligned}
\sum_{k=h}^{q} d_{\ell_k} &= \sum_{j=\ell_{h-1}+1}^{\ell} d_j - \sum_{k=h}^{q} \sum_{j=\ell_{k-1}+1}^{\ell_{k-1}} d_j \\
&\overset{\ell \text{ critical}}{\geq} \sum_{j=\ell_{h-1}+1}^{\ell} \varepsilon p_j - \sum_{k=h}^{q} \sum_{j=\ell_{k-1}+1}^{\ell_{k-1}} d_j \\
&\overset{(1)}{\geq} \sum_{j=\ell_{h-1}+1}^{\ell} \varepsilon p_j - \frac{\varepsilon}{2} \sum_{k=h}^{q} \sum_{j=\ell_{k-1}+1}^{\ell_{k-1}} p_j \\
&\geq \frac{\varepsilon}{2} \sum_{j=\ell_{h-1}+1}^{\ell} p_j = \frac{\varepsilon}{2} \sum_{k=h}^{q} |\mathcal{P}_{\ell_k}|.
\end{aligned} \tag{2}$$

Let us replace each pivot $d \in D_j$, $j \in \mathcal{L}$, with $2/\varepsilon$ copies, and let us call the new set D'_j. Each copy of a pivot inherits the level of the original element. Let us sort $D' := \cup_{j \in \mathcal{L}} D'_j$ in non-increasing order of level (breaking ties arbitrarily), and similarly sort P. Now, for every element $p \in P$ according to this order, we match p with

the first unmatched element $d' \in D'$. The b-matching is obtained by collapsing the copies in D' of the same pivot in D. Observe that this allows us to match all elements of P since

$$|D'| = \frac{2}{\varepsilon}|D| = \frac{2}{\varepsilon}\sum_{k=1}^{q} d_{\ell_k} \overset{(2) \text{ with } h=1}{\geq} \frac{2}{\varepsilon}\cdot\frac{\varepsilon}{2}\sum_{k=1}^{q}|\mathcal{P}_{\ell_k}| = |P|.$$

It remains to show that the condition on the levels is satisfied. Suppose there exists some $d' \in D'$ matched to $p \in P$ with $\ell_{h-1} = \ell(d') < \ell(p)$. By construction this implies that

$$\frac{2}{\varepsilon}\sum_{k=h}^{q} d_{\ell_k} = \sum_{k=h}^{q}|D'_{\ell_k}| < \sum_{k=h}^{q}|\mathcal{P}_{\ell_k}|,$$

which contradicts (2). □

Consider a sequence of t deletions, and let T be time right before the $(t+1)$-st deletion occurs (or the end of the execution if no such deletion exists). We wish to bound the expected total update time till time T as a multiple of t. To that aim, we need a more global notation. Suppose that level i is updated q_i times in total, and let P_i^j be the total set of pivots in the j-th such update. We use $P(i)$ to denote the multiset of pivots given by the union of the sets P_i^j, and define $p(i) := |P(i)|$. The pivots of type P_i^j where level i is charged on the j-th update are called charged and denoted $CH(i)$, $ch(i) := |CH(i)|$. The charged deleted pivots at level i are denoted by $D(i)$, $d(i) := |D(i)|$.

We call level i *globally-charged* iff $ch(i) \geq \frac{\varepsilon}{4}p(i)$, and *globally-uncharged* otherwise. We also let GC denote the (random) set of globally-charged levels. We next show that, in order to bound the total update time, we can focus on globally-charged levels only.

Lemma 5. *The expected running time of the algorithm is* $O(fn)+O\left(\sum_{i=0}^{L}\Pr[i \in GC]\cdot \mathrm{E}\left[\frac{f^2}{\varepsilon}2^i d(i) \,\middle|\, i \in GC\right]\right)$.

Proof. Excluding the initialization cost of $O(fn)$ and by Lemma 2, we can focus on bounding $O(\sum_i f^2 p(i) 2^i)$. We use the following token argument to upper bound the latter cost. We provide $f^2 2^i$ tokens to each pivot $p \in P(i)$, where each token can pay for a large enough constant amount of work. Then we transfer these tokens to charged deleted pivots in globally-charged levels so that all tokens are transferred and each charged deleted pivot at level i is charged with at most $\frac{4f^2}{\varepsilon}2^i$ tokens.

We next describe the transfer process. Let M denote the b-matchings in Lemma 4. In particular, each pivot $p \in P(i)$ is matched with some charged deleted pivot $M(p)$ at no lower level, and each charged deleted pivot is matched with at most $2/\varepsilon$ pivots. We remark that uncharged pivots p have their $M(p)$ at a strictly higher level by construction. First of all, each pivot $p \in P(i)$ transfers its tokens to the corresponding charged deleted pivot according to M. Note that at this point each charged deleted pivot at level i owns at most $\frac{2f^2}{\varepsilon}2^i$ tokens.

Next we proceed in increasing order of level i. For a given level i, each charged deleted pivot $d \in D(i)$ owns the tokens originally owned by d and possibly tokens transferred from lower levels. If $i \in GC$ we do nothing. Otherwise (i.e., $i \notin GC$), we define a b-matching M^i where each charged deleted pivot $d \in D(i)$ is matched with $\frac{2}{\varepsilon}$ distinct uncharged pivots in $P(i)$ so that no uncharged pivot in $P(i)$ is matched twice. Note that this is possible since, by definition of globally-uncharged level, the uncharged pivots in $P(i)$ are at least

$$p(i) - ch(i) \geq \left(1-\frac{\varepsilon}{4}\right)p(i) \geq \left(1-\frac{\varepsilon}{4}\right)\frac{4}{\varepsilon}ch(i) \geq \frac{2}{\varepsilon}ch(i),$$

where we assumed $\varepsilon \leq 2$ w.l.o.g. Now d transfers an $\frac{\varepsilon}{2}$-fraction of its tokens to each corresponding pivot in $M^i(d)$. Finally each matched uncharged pivot $p \in P(i)$ transfers the received tokens to the corresponding deleted pivot $M(p)$ in the global b-matching M. Observe that $M(p)$ must be at strictly higher level than p, hence the process is well-defined.

Clearly at the end of the process all the tokens are transferred to charged deleted pivots in globally-charged levels and no token is left. We next prove by induction that, at the end of iteration i (where level i is considered), the number of tokens charged to each $d \in D(i)$ is at most $\frac{4}{\varepsilon}f^2 2^i$. The claim follows.

The base of the induction $i = 0$ is trivially true. Indeed, if $i \notin GC$ all the tokens of d are transferred to higher levels. Otherwise d can only be charged with the starting number of tokens, which is at most $\frac{2f^2}{\varepsilon}$ since there are no lower levels that can transfer tokens to level i. Note that the number of tokens that d is charged with does not change in the rest of the token transfer process.

Next consider a level $i > 0$, and assume the claim is true for levels $i-1$ and lower. For any $d \in D(i)$, again the claim holds trivially if $i \notin GC$. Otherwise, d initially has up to $\frac{2f^2}{\varepsilon}2^i$ tokens. Furthermore, d can receive extra tokens from up to $\frac{2}{\varepsilon}$ pivots p of strictly lower levels. Each such p at level $\ell \leq i-1$ transfers to d an $\frac{\varepsilon}{2}$-fraction of the tokens of some charged deleted pivot of level ℓ. By the inductive hypothesis, the total number of tokens received by d at the end of iteration i is at most

$$\frac{2}{\varepsilon}f^2 2^i + \frac{2}{\varepsilon}\cdot\frac{\varepsilon}{2}\cdot\frac{4}{\varepsilon}f^2 2^{i-1} = \frac{4}{\varepsilon}f^2 2^i.$$

Again, the number of tokens that d is charged with does not change in the rest of the token transfer process. □

Based on the above lemma, what remains to show is a bound for $\mathrm{E}\left[2^i d(i) \,\middle|\, i \in GC\right]$. We bound this in terms of $\mathrm{E}[t(i) \mid i \in GC]$, where $t(i)$ is the (random) number of deletions that happen at level i. Instead of considering $t(i)$ directly, we rather focus on the following quantity. For a pivot $p(S)$ sampled from some set S (considering only the uncovered elements at that time), let $i(S)$ be the relative position of $p(S)$ in S w.r.t. the deletion order. We remark that $i(S)$ is uniformly distributed in $\{1,\ldots,|S|\}$. Define $x(i) := \sum_{S:p(S)\in D(i)} i(S)$. Notice that deterministically $x(i) \leq t(i)$ since all the elements that appear in a set S no later than the respective pivot $p(S)$ in the deletion order are deleted assuming $p(S)$ is deleted.

Let us also condition on $p(i) = p$ for some fixed value p, and consider $\mathrm{E}[x(i) \mid i \in GC, p(i) = p]$. We now relate this quantity to another random process. Suppose is an adversary that defines a collection of exactly p sets $\mathcal{S}$ (possibly with repetition), where each set $S \in \mathcal{S}$ has size $|S| \in [2^i, 2^{i+1})$, and a deletion sequence over the elements of the sets. Now we sample a pivot $\tilde{p}(S)$ uniformly at random in each set $S \in \mathcal{S}$, and let $\tilde{i}(S)$ be the relative position of the pivot $\tilde{p}(S)$ in S w.r.t. the deletion sequence. The adversary is informed about the values $\tilde{i}(S)$. The adversary chooses a subcollection $\mathcal{S}' \subseteq \mathcal{S}$ of size at least $\frac{\varepsilon^2}{8}p$, and computes $\tilde{x}_p(i) = \sum_{S\in\mathcal{S}'}\tilde{i}(S)$. The adversary makes these choices in order to minimize $\mathrm{E}\left[\tilde{x}_p(i)\right]$.

LEMMA 6. $\mathrm{E}\left[\tilde{x}_p(i)\right] \leq \mathrm{E}\left[x(i) \mid i \in GC, p(i) = p\right]$.

PROOF. We use a coupling argument. Intuitively, the adversary can mimic the behavior of any execution of our decremental algorithm. In more detail, consider any execution of the decremental algorithm such that $i \in GC$ and $p(i) = p$. We couple the behavior of the adversary with this execution as follows. The adversary selects the same deletion order as in the input, and as collection $\mathcal{S}$ precisely the sets that appear at level i right before each update phase that involves that level (hence $|\mathcal{S}| = p$). By coupling, we can assume that the sampled pivots in $\mathcal{S}$ are precisely the pivots $P(i)$ of level i in the execution of the algorithm. The collection $\mathcal{S}'$ is given by the sets $S \in \mathcal{S}$ such that the corresponding pivots are deleted and charged in the considered execution of the algorithm. Observe that $d(i) \geq \frac{\varepsilon}{2} ch(i) \geq \frac{\varepsilon^2}{8} p(i) = \frac{\varepsilon^2}{8} p$, hence the constraint $|\mathcal{S}'| \geq \frac{\varepsilon^2}{8} p$ is satisfied. One has $\tilde{x}_p(i) = x(i)$ deterministically in the above construction, hence $\mathrm{E}\left[\tilde{x}_p(i)\right] = \mathrm{E}\left[x(i)\right]$. The claim follows since the adversary makes the optimal choices in order to minimize $\mathrm{E}\left[\tilde{x}_p(i)\right]$. □

LEMMA 7. $\mathrm{E}\left[\tilde{x}_p(i)\right] \geq \frac{\varepsilon^4}{1024} 2^i p$.

PROOF. Consider the collection $\mathcal{S}$ of p sets and the deletion order chosen by the adversary. Once the p pivots $\tilde{P}(i)$ are fixed, the best strategy for the adversary is to choose the sub-collection $\mathcal{S}'$ of precisely $\frac{\varepsilon^2}{8} p$ sets S with smallest $\tilde{i}(S)$ (breaking ties arbitrarily). It remains to bound the expected value of $\tilde{x}_p(i) = \sum_{S \in \mathcal{S}'} \tilde{i}(S)$.

We say that a set $S \in \mathcal{S}$ is *bad* if $\tilde{i}(S) \leq \frac{\varepsilon^2}{32} 2^i$ and *good* otherwise. We let $b(i)$ and $g(i)$ be the number of bad and good sets, respectively. Observe that each set is bad independently with probability at most $\frac{\varepsilon^2}{32}$, hence $\mathrm{E}\left[b(i)\right] \leq \frac{\varepsilon^2}{32} p$. By Markov's inequality,

$$\Pr\left[b(i) \geq \frac{\varepsilon^2}{16} p\right] \leq \frac{1}{2}.$$

Given the event $\mathcal{E} = \left\{b(i) < \frac{\varepsilon^2}{16} p\right\}$, one has that at least one half of the $\frac{\varepsilon^2}{8} p$ selected sets are good, in which case deterministically

$$\tilde{x}_p(i) \geq \frac{\varepsilon^2}{16} p \cdot \frac{\varepsilon^2}{32} 2^i = \frac{\varepsilon^4}{512} 2^i p.$$

We can conclude that

$$\mathrm{E}\left[\tilde{x}_p(i)\right] \geq \Pr\left[\mathcal{E}\right] \cdot \mathrm{E}\left[\tilde{x}_p(i) \mid \mathcal{E}\right] \geq \frac{1}{2} \cdot \frac{\varepsilon^4}{512} 2^i p. \quad \square$$

Finally, we put the above lemmas together to obtain the desired bound.

LEMMA 8. *The expected running time of the algorithm in the decremental case is $O\left(fn + \frac{f^2}{\varepsilon^5} \cdot t\right)$, where t is the number of deletions.*

PROOF. Let us consider a given level i. One has

$$\mathrm{E}\left[t(i) \mid i \in GC, p(i) = p\right] \geq \mathrm{E}\left[x(i) \mid i \in GC, p(i) = p\right] \overset{\text{Lem. 6}}{\geq} \mathrm{E}\left[\tilde{x}_p(i)\right] \overset{\text{Lem. 7}}{\geq} \frac{\varepsilon^4}{1024} 2^i p. \tag{3}$$

Hence

$$\mathrm{E}\left[t(i) \mid i \in GC\right] = \sum_p \Pr\left[p(i) = p \mid i \in GC\right] \cdot \mathrm{E}\left[t(i) \mid i \in GC, p(i) = p\right]$$

$$\overset{(3)}{\geq} \sum_p \Pr\left[p(i) = p \mid i \in GC\right] \cdot \frac{\varepsilon^4}{1024} 2^i p = \frac{\varepsilon^4}{1024} 2^i \cdot \mathrm{E}\left[p(i) \mid i \in GC\right]$$

$$\geq \frac{\varepsilon^4}{1024} 2^i \cdot \mathrm{E}\left[d(i) \mid i \in GC\right]. \tag{4}$$

Now, we note that

$$\sum_{i=0}^{L} \Pr\left[i \in GC\right] \cdot \frac{f^2}{\varepsilon} 2^i \mathrm{E}\left[d(i) \mid i \in GC\right]$$

$$\overset{(4)}{\leq} \sum_{i=0}^{L} \Pr\left[i \in GC\right] \cdot \frac{f^2}{\varepsilon} 2^i \frac{1024}{\varepsilon^4 2^i} \mathrm{E}\left[t(i) \mid i \in GC\right]$$

$$\leq \sum_{i=0}^{L} \frac{1024 f^2}{\varepsilon^5} \mathrm{E}\left[t(i)\right] = \frac{1024 f^2}{\varepsilon^5} t.$$

The lemma now follows from Lemma 5. □

We summarize the results in the following theorem.

THEOREM 1. *Given an $\varepsilon > 0$, let $\Delta = \frac{f^2}{\varepsilon^5}$. There exists a decremental algorithm for set cover that achieves an $f(1+\epsilon)$ approximation and takes $O(\Delta \cdot t)$ time in expectation over t updates.*

3 The Fully Dynamic Set Cover Algorithm

In this section, we extend the algorithm for the decremental case to the fully dynamic case. At any time t, let $A \subseteq X$ denote the elements that need to be covered; we call these the *active* elements. Our goal is to maintain a feasible set cover $\mathcal{F}$ for the active elements A and ensure that the cost of $\mathcal{F}$ is at most $(1+\epsilon)f$ times that of an optimal solution. At the beginning, $A = \emptyset$ and $\mathcal{F} = \emptyset$. Elements are then inserted or deleted from A in a fixed sequence, independent of the randomness of the algorithm. If an element is inserted and then gets deleted and reinserted, we treat the two insertions separately as two copies of the same element.

3.1 The Algorithm

We now describe the *update phases* where the algorithm changes its solution in response to the insertions and deletions of elements. The update phases are very similar to the decremental algorithm, but with a few critical changes. To describe the changes, we need to introduce some additional notation. Just like the decremental algorithm, the fully dynamic algorithm maintains a set of pivots P, and at any time, the solution $\mathcal{F}$ can be completely specified by P as follows: $\mathcal{F} = \{S \mid S \ni p\}$. $S(p)$ denotes the set of elements from which a pivot $p \in P$ is chosen and $\mathcal{X}(p)$ denotes the set of elements p is accounted to cover at any point of time. If $|S(p)| \in [2^i, 2^{i+1})$, we say that pivot p is a level-i pivot, and denote $\ell(p) = i$. We call the sets $\{S \mid S \ni p, \ell(p) = i\}$ level-i sets. In addition, we partition $X(p)$ into two subsets $Orig(p)$ and $Extra(p)$, that is $X(p) = Orig(p) \cup Extra(p)$ and $Orig(p) \cap Extra(p) = \emptyset$. An element $e \in Orig(p)$ is called an *original* element and an element $e \in Extra(p)$ is called an *extra* element. $Orig(p)$ consists of all elements that p is accounted to cover at the time when p was chosen to be a pivot and $\mathcal{F}^+ = \{S \in \mathcal{S}' : p \in S\}$ sets are included in the solution. Thus, $S(p) \subseteq Orig(p)$. It is possible that p is accounted to cover more elements due to later updates. Those elements are added to $Extra(p)$. Along with P, the algorithm also maintains sets D and U of deleted and undeleted

pivots respectively, just like in the decremental algorithm. When an element $e \in P$ is deleted, we move e from U to D but keep this element in P. Changes to P are done only during an update phase which we describe below.

Insertion of a new active element. Supppose a new element e is inserted in the set of active elements A. If $\{S \mid S \ni e\} \cap \mathcal{F} \neq \emptyset$, then e is already covered by the current solution $\mathcal{F}$. In this case, let S be the set containing e at the highest level breaking ties arbitrarily. If $p \in P \cap S$ denotes the pivot in S, then we insert e in $Extra(p)$ and $X(p)$, and $\mathcal{F}$ remains unchanged. Otherwise, e is not covered by the current solution $\mathcal{F}$. In this case, we include e as a level-0 pivot and set $S(e) = \{e\}$, $X(e) = Orig(e) = \{e\}$. We update $\mathcal{F} = \mathcal{F} \cup \{S \mid S \ni e\}$. We also update the sets P and U to include e.

Deletion of an existing active element. When an element e is deleted from the set of active elements A, we mark e as deleted from the sets $\{S \mid S \ni e\} \cap \mathcal{F}$. If $e \in P$, we move e from U to D. By doing so, if $|D| > \epsilon \cdot |P|$, then we say that an update phase has been triggered, and perform the following additional steps.

Update Phase. First, we fix a critical level ℓ using the *level fixing process* of the decremental algorithm. Having fixed this level ℓ, we discard all sets $\mathcal{F}(p)$ from $\mathcal{F}$ that were added by pivots p at levels ℓ or lower, i.e., where $\ell(p) \leq \ell$. Correspondingly, these pivots p are also removed from P and from either D or U depending on whether they are deleted or undeleted. As a result, a set of active elements become uncovered in $\mathcal{F}$; this set is denoted X'.

Next, the update phase has two steps, a *movement step* and a *covering step*, to cover the elements in X'.

Movement step: For each element $e \in X'$, we check if there is a set $S \in \mathcal{F}$ containing e at a level $\ell' > \ell$. If yes, we select a set $S \ni e$ at the highest level (breaking ties arbitrarily). If $p = P \cap S$, then e is added to $Extra(p)$ and $X(p)$.

Covering step: Let $Y' \subseteq X'$ denote the elements left uncovered after the movement step. We now run the Random Cover subroutine on the instance $(Y', \mathcal{S}')$ induced by Y' and add the resulting sets $\mathcal{F}'$ to the overall solution $\mathcal{F}$. We say that levels ℓ and below have been updated in the current update phase. The newly selected pivots are added to P and U. For every newly chosen pivot $p \in Y' \cap P$, if $S(p) \in [2^i, 2^{i+1})$, then pivot p is a level-i pivot and we include $\{S \mid S \ni p\}$ at level i. Note that it is possible that $i > \ell$ due to newly inserted elements. Also note that all the elements of Y' now become original elements after the covering step.

The next lemma, shown in Section 4, establishes the time taken to implement the above algorithm.

LEMMA 9. *The above algorithm for the insertion of a new element, or the deletion of an existing element that does not trigger an update phase, takes $O(f)$ time. The time complexity of an update phase is $O(f|X'|)$.*

3.2 Analysis of the Competitive Ratio

LEMMA 10. *The competitive ratio of the algorithm is at most $f/(1-\varepsilon)$.*

PROOF. Consider the data structure right before the t-th update. Let P^t be the set of pivots at that time, with U^t being the subset of undeleted pivots. We also let OPT^t and $\mathcal{F}^t$ be the optimal and approximate solution at that time.

Observe that $|\mathcal{F}^t| \leq f \cdot |P^t|$ by construction. We claim that $|OPT^t| \geq |U^t|$. This implies the claim since by construction $|P^t| \leq |U^t|/(1-\varepsilon)$ at any time.

To see that, let us show by a simple induction that, for any two distinct $p, p' \in U^t$, there is no set $S \in \mathcal{F}$ covering both p and p'. Thus OPT^t needs to include a distinct set for each element of U^t. Since we start with an empty set cover, the property holds at the outset. Assume the property holds up to step $t \geq 1$, and consider step $t+1$. If an element e is inserted at time $t+1$, then e becomes a pivot if and only if e is not covered by any existing set in $\mathcal{F}$. Hence, the property holds. If a non-pivot element e is deleted at $t+1$, or e is a pivot but its deletion does not trigger an update phase, then since we do not change the solution $\mathcal{F}$, the property holds by the inductive hypothesis.

Now, assume e is a pivot and its deletion triggers an update phase at critical level ℓ', with the elements covered at levels $i \leq \ell$ being denoted by X'. Note that we select a set of new pivots from $Y' \subseteq X'$ and let $\mathcal{F}'$ denote the new sets that are added after the update phase. We consider three cases. if $p, p' \in \mathcal{F}$, then they do not belong to the same set by the inductive hypothesis. If $p, p' \in \mathcal{F}'$, then they do not belong to the same set since the Random Cover subroutine picked both these elements as pivots. Finally, if $p \in \mathcal{F}$ and $p' \in \mathcal{F}'$, then the movement step ensures that p' is not covered by $\mathcal{F}$ whereas all sets containing p are in $\mathcal{F}$. Therefore, p and p' do not belong to the same set in this case either. Therefore, the property holds after the $(t+1)$-st update. □

3.3 Analysis of the Amortized Update Time

Consider a sequence of t updates, and let T be the time right before the $(t+1)$-st update occurs (or the end of the execution if no such update occurs). Our goal is to bound the expected update time till T as $O\left(t \cdot \frac{f^2 \log n}{\epsilon}\right)$. We recall some definitions from Section 2 and introduce some new notation for the purpose of the analysis.

Old notation. Recall the definition of the critical level ℓ and Lemma 4. Note that when performing an update at a critical level ℓ, a level $i \leq l$ is said to be charged if $d_i \geq \frac{\epsilon}{2} p_i$ and uncharged otherwise.

Suppose that level i is updated q_i times in total, and let P_i^j be the total set of pivots in the j-th such update. $P(i)$ is the multiset of pivots obtained by taking union over P_i^j and $p(i) = |P(i)|$. The pivots of type P_i^j where level i is charged on the j-th update are called charged and denoted $CH(i)$, $ch(i) = |CH(i)|$. The charged deleted pivots at level i are denoted $D(i), d(i) = |D(i)|$. Let I be the total number of insertions up to time T.

Recall that a level i is globally-charged iff $ch(i) \geq \frac{\epsilon}{4} p(i)$, and globally-uncharged otherwise. Let GC denote the random set of globally-charged levels.

New notation. Let us now define a new mapping R that maps an element e (on which an update phase operates) either to a charged deleted pivot or to an insertion. To construct this mapping, if an element e takes part in q_e update phases, including both the movement and the covering steps, then each of these occurrences is treated separately.

An element $e \in Orig(p)$ is mapped to a charged deleted pivot d, i.e., $R(e) = d$, if $M(p) = d$. Now consider an element $e \in Extra(p)$. If e has never been an original element, then it must have been inserted

as an extra element and has taken part only in movement steps since then. This is because whenever a covering step processes an element, it becomes an original element. In this case, we map e to its insertion, denoted e_I and set $R(e) = e_I$. Otherwise, consider the last update phase, when e was an original element just before the update and became an extra element immediately after it. If $e \in Orig(p')$ and $M(p') = d'$ during that phase, then set $R(e) = d'$.

Note that all q_e occurrences of e are mapped by R to either to the insertion e_I or to charged deleted pivots. If $R(e) = e_I$, we say insertion e_I is responsible for e and if $R(e) = d$, we say the charged deleted pivot d is responsible for e. Note that a charged deleted pivot d is responsible for an element e if an only if $e \in Orig(p)$ for some pivot p and $M(p) = d$.

Let us use $L = \lfloor \log_2 n \rfloor$ to denote the largest level and $\mathcal{X}$ to denote the multiset of elements obtained by taking the union of $Orig(p)$ and $Extra(p)$ over all $p \in P$.

LEMMA 11. *Each charged deleted pivot $d \in D(i)$ is responsible for at most $(2/\epsilon) \cdot f \cdot 2^{i+1} \cdot (L+1)$ elements of $\mathcal{X}$.*

PROOF. Consider a deleted pivot $d \in D(i)$ and let $P_d = \{p \mid M(p) = d\}$ be the set of pivots mapped to d. Consider any pivot $p \in P_d$. Note that, d is only responsible for the elements in $Orig(p)$ and that $\ell(p) \leq \ell(d) = i$. By definition of $\ell(p)$, each set containing p covers less than $2^{\ell(p)+1}$ new elements at the time p was selected to be a pivot. Hence $|Orig(p)| < f \cdot 2^{\ell(p)+1} \leq f \cdot 2^{i+1}$. Now let us count the number of times d was held responsible for $e \in Orig(p)$. The element e was an original element just before the update phase that operates on d. If e gets processed during that phase by the covering step, then d was responsible for e only once. Otherwise, e gets processed by the movement step during that phase and becomes an extra element. If e is processed r times by the movement step before becoming an original element again, then d is responsible $r+1$ times for e. However, the level of e strictly increases after each movement step. Therefore, $r \leq L$. Thus, d can be responsible for e at most $L+1$ times. This holds for all $p \in P_d$. Now, the claim follows noting $|P_d| \leq 2/\varepsilon$ by Lemma 4. □

LEMMA 12. *Each insertion $e_I \in I$ is responsible for at most $L+1$ elements of $\mathcal{X}$.*

PROOF. An insertion e_I is only responsible for the element e. If e takes part in r movement steps before becoming an original element for the first time, then e_I is responsible $r+1$ times for e. Since the level of e strictly increases after each movement step, we have $r \leq L$. Thus, the claim follows. □

We now have an analog of Lemma 5.

LEMMA 13. *The expected running time of the algorithm after t updates is*

$$O\left(f \cdot |I| \cdot (L+1) + \sum_{i=0}^{L} \Pr[i \in GC] \cdot \mathrm{E}\left[\frac{f^2}{\varepsilon} \cdot 2^i \cdot d(i) \cdot (L+1) \,\middle|\, i \in GC\right]\right).$$

PROOF. From Lemma 9, the total update time up to time T is $O(f \cdot (|I| + |\mathcal{X}|))$. Now from Lemma 11 and Lemma 12,

$$f \cdot (|I| + |\mathcal{X}|) = f \cdot \left((L+1) \cdot |I| + (2/\epsilon) \cdot f \cdot (L+1) \cdot \sum_i d(i) \cdot 2^{i+1}\right).$$

We can give each charged deleted pivot $\frac{4}{\epsilon} \cdot f \cdot (L+1) \cdot 2^i$ tokens and then follow the token transfer process of Lemma 5 so that all these tokens are transferred to charged deleted pivots in the globally charged levels. Moreover, each charged deleted pivot in a globally charged level contains at most $\frac{8}{\epsilon} \cdot f \cdot (L+1) \cdot 2^i$ tokens. The lemma now follows. □

Let $t(i)$ denote the (random) number of deletions at level i up to time T, and let $t' = \sum_i t(i)$. Then $t' \leq t$. Exactly as in the decremental case, we can now use Lemmas 6, 7, and 8 to bound

$$\sum_{i=0}^{L} \Pr[i \in GC] \cdot \mathrm{E}\left[\frac{f^2}{\varepsilon} 2^i d(i)(L+1) \,\middle|\, i \in GC\right] \leq \frac{1024}{\epsilon^5} f^2 t(L+1). \tag{5}$$

LEMMA 14. *The expected running time of the algorithm in the fully dynamic case is $O\left(\frac{f^2 \log n}{\epsilon^5} \cdot t\right)$, where t is the number of updates.*

PROOF. This follows from Lemmas 8 and 13, and Eq. (5), noting that $t' \leq t$, $|I| \leq t$, and $L = (logn)$. □

We summarize the results in the following theorem.

THEOREM 2. *Given an $\varepsilon > 0$, let $\Delta = \frac{f^2 \log n}{\epsilon^5}$. There exists a fully-dynamic algorithm for set cover that achieves an $f(1+\epsilon)$ approximation and takes $O(\Delta \cdot t)$ time in expectation over t updates.*

4 Implementation Details and Running Time

In this section, we give implementation details of the algorithms, leading to the proofs of Lemma 2 (decremental) and Lemma 9 (fully dynamic).

4.1 Decremental Algorithm

We assume that any given set cover instance $(X', \mathcal{S}')$, with maximum frequency f is represented as follows. Elements (resp., sets) are labelled 1 to $n' = |X'|$ (resp., $m' = |\mathcal{S}'|$). W.l.o.g. we can assume that each set covers at least one element of X', so that $m' \leq f \cdot n'$. We have a vector SET indexed by elements, where $SET[e]$ is the list of sets $\mathcal{S}'(e)$ containing e. Observe that $SET[e]$ contains at most f entries. We assume that sets are described by a vector $ELEM$ indexed by sets, where $ELEM[S]$ is a list of elements contained in set S. We keep a pointer from each $e \in ELEM[S]$ to the corresponding entry S in $SET[e]$ and vice-versa.

In order to implement deletions, we proceed as follows. We maintain a Boolean vector DEL indexed by $e \in X'$, which is initialized to false. When element e is deleted, we set $DEL[e] = true$. Furthermore, we scan $SET[e]$, and for each $S \in SET[e]$ we remove e from $ELEM[S]$. Note that this can be done in $O(1)$ time for each set S using the pointers mentioned above, i.e., in time $O(f)$ per element e. This also implies that deleting all the elements one by one takes $O(f|X'|)$ time in total.

The Random Cover Subroutine. The Random Cover procedure computes a sequence of pivots P'. Furthermore, for each pivot $p \in P'$, it computes a collection $\mathcal{F}(p)$ of sets that are added to the solution because of p, and the corresponding set $\mathcal{X}(p)$ of newly covered elements. This takes $O(f + |\mathcal{X}(p)|)$ time for a given pivot p.

It remains to specify how we efficiently extract a set S of maximum cardinality at each step to select a pivot. We maintain a list $SORT$ whose entries are pairs (i, L_i), where i is the cardinality of a set and L_i is the list of sets of cardinality i. We store all such entries with L_i not empty, in decreasing order of i. This list can be initialized in linear time $O(f|X'|)$ (say, using radix sort). We also maintain pointers from each set S to the corresponding entry in the list $L_{|S|}$.

The first element of the first list L_i is the selected set S at each step. Then we update $SORT$ as follows. Each time we remove an element e from some set S' of cardinality i, we remove S' from L_i and add it to L_{i-1}. Note that this might involve creating a new entry $(i-1, L_{i-1})$ in $SORT$ (if S becomes the only set of cardinality $i-1$), or deleting the entry (i, L_i) from $SORT$ (if S was the only set of cardinality i). In any case, these operations can be performed in $O(1)$ time. It follows that the entire procedure can be implemented in time $O(f \cdot |X'|)$ time.

The Set Cover Solution. We store and maintain the approximate solution as follows. We maintain the set cover instance under deletions as described before. Furthermore, we maintain vectors $\mathcal{F}$ and $\mathcal{X}$. For a pivot p, $\mathcal{F}(p)$ is the corresponding list of selected sets because of p, and $\mathcal{X}(p)$ is the associated list of newly covered elements due to these selected sets. These two lists are empty if p is not a pivot.

Level selection. We maintain counters D and P labelled by levels $i = 0, \ldots, \lceil \log_2 n \rceil$, where $D[i]$ (resp., $P[i]$) is the number of deleted pivots (resp., all pivots) at level i. When we delete a pivot at level i, we increment $D[i]$. When we update at critical level ℓ, we set $D[i] = P[i] = 0$ for all $i \leq \ell$. Furthermore we increment $P[i]$ for each newly computed pivot of level i. Clearly these operations have amortized cost $O(1)$ per update. We similarly maintain the total number $\tilde{D}$ and $\tilde{P}$ of deleted pivots and all pivots, respectively. By comparing $\tilde{D}$ and $\tilde{P}$ at each deletion, we can check whether the condition for the update of a suffix is satisfied. In that case, using D and P, it is easy to compute in $O(\ell^2)$ time the lowest critical level ℓ.

Update Phase. We next describe how, given a critical level ℓ, we update the approximate solution. We keep a list $GREEDY$ whose entries are pairs (j, P_j). Here j is a level and P_j is the list of pivots of that level. We keep such entries only for non-empty P_j, in increasing order of j.

Given a critical level ℓ, we scan the list $GREEDY$ and compute $P' := \cup_{j \leq \ell} P_j$ together with $X' := \cup_{p \in P'} \mathcal{X}(p)$ (represented as lists). We remove all the corresponding entries from $GREEDY$, and reset the corresponding values of $\ell(p)$, $S(p)$, $\mathcal{F}(p)$, and $\mathcal{X}(p)$.

Let us show how to build the data structures for the subinstance $(X', \mathcal{S}')$, with $\mathcal{S}' = \{S \cap X' : S \in \mathcal{S} \text{ and } S \cap X' \neq \emptyset\}$, in time $O(f|X'|)$. Let $n' = |X'|$ and $m' = |\mathcal{F}'| \leq fn'$. By scanning the entries of SET corresponding to X', we build the list of indexes $\mathcal{S}'$. We now map X' into a set of new indexes in $[1, O(n')]$ by means of a perfect hash function, and similarly map $\mathcal{F}'$ into a set of new indexes in $[1, O(m')]$. These perfect hash functions can be built in expected linear time using well-known constructions (say, 2-level hashing [20]). Observe that some indexes might not be used: we interpret those indexes as dummy elements and sets. Given this, we can easily build in linear time the data structures SET' and $ELEM'$ for the new instance.

The use of random hash functions can be avoided by assuming that we are given access to two arrays MAP_{elem} and MAP_{set} respectively of size n and $m \leq nf$ that are initialized to all zeros. We now map X' to $[1, n']$ and $\mathcal{S}'$ to $[1, m']$ as follows. We create vectors MAP^{-1}_{elem} and MAP^{-1}_{set} of size n' and fn', resp., which are initialized to zero. We iterate over X': when considering the j-th element of X of global id k (that is, it is the k-th element in $[1, n]$), then we set $MAP_{elem}[k] = j$ and $MAP^{-1}_{elem}[j] = k$. Similarly, we iterate over $\mathcal{S}'$ and update MAP_{set} and MAP^{-1}_{set} analogously. In order to handle possible duplicates in $\mathcal{S}'$ (and possibly in X'), we simply do not perform the update when we find an entry of MAP_{set} (or MAP_{elem}) that is already non-zero. Now, X' has been mapped to $[1, n']$ and $\mathcal{S}'$ has been mapped to $[1, m']$. This allows us to build vectors $ELEM'$ and SET' in the same way as above. Once the update phase has ended, we reset MAP_{elem} and MAP_{set} to 0 by iterating over MAP^{-1}_{elem} and MAP^{-1}_{set}. The overall process takes $O(fn')$ time.

We feed the vectors SET' and $ELEM'$ to the Random Cover subroutine that outputs a list P' of pivots, plus the associated values $\ell(p)$, $S(p)$, $\mathcal{F}(p)$, and $\mathcal{X}(p)$ for each $p \in P'$. Using MAP^{-1}_{elem} and MAP^{-1}_{set} we can map back the indexes of the corresponding elements and sets into the original indexes.

We remark that by construction, we will have $\ell(p) \leq \ell$. Now, we build a list $GREEDY'$ of the same type as $GREEDY$, however restricted to pivots in P' and to the respective levels. Finally, we concatenate $GREEDY'$ to the beginning of the list $GREEDY$.

We are now ready to prove Lemma 2. We first observe that, up to constant factors, the time taken in one level fixing process at critical level ℓ is at most that in the subsequent update phase. Indeed, recall that level fixing at level ℓ takes time $O(\ell^2)$. Since ℓ is the lowest critical level, $\ell - 1$ is not critical, therefore there is at least one deleted pivot in level ℓ. This implies that this update involves at least 2^ℓ elements, thus having cost $\Omega(2^\ell)$. The claim follows.

We can therefore focus on the cost of the update phase. Each pivot p at level i that participates in the update corresponds to at most f sets of size at most 2^{i+1} each. Hence the set X' of elements that participate in the update has size at most $\sum_{i \leq \ell} f p_i 2^{i+1}$. As argued above, we can build the corresponding set cover instance $(X', \mathcal{S}')$ and run the Random Cover subroutine on it in $O(f|X'|)$ time. This completes the proof of Lemma 2.

4.2 Fully Dynamic Algorithm

We now briefly describe the changes in the fully-dynamic algorithm that leads to Lemma 9. Along with each $\mathcal{X}(p)$, we also maintain two disjoint subsets of $\mathcal{X}(p)$: $Orig(p)$ and $Extra(p)$. When a set S is included in the current solution with pivot p and newly covers elements $\mathcal{X}(p)$, we follow the implementation details of the decremental case. In addition, we set $Orig(p) = \mathcal{X}(p)$ and $Extra(p) = \emptyset$. This does not change the asymptotic run time.

When an element e is inserted, we iterate over $SET(e)$ and check if there exists any set $S \in SET(e)$ present in the current solution. For each $S \in SET(e)$, the check can be implemented in $O(1)$ time by maintaining a Boolean variable for every $S \in \mathcal{S}$ and setting it to 1 whenever it is included in the solution. If $S \in \mathcal{S}$, we also maintain a pointer to its pivot p, and the value $\ell(p)$. If none of the sets in $SET(e)$ is present, then we include all sets in $SET(e)$ in the solution, mark e as the pivot for these sets and set $\ell(e) = 0$. We set

$\mathcal{X}(e) = \{e\}$, $Orig(e) = \{e\}$ and $Extra(e) = \emptyset$. This entire process can be implemented in $O(f)$ time as $|SET(e)| \leq f$. Thus, insertion takes $O(f)$ time.

Deletion without an update phase is implemented in the same way as in the decremental case and, therefore, takes $O(f)$ time as well. If there is an update phase at a critical level ℓ, given the instance $(X', \mathcal{S}')$, we first execute the movement step as follows. We scan every element $e \in X'$ and iterate over $SET(e)$ to check if there exists a set $S \in SET(e)$ that is in the current solution at a level strictly higher than ℓ. If so, we include e as an extra element in the highest such set and discard e from X'. This entire operation takes $O(f|X'|)$ time.

Let Y' be the uncovered elements at the end of the movement step. We run the covering step as in the decremental case over Y'. This takes $O(f|Y'|)$ time which is bounded by $O(f|X'|)$. Whenever a new pivot p is selected, the level of p and the corresponding sets are determined in $O(|\mathcal{S}(p)|)$ time. Accordingly, the lists L_is are updated. These updates take $O(1)$ time per set.

Thus, the update phase in the fully dynamic algorithm can be implemented in $O(f|X'|)$ time, thereby proving Lemma 9.

5 Conditional Lower Bounds for Dynamic Set Cover

A fast algorithm for a dynamic problem usually gives a fast algorithm for its static version. If we can solve dynamic Set Cover with preprocessing time $P(n, f)$ and update time $T(n, f)$, then we can solve the static Set Cover problem in $P(n, f) + n \cdot T(n, f)$ time: n updates are sufficient in order to create an offline instance. This simple connection immediately leads to some lower bounds for dynamic Set Cover. In particular, we get that it is NP-Hard to get an $o(\log n)$ approximation with polynomial preprocessing and update times. However, this connection does not give any lower bound in the polynomial time solvable regime of Set Cover where a $\min\{f, \log n\}$ approximation is possible in linear time. Could there be a dynamic algorithm with such approximation factors that has $o(f)$ or even $O(1)$ update time? This would not imply a new static algorithm for Set Cover.

Of course, such an algorithm is impossible if to insert an element we must explicitly specify the $O(f)$ sets it appears in. But in the model we consider, an update can be specified with much fewer bits. We assume that all elements X and sets $\mathcal{S}$ are given in advance, as well as all the membership information. Then, an update can add or remove an elements (in the Element-Update case) or sets (in the Set-Updates case). Only elements from X (or sets from $\mathcal{S}$) can be added or removed, and when an element is removed then a set-cover does not need to cover it. Notice that $O(\log |X|)$ bits are needed to specify an element insertion, and its membership in all the sets is known from the initial input. In this model, it is conceivable that an algorithm can spend $o(f)$ time per update and maintain some non-trivial approximation. The results in this section show that this is unlikely.

Under SETH, we show that no algorithm can preprocess an instance with m sets and n elements in $poly(n, m)$ time, and subsequently maintain element (or set) updates in $O(m^{1-\varepsilon})$ time, for any $\varepsilon > 0$, unless the approximation factor is essentially m^{δ}, for some $\delta > 0$. Note that a factor m approximation can be maintained trivially in constant time (pick either zero or all sets). and we show that essentially any $m^{o(1)}$ approximation algorithm requires $\Omega(m^{0.99})$ time.

THEOREM 3 (MAIN LOWER BOUND). *Let $n^{\omega(1)} < m < 2^{o(n)}$ and $t \geq 2$, such that $t = (n/\log m)^{o(1)}$. Assuming SETH, for all $\varepsilon > 0$, no dynamic algorithm can preprocess a collection of m sets over a universe $[n]$ in $poly(n, m)$ time, and then support element (or set) updates in $O(m^{1-\varepsilon})$ amortized time, and answer Set Cover queries in $O(m^{1-\varepsilon})$ amortized time with an approximation factor of t.*

We can state the following corollary in terms of the frequency bound f.

COROLLARY 1. *Assuming SETH, any dynamic algorithm for Set Cover on n elements and frequency bound f, where $n^{\omega(1)} < f < 2^{o(n)}$, that has polynomial time preprocessing and amortized update and query time $O(f^{1-\varepsilon})$, for some $\varepsilon > 0$, must have approximation factor at least $(n/\log f)^{\Omega(1)}$.*

PROOF. Without assuming anything about the instances in Theorem 3 we can conclude that $f \leq m$ while $m \leq nf < f^2$. Therefore, any approximation algorithm with factor $O((n/\log f)^{\delta})$ also gets an approximation of $O((n/2\log f)^{\delta}) = O((n/\log f^2)^{\delta})$ which is smaller than $O((n/\log m)^{\delta})$ and it is enough to refute SETH via Theorem 3. □

The rest of this section is dedicated to the proof of Theorem 3. Our starting point is the following SETH-based hardness of approximation result, which was proven first in [1] with a slightly smaller approximation factor, and was strengthened in [16] using, in part, the technique of [35]. These results use the distributed PCP framework of [1] for hardness of approximation results in P, and ours is the first application of this framework to dynamic problems.

THEOREM 4 ([1, 16, 35]). *Let $n^{\omega(1)} < m < 2^{o(n)}$ and $t \geq 2$, such that $t = (n/\log m)^{o(1)}$. Given two collections of m sets $\mathcal{A}, \mathcal{B}$ over a universe $[n]$, no algorithm can distinguish the following two cases in $O(m^{2-\varepsilon})$ time, for any $\varepsilon > 0$, unless SETH is false:*

YES case *there exist $A \in \mathcal{A}, B \in \mathcal{B}$ such that $B \subseteq A$; and*
NO case *for every $A \in \mathcal{A}, B \in \mathcal{B}$ we have $|A \cap B| < |B|/t$.*

From this theorem and standard manipulations it is easy to conclude the following statement. There are two differences in the statement below: first, the sizes of $\mathcal{A}$ and $\mathcal{B}$ are asymmetric, and second, the approximation is in terms of the number of sets required to cover a single $b \in B$, rather than the size of the overlap.

LEMMA 15. *Let $n^{\omega(1)} < m < 2^{o(n)}$ and $t \geq 2$, such that $t = (n/\log m)^{o(1)}$, and for all $0 < a \leq 1$. Given two collections of sets $\mathcal{A}, \mathcal{B}$ over a universe $[n]$, where $|\mathcal{B}| = m$ and $|\mathcal{A}| = m^a$, no algorithm can distinguish the following two cases in $O(m^{1+a-\varepsilon})$ time, for any $\varepsilon > 0$, unless SETH is false:*

YES case *there exist $A \in \mathcal{A}, B \in \mathcal{B}$ such that $B \subseteq A$; and*
NO case *there do not exist t sets $A_1, \ldots, A_t \in \mathcal{A}$, and a set $B \in \mathcal{B}$ such that $B \subseteq A_1 \cup \cdots \cup A_t$.*

PROOF. Assume for contradiction that such an algorithm exists. Given an instance $\mathcal{A}, \mathcal{B}$ of the problem in Theorem 4 we show how

to solve it in $O(m^{2-\varepsilon})$ time. Partition $\mathcal{A}$ into $k = m^{1-a}$ collections $\mathcal{A}_1, \ldots, \mathcal{A}_k$ of size m^a each, and invoke our algorithm on the asymmetric instance $\mathcal{A}_i, \mathcal{B}$ for each $i = 1 \cdots k$. The total time will be $k \cdot O(m^{1+a-\varepsilon}) = O(m^{2-\varepsilon})$. If the original (symmetric) instance was a YES case, then clearly at least one of the k asymmetric instances is a YES case. On the other hand, if it was a NO case, then any $A \in \mathcal{A}$ cannot cover more than a $1/t$ fraction of any set $B \in \mathcal{B}$ and therefore all the asymmetric instances are NO cases. □

Next we take this static set-containment problem and reduce it to dynamic Set Cover. We show two distinct reductions, a simpler one for the element updates case, and then a more complicated one with set updates.

5.1 Element Updates

Given an instance $\mathcal{A}, \mathcal{B}$ of the problem in Lemma 15, we construct an instance of dynamic Set Cover with approximation factor $(t-1)$ as follows. The universe $[n]$ will be the same, and all sets in $\mathcal{A}$ will appear in the instance. However, the sets in $\mathcal{B}$ will not, and they will be implemented implicitly in a dynamic way. Initially, all the universe elements are activated, and the algorithm may preprocess the instance. Note that the number of sets is only m^a.

For each set $B_i \in \mathcal{B}$ we will have a stage. We start the stage by removing from the universe all elements $e \in B_i$ that belong to B_i. After we do these $O(n)$ updates, we ask a Set Cover query. If the answer is less than t then we can stop and answer YES. Otherwise, we finish the stage by adding back all the elements that we removed and move on to the next stage. After we finish all m stages for all the sets in $\mathcal{B}$, we answer NO.

In total we have $O(nm)$ updates and queries, and so the final runtime is $P(n, m^a) + O(nm) \cdot (T(n, m^a) + Q(n, m^a))$. Assume we have an algorithm with update and query time $T(n, m^a) + Q(n, m^a) = O(m^{a\cdot(1-\varepsilon)})$ and polynomial preprocessing, $P(n, m^a) = O(m^{a\cdot c})$ for some $c \geq 1$, then we can choose $a = 1/c$ and get an algorithm for the problem in Lemma 15 with runtime $O(m^{1+a-\varepsilon a})$, contradicting SETH.

Finally, let us show the correctness of the answer. If we are in the YES case, then there is a set $B \in \mathcal{B}$ that is contained in some set in $\mathcal{A}$. When we ask a query at the stage corresponding to this set B, the size of the minimum set cover is 1. To see this note that all active universe elements are the elements of B and so we can cover all of them with some set in $\mathcal{A}$. Therefore, our $(t-1)$ approximation algorithm must output an answer that is less than t and we will output YES. On the other hand, if we are in the NO case, then in all stages, the size of the minimum set cover is at least t since at least t sets from $\mathcal{A}$ are required to cover any set in $\mathcal{B}$. Thus, the approximation algorithm will always return an answer that is at least t and we will never output YES.

5.2 Set Updates

The previous reduction fails in this case because we are only allowed to update sets, not elements. A natural approach for extending it is to have all sets from $\mathcal{B}$ in our instance and then at each stage we activate one of them. This would work, except that the number of sets grows to m which would only give us a weaker lower bound. Indeed such a simple reduction can rule out $O(m^{1-\varepsilon})$ update times if the preprocessing is restricted to take subquadratic time. A different idea is to add n auxiliary sets, one per element, so that this set only contains that element. Then, if we want to remove an element, we can add this set and somehow ensure that it is a part of the solution so that, effectively, the corresponding element is removed. This is the approach we take. The main challenge, however, is that these auxiliary sets have to be picked in our set cover solution and so they contribute to the size of the optimal solution. That is, we will no longer have a set cover of size 1 in the YES case and the gap between the YES and NO cases changes. To overcome this issue, we introduce another idea where we create many copies of everything and combine them into one instance in a certain way.

Given an instance $\mathcal{A}, \mathcal{B}$ of the problem in Lemma 15, we construct an instance of dynamic Set Cover with approximation factor $(t-1)$ as follows.

Our universe will be $k := n^2$ times larger, and for each element $e \in [n]$ in the universe of the original instance we will add k elements $e^1, \ldots, e^k$ to our instance. (So, our universe is isomorphic to $[kn]$.)

For each set $A \in \mathcal{A}$ we construct t sets $A^1, \ldots, A^k$ in our dynamic instance. All of these sets will remain activated throughout the reduction. The set A^i contains all elements e^i such that $e \in A$. That is, A^i contains the i^{th} copy of all the elements that were in A. Note that A^i does not contain e^j for any $i \neq j$.

We also add sets $S_1, \ldots, S_n$ which will be activated dynamically, and we let S_e contain all copies of the element $e \in [n]$. That is, S_e contains $e^1, \ldots, e^k$. These sets will allow us to simulate the deactivation of a set B.

Next we explain the dynamic part of the reduction. For each set $B \in \mathcal{B}$ we have a stage where we effectively deactivate all universe elements that are not in B. To do this, we activate the set S_e for all $e \notin B$ such that e is not in B. Note that we have activated up to n sets S_e, and that together they cover all copies of all elements that are in the complement of B. After we perform these $O(n)$ updates, we ask a Set Cover query. If the answer to the query is at most $(n+k)\cdot(t-1)$ we return YES. Otherwise, we undo the changes we made in this stage and we move on to the next $B \in \mathcal{B}$. After all the stages are done, we return NO.

The runtime analysis is similar to before since the only difference is in the universe size which increased from n to $kn = n^3$ but it is still $m^{o(1)}$. We have $O(nm)$ updates and queries, and so the final runtime is $P(nk, m^a) + O(nm) \cdot (T(nk, m^a) + Q(nk, m^a))$. Assume we have an algorithm with update and query time $T(nk, m^a) + Q(nk, m^a) = O(m^{a\cdot(1-\varepsilon)})$ and polynomial preprocessing, $P(nk, m^a) = O(m^{a\cdot c})$ for some $c \geq 1$, then we can choose $a = 1/c$ and get an algorithm for the problem in Lemma 15 with runtime $O(m^{1+a-\varepsilon a})$, contradicting SETH.

Finally, we show the correctness of the answer. For the YES case, there is a set $B \in \mathcal{B}$ that is contained in some set in $\mathcal{A}$. When we ask a query at the stage corresponding to this set B, the size of the minimum set cover is at most $n+k$. This is because of the following set cover: Choose all sets S_e that are active in this stage; this cover all copies of all universe elements that are not in B. Then choose all copies A^i of the set $A \in \mathcal{A}$ that contains B; this covers all copies of all elements that are in B. Therefore, our $(t-1)$ approximation algorithm must output an answer that is at most $(n+k)(t-1)$ and we will output YES. On the other hand, in the NO case, the size of the minimum set cover is at least $k \cdot t$ in every stage. This is because

at least t sets from $\mathcal{A}$ are required to cover any set in $\mathcal{B}$, and in a stage of some set B the only way we can cover copies of elements that belong to B is by choosing copies of sets A that contain them. There are k copies of the universe elements, and for each such copy we have to choose at least t sets from A to cover the elements of that copy, and these sets do not contain any elements from any other copy of the universe. Thus, the approximation algorithm will always return an answer that is at least kt, which is larger than $(n+k)(t-1)$ since $k = n^2$ and $t = n^{o(1)}$, and we will never output YES.

Acknowledgements. The authors are grateful to Shay Solomon for pointing out an error in a preliminary version of this paper. The authors would also like to thank the anonymous reviewers for their insightful comments. F. Grandoni is partially supported by the SNSF grants 200021_159697/1 and 200020B_ 182865/1. D. Panigrahi is partially supported by NSF contracts CCF 1535972, CCF 1527084, an NSF CAREER Award CCF 1750140, and the Indo-US Virtual Networked Joint Center on Algorithms under Uncertainty. B. Saha is partially supported by an NSF CRII grant CCF 1464310, an NSF CAREER Award CCF 1652303, and an Alfred P. Sloan fellowship.

References

[1] Amir Abboud, Aviad Rubinstein, and R. Ryan Williams. 2017. Distributed PCP Theorems for Hardness of Approximation in P. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*. 25–36.

[2] Amir Abboud and Virginia Vassilevska Williams. 2014. Popular conjectures imply strong lower bounds for dynamic problems. In *Foundations of Computer Science (FOCS)*. 434–443.

[3] Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. 2018. Matching triangles and basing hardness on an extremely popular conjecture. *SIAM J. Comput.* 47, 3 (2018), 1098–1122.

[4] Ittai Abraham, Shiri Chechik, and Sebastian Krinninger. 2017. Fully dynamic all-pairs shortest paths with worst-case update-time revisited. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 440–452.

[5] Ittai Abraham, Shiri Chechik, and Sebastian Krinninger. 2017. Fully dynamic all-pairs shortest paths with worst-case update-time revisited. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 440–452.

[6] Sepehr Assadi and Sanjeev Khanna. 2018. Tight bounds on the round complexity of the distributed maximum coverage problem. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2412–2431.

[7] Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. 2018. Fully dynamic maximal independent set with sublinear update time. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*. ACM, 815–826.

[8] Surender Baswana, Manoj Gupta, and Sandeep Sen. 2015. Fully Dynamic Maximal Matching in $O(\log n)$ Update Time. *SIAM J. Comput.* 44, 1 (2015), 88–113.

[9] Aaron Bernstein and Cliff Stein. 2016. Faster fully dynamic matchings with small approximation ratios. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 692–711.

[10] Sayan Bhattacharya, Deeparnab Chakrabarty, and Monika Henzinger. 2017. Deterministic fully dynamic approximate vertex cover and fractional matching in O (1) amortized update time. In *International Conference on Integer Programming and Combinatorial Optimization*. Springer, 86–98.

[11] Sayan Bhattacharya, Deeparnab Chakrabarty, Monika Henzinger, and Danupon Nanongkai. 2018. Dynamic Algorithms for Graph Coloring. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 1–20.

[12] Sayan Bhattacharya, Monika Henzinger, and Giuseppe F Italiano. 2015. Design of dynamic algorithms via primal-dual method. In *International Colloquium on Automata, Languages, and Programming*. Springer, 206–218.

[13] Sayan Bhattacharya, Monika Henzinger, and Giuseppe F Italiano. 2015. Deterministic fully dynamic data structures for vertex cover and matching. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 785–804.

[14] Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. 2016. New deterministic approximation algorithms for fully dynamic matching. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*. ACM, 398–411.

[15] Shiri Chechik, Thomas Dueholm Hansen, Giuseppe F. Italiano, Jakub Lacki, and Nikos Parotsidis. 2016. Decremental Single-Source Reachability and Strongly Connected Components in $\tilde{O}(m\sqrt{n})$ Total Update Time. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*. 315–324.

[16] Lijie Chen. 2018. On The Hardness of Approximate and Exact (Bichromatic) Maximum Inner Product. In *33rd Computational Complexity Conference, CCC 2018, June 22-24, 2018, San Diego, CA, USA*. 14:1–14:45.

[17] Camil Demetrescu and Giuseppe F. Italiano. 2004. A New Approach to Dynamic All Pairs Shortest Paths. *J. ACM* 51, 6 (Nov. 2004), 968–992.

[18] Irit Dinur and David Steurer. 2014. Analytical approach to parallel repetition. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*. ACM, 624–633.

[19] Greg N Frederickson. 1985. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM J. Comput.* 14, 4 (1985), 781–798.

[20] Michael L. Fredman, János Komlós, and Endre Szemerédi. 1984. Storing a Sparse Table with 0(1) Worst Case Access Time. *J. ACM* 31, 3 (1984), 538–544.

[21] Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Debmalya Panigrahi. 2017. Online and dynamic algorithms for set cover. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*. ACM, 537–550.

[22] Manoj Gupta and Richard Peng. 2013. Fully dynamic (1+ e)-approximate matchings. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*. IEEE, 548–557.

[23] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. 2015. Unifying and Strengthening Hardness for Dynamic Problems via the Online Matrix-Vector Multiplication Conjecture. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*. 21–30.

[24] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. 2015. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*. ACM, 21–30.

[25] Monika R Henzinger and Valerie King. 1999. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *Journal of the ACM (JACM)* 46, 4 (1999), 502–516.

[26] Jacob Holm, Kristian De Lichtenberg, and Mikkel Thorup. 2001. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *Journal of the ACM (JACM)* 48, 4 (2001), 723–760.

[27] Piotr Indyk, Sepideh Mahabadi, Ronitt Rubinfeld, Jonathan Ullman, Ali Vakilian, and Anak Yodpinyanee. 2017. Fractional set cover in the streaming model. In *LIPIcs-Leibniz International Proceedings in Informatics*, Vol. 81. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[28] Giuseppe F. Italiano, Adam Karczmarz, Jakub Lacki, and Piotr Sankowski. 2017. Decremental single-source reachability in planar digraphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*. 1108–1121.

[29] Subhash Khot and Oded Regev. 2008. Vertex cover might be hard to approximate to within 2- ε. *J. Comput. System Sci.* 74, 3 (2008), 335–349.

[30] Tsvi Kopelowitz, Seth Pettie, and Ely Porat. 2016. Higher Lower Bounds from the 3SUM Conjecture. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*. 1272–1287.

[31] Andrew McGregor and Hoa T Vu. 2016. Better streaming algorithms for the maximum coverage problem. *arXiv preprint arXiv:1610.06199* (2016).

[32] Ofer Neiman and Shay Solomon. 2016. Simple deterministic algorithms for fully dynamic maximal matching. *ACM Transactions on Algorithms (TALG)* 12, 1 (2016), 7.

[33] Krzysztof Onak and Ronitt Rubinfeld. 2010. Maintaining a large matching and a small vertex cover. In *Proceedings of the forty-second ACM symposium on Theory of computing*. ACM, 457–464.

[34] Krzysztof Onak, Baruch Schieber, Shay Solomon, and Nicole Wein. 2018. Fully Dynamic MIS in Uniformly Sparse Graphs. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*. 92:1–92:14.

[35] Aviad Rubinstein. 2018. Hardness of approximate nearest neighbor search. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*. 1260–1268.

[36] Shay Solomon. 2016. Fully dynamic maximal matching in constant update time. In *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*. IEEE, 325–334.

[37] Christian Wulff-Nilsen. 2017. Fully-dynamic minimum spanning forest with improved worst-case update time. In *49th ACM Symposium on Theory of Computing*. ACM, 1130–1143.