Nearly Optimal Dynamic Set Cover: Breaking the Quadratic-in-f Time Barrier

Anton Bukov [*] Shay Solomon [†] Tianyi Zhang [‡]

Abstract

The dynamic set cover problem has been subject to extensive research since the pioneering works of [BHI, ICALP'15] and [GKKP17, STOC'17]. The input is a set system $(\mathcal{U}, \mathcal{S})$ on a fixed collection $\mathcal{S}$ of sets and a dynamic universe of elements, where each element appears in a most f sets and the cost of each set lies in the range $[1/C, 1]$; the ultimate goal is to maintain a set cover under insertions and deletions of elements, with optimal bounds on both the approximation factor and the update time.

Most previous works considers the low-frequency regime, namely $f = O(\log n)$, and this line of work has culminated with a deterministic $(1+\epsilon)f$-approximation algorithm with amortized update time $O(\frac{f^2}{\epsilon^3} + \frac{f}{\epsilon^2}\log C)$ [BHNW, SODA'21] and a randomized f-approximation algorithm against an oblivious adversary with expected amortized update time $O(f^2)$ for the unweighted case [AS, ESA'21]. In the high-frequency regime of $f = \Omega(\log n)$, an $O(\log n)$-approximation algorithm with amortized update time $O(f \log n)$ was given by [GKKP17, STOC'17], and recently [SU, STOC'23] showed that the same update time of $O(f \log n)$ suffices for achieving approximation $(1+\epsilon)\ln n$.

Interestingly, at the intersection of the two regimes, i.e., $f = \Theta(\log n)$, the state-of-the-art results coincide (ignoring the dependencies on ϵ and C): approximation $\Theta(f) = \Theta(\log n)$ with amortized update time $O(f^2) = O(f \log n) = O(\log^2 n)$. Up to this date, no previous work achieved update time of $o(f^2)$, even allowing randomization against an oblivious adversary and even for a worse approximation guarantee.

In this paper we break the $\Omega(f^2)$ update time barrier via the following results:

- $(1+\epsilon)f$-approximation can be maintained in $O\left(\frac{f}{\epsilon^2}\log^* f + \frac{f}{\epsilon^3} + \frac{f}{\epsilon^2}\log C\right) = O_{\epsilon,C}(f \log^* f)$ expected amortized update time [1]; our algorithm works against an adaptive adversary.
- $(1+\epsilon)f$-approximation can be maintained deterministically in $O\left(\frac{1}{\epsilon} f \log f + \frac{f}{\epsilon^3} + \frac{f}{\epsilon^2}\log C\right) = O_{\epsilon,C}(f \log f)$ amortized update time.

Assuming element updates are specified *explicitly*, our randomized algorithm is near-optimal: $(1+\epsilon)f$ approximation is optimal up to the ϵ-dependence and the update time $O_{\epsilon,C}(f \log^* f)$ exceeds the time needed to specify an update by a $\log^* f$ factor. We view this slack of $\log^* f$ factor as interesting in its own right — we are not aware of any problem for which the state-of-the-art dynamic algorithm admits a slack of $\log^* f = O(\log^* n)$ from optimality.

[*]Tel Aviv University, bukov.anton@gmail.com

[†]Tel Aviv University, shayso@tauex.tau.ac.il

[‡]Tel Aviv University, tianyiz21@tauex.tau.ac.il

[1]$\log^*$ is the iterated logarithm; we use the notation $O_{\epsilon,C}(\cdot)$ to suppress factors that depend on ϵ and C.

Contents

1 Introduction

In the static set cover problem, we are given a set system $(\mathcal{U}, \mathcal{S})$, where $\mathcal{U}$ is a universe of n elements and $\mathcal{S}$ is a collection of m sets $s \in \mathcal{S}$ of elements in $\mathcal{U}$, each of which associated with a cost $c_s \in [\frac{1}{C}, 1]$. The *frequency* of the set system $(\mathcal{U}, \mathcal{S})$, denoted by $f = f(\mathcal{U}, \mathcal{S})$, is the maximum number of sets in $\mathcal{S}$ any element in $\mathcal{U}$ belongs to. A collection of sets $\mathcal{S}' \subseteq \mathcal{S}$ is called a *set cover* of $\mathcal{U}$ if any element in $\mathcal{U}$ belongs to at least one set in $\mathcal{S}'$. The basic goal is to compute a *minimum set cover*, i.e., a set cover $\mathcal{S}^* \subseteq \mathcal{S}$ whose cost $c(\mathcal{S}^*) = \sum_{s \in \mathcal{S}^*} c_s$ is minimum.

The set cover problem is a central NP-hard problem, which admits two classic algorithms: a *greedy* $\ln n$-approximation and a primal-dual f-approximation. Importantly, one cannot achieve approximation $(1-\epsilon)\ln n$ unless $\mathrm{P} = \mathrm{NP}$ [WS11, DS14] as well as approximation $f - \epsilon$ for any fixed f under the unique games conjecture [KR08]. The greedy and primal-dual approximation algorithms for set cover have been extremely well-studied in the static setting and are well-understood by now, and an extensive body of work from recent years aims at efficiently "dynamizing" these algorithms. In the *dynamic* setting of set cover, the goal is to maintain a set cover of low cost, while the universe $\mathcal{U}$ evolves over time. More specifically, the goal is to maintain a set cover $T \subseteq \mathcal{S}$ of low cost while supporting two types of element updates:

- **Insertion.** A new element e enters $\mathcal{U}$, and the input specifies the sets in $\mathcal{S}$ that it belongs to.
- **Deletion.** An existing element in $\mathcal{U}$ is deleted from all sets in $\mathcal{S}$ that it belonged to.

The two main quality measures of a dynamic algorithm are its approximation ratio $\frac{c(T)}{c(S^*)}$ and update time, where the holy grail is to achieve approximation approaching that of the best *static* algorithm with as small as possible update time. In the context of set cover: (1) for approximation, given the aforementioned lower bounds, the goal would be either an $O(\log n)$ or $O(f)$ approximation, and (2) for update time, since it takes $\Theta(f)$ time to explicitly represent an element update (by specifying all the sets to which it belongs), the natural goal would be update time $O(f)$.

The dynamic set cover problem was first studied in [BHI15], where a deterministic primal-dual algorithm with $O(f^2)$ approximation and $O(f\log(m+n))$ (amortized) update time was presented.[2] Later on, a deterministic $O(\log n)$-approximation algorithm with $O(f\log n)$ update time was given in [GKKP17]. This work of [GKKP17] essentially "dynamizes" the greedy algorithm in the high-frequency regime, namely $f = \Omega(\log n)$. In a recent work [SU23], the authors improved the approximation to $(1+\epsilon)\ln n$ with $O\left(\frac{f\log n}{\epsilon^5}\right)$ amortized update time.

All other previous works, which we survey next, focus on the low-frequency regime of $f = O(\log n)$, and they all essentially dynamize the primal-dual algorithm. A deterministic $O(f^3)$-approximation algorithm with $O(f^2)$ update time was achieved in [GKKP17, BCH17]. The first $O(f)$ approximation was achieved in [AAG+19], where the authors proposed a randomized $(1+\epsilon)f$-approximation algorithm with update time $O(\frac{f^2\log n}{\epsilon})$; this algorithm works for unweighted instances only (where $c_s \equiv 1$ for all s) and it assumes an oblivious adversary. This result was subsumed by [BHN19], where a deterministic $(1+\epsilon)f$-approximation algorithm for weighted instances was presented, with update time of $O(\frac{f}{\epsilon^2}\log(Cn))$.

The works of [AAG+19, BHN19] with $(1+\epsilon)f$-approximation incur a slack of $\log n$ on the update time. Two subsequent works remove the dependency on $\log n$: [BHNW21] gave a deterministic $(1+\epsilon)f$-approximation algorithm with update time $O(\frac{f^2}{\epsilon^3}+\frac{f}{\epsilon^2}\log C)$, while [AS21] gave a randomized f-approximation algorithm with update time $O(f^2)$, but it assumes an oblivious adversary, and it only applies to unweighted instances.

[2]For brevity, in what follows we shall not make the distinction between amortized and worst-case update time.

To summarize, in the low frequency regime of $f = O(\log n)$, no previous work achieved update time of $o(f^2)$, even allowing randomization against an oblivious adversary and even for approximation larger than $O(f)$. For the high frequency regime of $f = \Omega(\log n)$, the only previous work achieves update time $O(f \log n)$ [GKKP17]; interestingly, at the intersection of the two regimes, i.e., $f = \Theta(\log n)$, the state-of-the-art results coincide (ignoring the dependencies on ϵ and C): approximation $\Theta(f) = \Theta(\log n)$ with amortized update time $O(f^2) = O(f \log n) = O(\log^2 n)$.

A fundamental question left open by previous works is whether one can break the quadratic-in-f update time barrier, ideally to achieve an update time of $O_{\epsilon,C}(f)$ (ignoring the dependencies on ϵ and C), i.e., linear in the time needed to explicitly specify an update.

Question 1.1. *Is there $O(f)$-approximation (or $O(\log n)$-approximation) algorithm for set cover with update time $o(f^2)$? Further, it is possible to achieve approximation approaching f (or $\ln n$) with update time approaching $O(f)$?*

Perspective: The Quest Towards Optimal Update Time. The quest towards *constant update time* algorithms for basic graph problems is an important research agenda in the field of dynamic graph algorithms, which has attracted a lot of research attention over the past decade [PS16, Sol16, BCH17, BGM17, GKKP17, SW18, BK19, HP20, BHNW21, AS21, BGK+22, BCPS23]. This research agenda coincides with the quest towards linear-time graph algorithms in the static sequential setting, since any constant update time algorithm (that uses at most linear time during preprocessing) gives rise to a linear-time static algorithm. Of course, not every dynamic graph problem admits a constant update time solution, even if the respective static problem admits a linear running time, and graph connectivity is a prime example [PD06, PT11].

The set cover problem is equivalent to the vertex cover problem in hypergraphs, where the *rank* of the hypergraph is the frequency f of the set-system. One can generalize any graph problem for hypergraphs, with one significant caveat: In dynamic hypergraphs, the time needed to explicitly specify an edge update is no longer constant, but rather $O(f)$. One may consider implicit updates instead (switching an edge "on" and "off"), which can be carried out in constant time, but even for implicit updates there are conditional lower bounds on the update time that are not far from $\Omega(f)$, albeit only for sufficiently high frequency; refer to [AAG+19] for details. Nonetheless, even ignoring such conditional lower bounds, the $O(f)$ time bound to explicitly specify an update in rank-f hypergraphs seems the natural generalization of constant update time in simple graphs for hypergraphs, and is thus a natural time barrier.

To the best of our knowledge, the previous work on the dynamic set cover problem provides the *first systematic study on any dynamic hypergraph problem.* Moreover, we are not aware of any nontrivial hypergraph problem that is solved within update time $O(f)$. Consequently, whether it is possible to fully resolve Question 1.1 — and obtain the *first update time of $O(f)$ for any rank-f hypergraph problem* (with a reasonably good approximation) — seems to be of major importance.

1.1 Our result

Our main result, which resolves Question 1.1 in the affirmative, is summarized in the following theorem; Table 1 provides a concise comparison between our and previous results.

Theorem 1.1. *For any set system $(\mathcal{U}, \mathcal{S})$ (with $\mathcal{U} = \emptyset$ initially) that undergoes a sequence of element insertions and deletions, where the frequency is always bounded by f, and for any $\epsilon \in (0, 0.1)$, there are dynamic algorithms that maintain a $(1+\epsilon)f$-approximate minimum set cover with the following amortized update time bounds.*

reference	approximation	update time	deterministic?	weighted?
[GKKP17]	$O(\log n)$	$O(f \log n)$	yes	yes
[SU23]	$(1+\epsilon)\ln n$	$O\left(\frac{f\log n}{\epsilon^5}\right)$	yes	yes
[BHI15]	$O(f^2)$	$O(f\log(m+n))$	yes	yes
[GKKP17, BCH17]	$O(f^3)$	$O(f^2)$	yes	yes
[AAG$^+$19]	$(1+\epsilon)f$	$O\left(\frac{f^2}{\epsilon}\log n\right)$	oblivious	no
[BHN19]	$(1+\epsilon)f$	$O\left(\frac{f}{\epsilon^2}\log(Cn)\right)$	yes	yes
[BHNW21]	$(1+\epsilon)f$	$O\left(\frac{f^2}{\epsilon^3}+\frac{f}{\epsilon^2}\log C\right)$	yes	yes
[BHNW21]	$(1+\epsilon)f$	$O\left(f\log^2(Cn)/\epsilon^3\right)$ (wc)	yes	yes
[AS21]	f	$O\left(f^2\right)$	oblivious	no
new	$(1+\epsilon)f$	$O\left(\frac{f}{\epsilon^2}\log^* f+\frac{f}{\epsilon^3}+\frac{f}{\epsilon^2}\log C\right)$	adaptive	yes
new	$(1+\epsilon)f$	$O\left(\frac{1}{\epsilon}f\log f+\frac{f}{\epsilon^3}+\frac{f}{\epsilon^2}\log C\right)$	yes	yes

Table 1: Summary of results on dynamic set cover. "wc" stands for "worst case".

- *Expected* $O\left(\frac{f}{\epsilon^2}\log^* f+\frac{f}{\epsilon^3}+\frac{f}{\epsilon^2}\log C\right)$, *via a randomized algorithm* against an adaptive adversary.
- *Deterministic* $O\left(\frac{1}{\epsilon}f\log f+\frac{f}{\epsilon^3}+\frac{f}{\epsilon^2}\log C\right)$.

Remark. For our deterministic algorithm, we shall assume for simplicity that the length of the update sequence is at least $\frac{1}{\epsilon}m\log(Cn)$. In this way, during preprocessing (before the first element is inserted to $\mathcal{U}$), we prepare a data structure of size $O(\frac{1}{\epsilon}m\log(Cn))$. (The same is done implicitly in previous works whose amortized update time is independent of n [BK19, BHNW21, AS21].) In these algorithms (including ours), all elements $e \in \mathcal{U}$ are assigned a level value $0 \le \mathsf{lev}(e) \le \lceil\log_{1+\epsilon}(Cn)\rceil+1$, and for each set $s \in \mathcal{S}$, we maintain a list of all elements $E_i(s) = \{e \in s, \mathsf{lev}(e) = i\}$ (in our algorithm, sets $A_i(s)$ and $P_i(s)$, which are defined a bit differently). Since the pointer to each set $E_i(s)$ needs to be retrieved in $O(1)$ time given the index i, we maintain an array of length $O(\log_{1+\epsilon}(Cn))$ storing all the pointers, even if some sets $E_i(s)$ might be empty. (For our randomized algorithm, we can simply use dynamic hash tables [DKM$^+$94].)

We emphasize two points regarding our randomized algorithm.

- It works against an adaptive adversary; this is the first randomized algorithm for dynamic set cover that does not make the assumption of an oblivious adversary.
- Assuming element updates are specified *explicitly*, the update time $O_{\epsilon,C}(f\log^* f)$ exceeds the time needed to specify an update by a $\log^* f$ factor. This slack of $\log^* f$ factor is interesting in its own right — **we are not aware of any problem for which the state-of-the-art dynamic algorithm admits a slack of** $\log^* f = O(\log^* n)$ **from optimality**. (A notable example where such a slack was studied is for the Disjoint-set data structure, where a highly influential line of work improved the $O(\log^* n)$ bound to an inverse-Ackermann bound, later shown to be tight.)

1.2 Technical and Conceptual Contribution

Our algorithm builds upon the primal-dual framework from [BHI15, BK19, BHN19, BHNW21]. In the primal-dual framework, all sets in $s \in \mathcal{S}$ are assigned to levels $\mathsf{lev}(s)$ numbered from 0 to

$L = \lceil \log_{1+\epsilon}(Cn) \rceil + 1$. For each element $e \in \mathcal{U}$, its level $\mathsf{lev}(e)$ is defined as the maximum level of any set it belongs to, namely $\mathsf{lev}(e) = \max_{s \ni e}\{\mathsf{lev}(s)\}$. This hierarchical partition of sets and elements into levels defines weights for elements and sets: Each element e is assigned a weight $\omega(e) = (1+\epsilon)^{-\mathsf{lev}(e)}$, and the weight $\omega(s)$ of each set s is given as the total weight of elements in it, namely $\omega(s) = \sum_{e \in s} \omega(e)$. A set s is called *tight* if $\omega(s) \geq c_s/(1+\epsilon)$. The primal-dual framework maintains a hierarchical partition into levels as above, aiming to satisfy the following invariants.

- $\omega(s) \leq c_s, \forall s \in \mathcal{S}$.
- All sets on level > 0 are tight.

If both invariants are met, then weak duality implies that the set $T \subseteq \mathcal{S}$ of all tight sets provides a $(1+\epsilon)f$-approximate set cover, i.e., $c(T) \leq (1+\epsilon)f \cdot c(\mathcal{S}^*)$.

Local approach. To dynamically maintain an approximate minimum set cover via the primal-dual framework, it is perhaps most natural to employ the so-called *local approach*: Each time an element is inserted or deleted, the algorithm will perform some *local* "fixing" steps "around the update" to recover both invariants, to restore a valid and up-to-date hierarchical partition (including up-to-date induced weights). This local approach, which was implemented in [BHI15], has two drawbacks: (1) The update time is $O(f \log(m+n))$, which in particular depends on m, n, and (2) the approximation ratio is $O(f^2)$ rather than $O(f)$. To shave the $\log n$ factor in the update time, [BK19] studied the special case of vertex cover, and introduced a new analysis of the local approach that improves the update time to $O(1)$. Although this new analysis of the local approach generalizes for set cover, it does not fix the second drawback of approximation $O(f^2)$.

Global approach. To obtain a $(1+\epsilon)f$-approximation, the subsequent works [BHN19, AAG$^+$19] adopted a *global approach* to maintain the primal-dual hierarchical partition. Basically, instead of recovering the invariants persistently after every element update, the global approach only handles the updates in the following lazy manner.

For each insertion of some element e, if we insist that $\omega(e) = (1+\epsilon)^{-\mathsf{lev}(e)}$, then $\omega(s)$ for some sets $s \ni e$ might exceed c_s; to satisfy the first invariant, we would have to raise the level of such sets, which might set off a long cascade of level changes of elements and sets. The lazy approach would be to simply assign the largest possible weight $\omega(e) = (1+\epsilon)^{-l}$ without violating any constraints $\omega(s) \leq c_s, s \ni e$. In this way, we have relaxed the requirement that $\omega(e)$ is equal to $(1+\epsilon)^{-\mathsf{lev}(e)}$ by assigning it a smaller weight $(1+\epsilon)^{-\mathsf{ilev}(e)}$ for some *intrinsic* level $\mathsf{ilev}(e)$. This relaxation naturally partitions all existing elements into two categories: (1) *active* elements e where $\omega(e) = (1+\epsilon)^{-\mathsf{lev}(e)}$, and (2) *passive* elements e where $\omega(e) = (1+\epsilon)^{-\mathsf{ilev}(e)} < (1+\epsilon)^{-\mathsf{lev}(e)}$.

For each deletion of some element e, we simply ignore it, and when deletions have accumulated to a large extent, a rebuild procedure is invoked, which rebuilds a carefully chosen "prefix" of the primal-dual hierarchical partition. Roughly speaking, when the approximation of the current set cover might exceed $(1+\epsilon)f$, the algorithm of [BHN19] looks for the lowest level k such that the fraction of deleted elements on levels $\leq k$ is large. Then the entire primal-dual hierarchy from levels 0 to k is rebuilt by first moving all existing elements on levels $\leq k$ to level $k+1$ and then pushing them downward using a discretized water-filling procedure. This ensures that for any element e that remains passive, the gap $\mathsf{ilev}(e) - \mathsf{lev}(e)$ decreases. It can be shown that the runtime of the rebuild procedure is $O(f|A_{\leq k}| + f|P_{\leq k}|)$, where $A_{\leq k}, P_{\leq k}$ are the sets of active and passive elements that lied on levels $\leq k$ before the rebuild, respectively. For the amortized analysis, the term $f|A_{\leq k}|$ can be charged to the deletions that have accumulated, and the term $f|P_{\leq k}|$ can be charged (via a potential function analysis) to the decrease of gaps $\mathsf{ilev}(e) - \mathsf{lev}(e), e \in P_{\leq k}$. Using the fact that the gap $\mathsf{ilev}(e) - \mathsf{lev}(e)$ is bounded by $O(\log n)$, an amortized update time of $O(f \log n)$ is derived.

Combining local and global approaches. To shave the $\log n$ factor while preserving a $(1+\epsilon)f$ approximation, [BHNW21] combines the local approach with the global approach in the following way. For insertion e, they assign the true weight $\omega(e) = (1+\epsilon)^{-\mathsf{lev}(e)}$, and apply the local approach from [BK19] to fix the violated constraints of the first invariant, if any. For deletion e, they follow the same rebuild procedure from [BHN19]. Now there is no dependency on $\log n$, since every element is always active (and the gap $\mathsf{ilev}(e) - \mathsf{lev}(e)$ does not exist at all).

Alas, this approach incurs a quadratic dependency on f. Indeed, in the analysis of [BHNW21], which uses a potential function $\Phi(\cdot)$, each newly inserted element e adds roughly $\omega(e) \cdot f(1+\epsilon)^{\mathsf{lev}(s)}$ units to the potential $\Phi(s)$ of element $s \ni e$, and summing over all up to f sets $s \ni e$, the total potential increase could be as large as f^2.

1.2.1 Our Approach

A careful balance between local and global approaches. To improve over previous works, and in particular to bypass the quadratic-in-f time barrier in [BHNW21], we seek a better balance between the local and global approaches. On the one hand, to avoid the quadratic-in-f potential increase due to an element insertion, we will still allow e to be passive, so that we can avoid the heavy cost that is incurred by the local approach to fix the violated constraints. On the other hand, we do not want e to be *too passive*, so that e does not participate in too many instances of rebuilding before it becomes active, as this might blow up the update time by a factor of $\log n$. To express this idea in terms of levels, we would like to balance two contradictory requirements: the first is that the gap $\mathsf{ilev}(e) - \mathsf{lev}(e)$ would be large, while the second is that the gap $\mathsf{ilev}(e) - \mathsf{lev}(e)$ would be small.

To optimize the balance, we need to overcome several highly nontrivial technical hurdles. Our resulting algorithm is inherently different than the previous ones, and so is our analysis. We next sketch the core idea of the argument (ignoring most of the technical issues that arise). When an element e is inserted, we will assign $\mathsf{ilev}(e) = \mathsf{lev}(e) + \log_{1+\epsilon} f$, which bounds the gap $\mathsf{ilev}(e) - \mathsf{lev}(e)$ by $O(\log f)$. On the one hand, we can show that the total potential increase due to fixing the violated constraints would be smaller by a factor of f, as compared to [BK19]; to fix the violated constraints, we basically follow the same local approach as in previous works (with several important modifications, which we skip here). On the other hand, if there are no violated constraints with respect to the intrinsic level $\mathsf{ilev}(e) = \mathsf{lev}(e) + \log_{1+\epsilon} f$ assigned to e, we can make sure that the total time spent on e would be $O(f \log f)$. More specifically, the algorithm will carefully make sure that the gap $\mathsf{ilev}(e) - \mathsf{lev}(e)$ never increases, which is a key technical challenge that the algorithm and analysis must face. Moreover, each time the passive element e participates in a call to the rebuild procedure, the gap $\mathsf{ilev}(e) - \mathsf{lev}(e)$ will decrease by at least one. Therefore e can participate in at most $\log_{1+\epsilon} f$ calls to the rebuild procedure, which we show ultimately implies that the total time spent on e is $O(f \log f)$.

Going below $O(f \log f)$ update time: sampling and bootstrapping. To go below $O(f \log f)$ update time, let us take a closer look at the rebuild procedure. For each passive element $e \in P_{\leq k}$, in previous works, one had to scan all the sets $s \ni e$ to test whether e can be activated on level $k+1$ (whether $\omega(s) - \omega(e) + (1+\epsilon)^{-k-1} \leq c_s$ is not violated for all $s \in e$), which takes time $O(f)$. The worst-case performance of the algorithm occurs when such tests always fail, so that one always pays $O(f)$ time to decrease the gap $\mathsf{ilev}(e) - \mathsf{lev}(e)$ by one. To improve the runtime, we would like to be able to decrease this gap *exponentially*, i.e., from $d = \mathsf{ilev}(e) - \mathsf{lev}(e)$ to $\log d$; Alas, this is not always possible. To overcome this hurdle, our key insight is to only sample $O(f/\log f)$ sets $s \ni e$ and test whether e can be activated with respect to all sampled sets (whether $\omega(s) - \omega(e) + (1+\epsilon)^{-k-1} \leq c_s$ is not violated for all sampled sets). If there are at least $10\log^2 f$ *witness* sets s for which the test is violated, then one of them will be sampled with good probability, and in that case we have shaved off a $\log f$ factor from the time needed to process e due to the rebuild procedure. Otherwise, we will push down the intrinsic level of e from level $k + 1 + \log_{1+\epsilon} f$ to level $k + 1 + 2\log_{1+\epsilon}\log_{1+\epsilon} f$, which increases $\omega(e)$ to $\frac{1}{\log^2_{1+\epsilon} f}(1+\epsilon)^{-k-1}$, and then apply the local approach to fix the violated constraints. A crucial observation is that we know that the total number of violations is bounded by $\log^2_{1+\epsilon} f$, which is exponentially smaller than the trivial bound f, and so we can bound the potential increase by $O(f)$ instead of $O(f^2)$. We demonstrate that by a careful repetition of this observation, the gap $\mathsf{ilev}(e) - \mathsf{lev}(e)$ can be reduced *exponentially* in $O(f)$ time, which ultimately leads to the desired update time of $O(f \log^* f)$.

Summary. The starting point of our work is the aforementioned dynamic primal-dual algorithms for set cover. However, to break the quadratic-in-f time barrier, and further to achieve the near-optimal (up to the $\log^* f$ slack factor) update time, we had to deviate significantly from previous works. The facts that our approach provides (1) the only randomized set cover algorithm that works against an adaptive adversary, and (2) a rare example of achieving optimal time to within a $\log^* n$ factor — may serve as some "evidence" for the novelty of our algorithm and its analysis.

2 Preliminaries

Definition 2.1. *For any real values $\epsilon \in (0,1), y \geq 1$ and integer $\eta \geq 1$, inductively define:*

$$(5\log)^{(\eta)}_{1+\epsilon} y = 5 \cdot \log_{1+\epsilon}\left((5\log)^{(\eta-1)}_{1+\epsilon} y\right)$$

where $(5\log)^{(0)}_{1+\epsilon} y = y$, and define $(5\log)^*_{1+\epsilon}(y)$ to be the minimum value of index η such that $(5\log)^{(\eta)}_{1+\epsilon}(y) \leq \frac{200}{\epsilon^2}$.

The following lemma shows that $(5\log)^*_{1+\epsilon}(y)$ is well-defined.

Lemma 2.1. $5 \cdot \log_{1+\epsilon} y \leq \sqrt{\frac{\epsilon}{5}} \cdot y$ *for any* $y \geq \frac{200}{\epsilon^2}$.

Proof. First, notice that $\ln(1+\epsilon) \geq \frac{1}{2}\epsilon$ for $\epsilon \in (0,1)$, so $5 \cdot \log_{1+\epsilon} y \leq \frac{10}{\epsilon}\ln y$. Thus, it suffices to show that $\ln y \leq \frac{\epsilon\sqrt{\epsilon}}{10\sqrt{5}} y$. Let $y = \frac{200}{\epsilon^2}(1+x)$ for some $x \geq 0$. Then

$$\ln y = \ln \frac{200}{\epsilon^2}(1+x) = 4\ln\frac{\sqrt{5}(1+x)^{1/4}}{\sqrt{\epsilon}} + \ln 8 \leq \frac{4\sqrt{5}}{\sqrt{\epsilon}}(1+x)^{1/4} \leq \frac{4\sqrt{5}}{\sqrt{\epsilon}}(1+x) = \frac{\epsilon\sqrt{\epsilon}}{10\sqrt{5}} y,$$

where the first inequality is due to $\ln z \leq z - 1$ for any $z > 0$ and $\ln 8 \leq 4$. □

Lemma 2.1 implies that $(5\log)_{1+\epsilon}(y) = O(\log^* y)$, since applying $(5\log)_{1+\epsilon}$ three times either results in something bounded by $\frac{200}{\epsilon^2}$, or decreases the argument exponentially, i.e. $(5\log)^{(3)}_{1+\epsilon}(y) \leq \epsilon/5 \cdot 5\log_{1+\epsilon} y \leq 2\ln y$.

2.1 Primal-dual framework

We will always assume that $f > \frac{\log C}{\epsilon}$, since otherwise we will simply apply the algorithm from [BHNW21]. For each element $e \in \mathcal{U}$, we assume all the sets s containing e are stored as an array, not a linked list, so that we can take uniformly random samples from all these sets in $O(1)$ time. This assumption is valid because only the elements are dynamic, while all sets are static.

We will follow the primal-dual framework from [BHNW21, BHN19, BK19]. However, there is a tiny difference: instead of aiming to satisfy $\omega(s) \leq c_s, \forall s \in \mathcal{S}$, we aim to satisfy $\omega(s) < c_s, \forall s \in \mathcal{S}$. This is not crucial for the approximation guarantee, but this simplifies the algorithm and the analysis.

Let $\epsilon \in (0, 0.1)$ be a constant. Define $L = \lceil \log_{1+\epsilon}(Cn) \rceil + 1$. Each set $s \in S$ is assigned a level $\mathsf{lev}(s) \in [L]$. The base level of a set is defined as $\mathsf{base}(s) = \lfloor \log_{1+\epsilon} 1/c_s \rfloor$.

Each element e will be assigned a level $\mathsf{lev}(e) = \max_{s \ni e}\{\mathsf{lev}(s)\}$ and weight $\omega(e)$, and $\omega(s) = \sum_{e \in s} \omega(e)$ denotes the total weight of $s \in \mathcal{S}$. In addition, for every set $s \in \mathcal{S}$, we also maintain a *dead weight* $\phi(s)$, and let $\omega^*(s) = \omega(s) + \phi(s)$ be the *composite weight*.

Definition 2.2. *A set s is called* tight, *if $\omega^*(s) \geq \frac{c_s}{1+\epsilon}$, and* slack *otherwise.*

2.2 Basic data structures

During the dynamic algorithm, we will not keep track of the value of $\mathsf{lev}(e)$ for all elements. Instead, we will maintain a *lazy level* $\mathsf{zlev}(e)$. In addition, we will maintain an *intrinsic level* $\mathsf{ilev}(e)$, which defines the weight of an element: $\omega(e) = (1+\epsilon)^{-\mathsf{ilev}(e)}$. Because of that, all elements have two categories: *active* and *passive*.

- **Active.** If an element e is *active*, then the value of $\mathsf{lev}(e)$ will be correctly maintained. For such elements we will have $\mathsf{zlev}(e) = \mathsf{ilev}(e) = \mathsf{lev}(e)$, and so $\omega(e) = (1+\epsilon)^{-\mathsf{lev}(e)}$. Let $A_i \subseteq \mathcal{U}$ be the set of active elements on level i. For each set s and each level index i, our algorithm explicitly maintains a list $A_i(s) \subseteq A_i$ which is the set of active elements in s on level i.

- **Passive.** If an element e is *passive*, due to runtime issues, we might not always keep track of the value $\mathsf{lev}(e)$ all the time. Instead, we can only maintain a *lazy* level $\mathsf{zlev}(e) \leq \mathsf{lev}(e)$ which is refreshed to $\mathsf{lev}(e)$ once in a while. The *intrinsic* level $\mathsf{ilev}(e)$ will satisfy $\mathsf{lev}(e) < \mathsf{ilev}(e) \leq \mathsf{zlev}(e) + \lceil \log_{1+\epsilon} \max\{f, \frac{2C}{\epsilon}\} \rceil$.

  Let $P_i \subseteq \mathcal{U}$ be the set of all passive elements whose intrinsic levels are i. For each set s and each intrinsic level i, our algorithm explicitly maintains a list $P_i(s) \subseteq P_i$ which is the set of passive elements in s on intrinsic level i. In contrast, we will not maintain a list for the set of passive elements in set s on lazy level i (since that would be too time-consuming), hence we are unable to enumerate all the passive elements in s on lazy level i.

For a set s and an index $i \geq \mathsf{lev}(s)$, the weight of s at level i is defined as:

$$\begin{aligned}\omega(s,i) &= \sum_{\text{active } e \in s} (1+\epsilon)^{-\max\{i, \max_{t \mid e \in t \neq s} \mathsf{lev}(t)\}} + \sum_{\text{passive } e \in S} (1+\epsilon)^{-\max\{i, \mathsf{ilev}(e)\}} \\ &= \sum_{e \in s} \min\left\{\omega(e), (1+\epsilon)^{-\max\{i, \max_{t \mid e \in t \neq s} \mathsf{lev}(t)\}}\right\}\end{aligned} \quad (1)$$

In other words, $\omega(s,i)$ is the weight of s if it were raised to level i. So by definition, $\omega(s) = \omega(s, \mathsf{lev}(s))$, and

$$\omega(s, \mathsf{lev}(s)+1) = \omega(s) - |A_{\mathsf{lev}(s)}(s)| \cdot \epsilon(1+\epsilon)^{-\mathsf{lev}(s)-1}, \quad (2)$$

which can be computed in $O(1)$ time once we know $\omega(s)$ and $|A_{\mathsf{lev}(s)}(s)|$. We assume that all powers of $1+\epsilon$ can be computed in constant time; one way of implementing this efficiently is to compute all these powers at the outset in $O(L)$ time.

Throughout the algorithm, let $T \subseteq \mathcal{S}$ be the set of all tight sets, and let $\phi = \sum_{s\in\mathcal{S}} \phi(s)$ be the total dead weight. For each i, let $S_i \subseteq \mathcal{S}$ be the set of sets at level i, let $T_i \subseteq T$ be the set of tight sets at level i, let E_i be the set of elements such that $\mathsf{zlev}(e) = i$ (note that E_i contains all active elements such that $\mathsf{lev}(e) = i$, since for them $\mathsf{zlev}(e) = \mathsf{lev}(e)$). Define $\phi_i = \sum_{s\in S_i} \phi(s)$, the total dead weight of sets on level i; similarly, we can define notations $\phi_{\leq i}, S_{\leq i}, T_{\leq i}, E_{\leq i}$.

Each set E_i, S_i, T_i will be maintained as a linked list, and we store all pointers to lists $\{S_i\}_{0\leq i\leq L}$, $\{T_i\}_{0\leq i\leq L}$, $\{E_i\}_{0\leq i\leq L}$ as three arrays of length $L+1$. When the values of $\mathsf{lev}(s), \mathsf{zlev}(e), \omega(s), \omega(e)$ change for a set s or an element e, we can update the lists and the values $\phi_i, \omega(E_i), \omega(S_i), \omega(T_i)$ accordingly in constant time.

Iterating over nonempty sets. In the algorithm, we want to be able to access nonempty sets from $\{E_i\}_{0\leq i\leq L}$, $\{S_i\}_{0\leq i\leq L}$, $\{T_i\}_{0\leq i\leq L}$ efficiently in the increasing order by i. If we were maintaining them in doubly linked lists, we would not be able to update the lists in constant time whenever we update the values of $\mathsf{zlev}(e), \mathsf{lev}(s)$. To cope with that, we rely on the way our algorithm update these values. First, notice that we can update the lists in constant time if we increase these values by one. Another idea is to maintain the doubly linked lists only for levels above $\lceil \log_{1+\epsilon} C \rceil + 1$, so we can update them in constant time whenever we set $\mathsf{lev}(s)$ to some $k \leq \lceil \log_{1+\epsilon} C \rceil + 1$. We use the same idea for $\{E_i\}_{0\leq i\leq L}$, but we also store E_i whenever $T_i \neq \emptyset$, even if $E_i = \emptyset$. That way, we are able to update the list when we set $\mathsf{zlev}(e)$ to $\max_{s\ni e}\{\mathsf{lev}(s)\}$. This allows us to compute quantities $\phi_{\leq i}$, $c(T_{\leq i})$ and $\omega(E_{\leq i})$ more efficiently.

Observation 2.1. *This linked list data structure allows us to compute quantities $\phi_{\leq i}, c(T_{\leq i})$ in $O\left(\left|T_{\leq i} \setminus T_{\leq \lceil \log_{1+\epsilon} C\rceil + 1}\right| + \frac{\log C}{\epsilon}\right)$ time, and enumerate elements from $E_{\leq i}$ or compute $\omega(E_{\leq i})$ in $O\left(\left|T_{\leq i} \setminus T_{\leq \lceil \log_{1+\epsilon} C\rceil + 1}\right| + \frac{\log C}{\epsilon} + |E_{\leq i}|\right)$ time.*

Implicit zeroing. We need a fast data structure for the following operation.

- Given a level index $0 \leq i \leq L$, we want to assign $\mathsf{lev}(s), \phi(s) \leftarrow 0$ for all $s \in S_i$, and we need to do this in constant time.

Updating the lists T_i, S_i or sums $\phi_i, \phi_0, c(T_i), c(S_i)$ can be done in constant time. However, this task is impossible if we want to explicitly update all the values $\phi(s), \mathsf{lev}(s) \leftarrow 0$ for all $s \in S_i$. So, we have to zero out each individual value $\mathsf{lev}(s), \phi(s)$ in an implicit way. To do this, for each set $s \in \mathcal{S}$, we will associate it with a time stamp $\mathsf{tm}(s)$ which indicates the latest time point when the value of $\mathsf{lev}(s)$ or $\phi(s)$ is explicitly updated. Then, create an array aux of length $L+1$, where each entry $\mathsf{aux}[i]$ stores the time point t of the latest zeroing operation to level i. Each time we want to access the values of $\mathsf{lev}(s), \phi(s)$, compare $\mathsf{tm}(s)$ and $\mathsf{aux}[\mathsf{lev}(s)]$. If $\mathsf{tm}(s) > \mathsf{aux}[\mathsf{lev}(s)]$, we know that s did not suffer the latest zeroing out on level $\mathsf{lev}(s)$, and hence $\mathsf{lev}(s), \phi(s)$ are referring to their current values; otherwise, s must have undergone a zeroing operation implicitly. In this case, explicitly set $\mathsf{lev}(s), \phi(s)$ to 0, and update $\mathsf{tm}(s)$ accordingly. Here we have implicitly assumed that the time values can be stored in a single word; otherwise, we would rebuild the entire dynamic set cover data structure and reset the time to zero.

Zeroing out the levels of $\mathsf{lev}(s)$ may also affect the levels of other elements. But in our algorithm, we will apply implicit zeroing in a careful manner, so that the levels of sets and elements are consistent.

2.3 Approximation guarantees

Invariant 2.1. *During the algorithm, we will maintain the following invariants.*

(1) For any set s, $\omega(s, \mathsf{lev}(s)+1) < c_s$.

(2) All sets at level at least 1 are tight.

(3) It always holds that $\phi \le \epsilon\,(c(T) + f \cdot \omega(\mathcal{U}))$.

Corollary 2.1. *If Invariant 2.1(1) holds, then $\omega(s) < (1+\epsilon)c_s$ for all sets $s \in \mathcal{S}$.*

Proof. Notice that $\omega(s) \ge |A_{\mathsf{lev}(s)}(s)| \cdot (1+\epsilon)^{-\mathsf{lev}(s)}$, since every $e \in A_{\mathsf{lev}(s)}(s)$ has $\omega(e) = (1+\epsilon)^{-\mathsf{lev}(s)}$. Thus,

$$\omega(s) - |A_{\mathsf{lev}(s)}(s)| \cdot \epsilon(1+\epsilon)^{-\mathsf{lev}(s)-1} \ge \omega(s) - \frac{\epsilon}{1+\epsilon}\omega(s) = \frac{\omega(s)}{1+\epsilon}.$$

Plugging this into Equation (1), we get $\omega(s) \le (1+\epsilon)\cdot\omega(s, \mathsf{lev}(s)+1) < (1+\epsilon)c_s$. □

Lemma 2.2 ([BHNW21])**.** *If Invariant 2.1 holds, then $\omega(s) < (1+\epsilon)c_s$, and $\omega(\mathcal{U}) \le (1+\epsilon)\cdot\mathsf{OPT}$, where $\mathsf{OPT} = c(\mathcal{S}^*)$ is the total weight of an optimal set cover $\mathcal{S}^*$.*

Proof. By Corollary 2.1, $\omega(s) < (1+\epsilon)c_s$. Furthermore, we have

$$\omega(\mathcal{U}) = \sum_{e\in\mathcal{U}} \omega(e) \le \sum_{s\in\mathcal{S}^*}\sum_{e\in s}\omega(e) \le (1+\epsilon)\cdot c(\mathcal{S}^*) = (1+\epsilon)\cdot\mathsf{OPT}$$

This first inequality relies on the fact that $\mathcal{S}^*$ is a valid set cover. □

Lemma 2.3 ([BHNW21])**.** *If Invariant 2.1 holds and the collection of tight sets T is a set cover, then T is a $(1+5\epsilon)f$-approximate set cover.*

Proof. By the definition of a tight set, we have $\omega^*(s) = \omega(s) + \phi(s) \ge \frac{c_s}{1+\epsilon}$. Then, the cost of T is bounded by

$$\begin{aligned} c(T) \le (1+\epsilon)\cdot\sum_{s\in T}(\omega(s)+\phi(s)) &\le (1+\epsilon)\cdot\omega(\mathcal{S}) + (1+\epsilon)\cdot\phi \\ &\le (1+\epsilon)f\cdot\omega(\mathcal{U}) + \epsilon(1+\epsilon)\cdot c(T) + \epsilon(1+\epsilon)f\cdot\omega(\mathcal{U}) \\ &\le (1+\epsilon)^2 f\cdot\omega(\mathcal{U}) + \epsilon(1+\epsilon)\cdot c(T) \end{aligned}$$

As $\epsilon \in (0, 0.1)$ and by Lemma 2.2, we have

$$c(T) \le \frac{(1+\epsilon)^2 f}{1-\epsilon(1+\epsilon)}\cdot\omega(\mathcal{U}) \le (1+5\epsilon)f\cdot\mathsf{OPT}$$

□

2.4 Glossary

Some of the notations used are summarized in Table 2 (placed in the last page for convenience).

3 Algorithm description

We will describe subroutines $\mathsf{Delete}(e)$, $\mathsf{Insert}(e)$, $\mathsf{FixLevel}(e, l)$, and $\mathsf{Rebuild}(k)$ which constitute the main update algorithm, whose pseudocode is given in Algorithm 1. At the beginning of the algorithm, we assume $\mathcal{U}$ is empty, and so all sets in $\mathcal{S}$ are initialized on level 0. When an element e is deleted from $\mathcal{U}$, we will call subroutine $\mathsf{Delete}(e)$ to deal with it; if an element e is inserted, then we will call $\mathsf{Insert}(e)$.

After that, we check if Invariant 2.1(3) is violated. If so, we find the smallest index k such that $\phi_{\leq k} > \epsilon \cdot (c(T_{\leq k}) + f \cdot \omega(E_{\leq k}))$ and then invoke subroutine $\mathsf{Rebuild}(k)$; this is repeated until Invariant 2.1(3) holds.

Algorithm 1: DynamicSetCover

```
1 initialize $\mathsf{lev}(s) = 0, \forall s \in \mathcal{S}$;
2 foreach element update $e$ do
3     if $e$ is deleted then
4         Delete($e$);
5     else
6         Insert($e$);
7     while Invariant 2.1(3) is violated do
8         find the smallest $k$ such that $\phi_{\leq k} > \epsilon\,(c(T_{\leq k}) + f \cdot \omega(E_{\leq k}))$;
9         Rebuild($k$);
```

To implement line 8 which finds the smallest index k such that $\phi_{\leq k} > \epsilon \cdot (c(T_{\leq k}) + f \cdot \omega(E_{\leq k}))$, start with $k = 0$ and each time increase k to the next index $k \leftarrow k'$ where $T_{k'} \neq \emptyset$ or $E_{k'} \neq \emptyset$, using the doubly linked list data structure, and check if $\phi_{\leq k} > \epsilon \cdot (c(T_{\leq k}) + f \cdot \omega(E_{\leq k}))$. In this way, the runtime of locating the smallest k would be $O\left(|T_{\leq k} \setminus T_{\leq \lceil \log_{1+\epsilon} C \rceil + 1}| + \frac{\log C}{\epsilon} + |E_{\leq k}|\right)$ (similarly to Observation 2.1); note that the amount of time spent per level is constant, since we have maintained the quantities per each level separately, and we just need to sum the quantities for prefixes of levels.

We will make sure that each of the Insert, Delete and $\mathsf{Rebuild}$ subroutines does not increase the gap $\mathsf{ilev}(e) - \mathsf{zlev}(e)$ for any passive element e. That property will allow us to bound the total time spent on element e.

3.1 Deletion

We handle deletions in the same way as [BHNW21]; refer to Algorithm 2 for the pseudocode. When an element e is deleted, the algorithm subtracts, for each set $s \ni e$, the value of $\omega(e)$ from its weight $\omega(s)$, and compensates for the loss by increasing the dead weight $\phi(s)$ by $\omega(e)$, if s was tight. This $\mathsf{Delete}(e)$ subroutine takes $O(f)$ time.

Besides, we also need to specify how to maintain the underlying data structures after an element deletion. If e is active, then we go over all sets $s \ni e$ and remove e from the linked list $A_{\mathsf{ilev}(e)}(s)$; if e is passive, then we go over all sets $s \ni e$ and remove e from the linked list $P_{\mathsf{ilev}(e)}(s)$. This operation takes time $O(f)$.

As for the invariants, since $\mathsf{Delete}(e)$ does not increase any weight $\omega(s)$, Invariant 2.1(1) is preserved. Invariant 2.1(2) is also preserved due to the way we modify the dead weights. Invariant 2.1(3) might have been violated due to the increases of dead weights, but it will be restored by the while loop on line 7 of Algorithm 1.

Algorithm 2: Delete(e)

```
1 foreach set $s \ni e$ do
2     $\omega(s) \leftarrow \omega(s) - \omega(e)$;
3     if $s$ was tight then
4         $\phi(s) \leftarrow \phi(s) + \omega(e)$;
```

3.2 Insertion

High-level idea. When inserting an element e, we aim to satisfy the constraint $\forall s \ni e$, $\omega(s) < c_s$. We try to make the newly inserted element e active at level $\mathsf{lev}(e) = \max_{s \ni e}\{\mathsf{lev}(s)\}$ if possible. If not, to make sure e is covered by a tight set, we try to make it passive at the lowest possible intrinsic level up to $l = \max_{s \ni e}\{\mathsf{lev}(s)\} + \lceil \log_{1+\epsilon} \max\{f, \frac{2C}{\epsilon}\}\rceil$, since we want $\mathsf{ilev}(e) \le \mathsf{zlev}(e) + \lceil \log_{1+\epsilon} \max\{f, \frac{2C}{\epsilon}\}\rceil$. This may still be impossible; in that case, we invoke $\mathsf{FixLevel}(e, l)$.

Thus, we invoke $\mathsf{FixLevel}(e, l)$ when we can't make the element passive on intrinsic level $l \le \max_{s \ni e}\{\mathsf{lev}(s)\} + \lceil \log_{1+\epsilon} f\rceil$. Our amortized analysis employs a potential function. As we will show later (the full analysis is given in Section 4.1, this is just for intuition), the *potential increase* and thus the amortized cost of $\mathsf{FixLevel}(e, l)$ is bounded by roughly $f^2 \cdot (1+\epsilon)^{-d}$, where $d = \mathsf{ilev}(e) - \mathsf{lev}(e)$. Hence by placing the element $\lceil \log_{1+\epsilon} f\rceil$ levels higher than $\mathsf{lev}(e)$, we can guarantee that the *potential increase* is roughly f. We note that we call to $\mathsf{FixLevel}(e, l)$ only when the gap $\mathsf{ilev}(e) - \mathsf{lev}(e)$ is at least $\log_{1+\epsilon} \frac{2C}{\epsilon}$ (hence the reason for taking the maximum of f and $\frac{2C}{\epsilon}$); that restriction, as we will show later, guarantees that the invariants are preserved.

3.2.1 Description of the Insert subroutine

Upon an insertion e, assign $\mathsf{zlev}(e) \leftarrow \max_{s \ni e}\{\mathsf{lev}(s)\}$ (thus making $\mathsf{zlev}(e) = \mathsf{lev}(e)$). Define $l = \max_{s \ni e}\{\mathsf{lev}(s)\} + \lceil \log_{1+\epsilon} \max\{f, \frac{2C}{\epsilon}\}\rceil$ and $F = \{s \ni e \mid \omega(s) + (1+\epsilon)^{-l} \ge c_s\}$. Next, we branch into two cases:

(1) If $F = \emptyset$, then it is possible to insert e at an intrinsic level $\mathsf{ilev}(e) \le l$ without violating $\omega(s) < c_s$ for any $s \ni e$. In that case, we compute the smallest index $h \ge \mathsf{zlev}(e)$ such that $\forall s \ni e$, $\omega(s) + (1+\epsilon)^{-h} < c_s$ and set $\mathsf{ilev}(e) \leftarrow h$. After that, we update the weights of sets by going over each set $s \in e$ and setting $\omega(s) \leftarrow \omega(s) + \omega(e)$. We also add e to $A_h(s)$ if e is active (i.e., $\mathsf{ilev}(e) = \mathsf{zlev}(e)$), or $P_h(s)$ if e is passive, for each $s \ni e$ (which is omitted in the pseudocode).

    Computing h in $O(f)$ time. To compute h in $O(f)$ time, we can first compute the minimum value of the gap $c_s - \omega(s)$, and then use binary search over the interval $[\mathsf{zlev}(e), l]$ to find h, which takes time $O\left(\log\left(\log_{1+\epsilon} \max\{f, \frac{2C}{\epsilon}\}\right)\right)$. Note that $\log_{1+\epsilon} \frac{2C}{\epsilon} = O\left(\frac{\log C}{\epsilon} + \frac{1}{\epsilon^2}\right)$, since $\log_{1+\epsilon} x = O\left(\frac{\log x}{\epsilon}\right)$ for any $x > 0, \epsilon \in (0,1)$. Recall that we assume $f > \frac{\log C}{\epsilon}$. Then the time to find h is $O\left(\log\left(\frac{\log f}{\epsilon} + \frac{\log C}{\epsilon} + \frac{1}{\epsilon^2}\right)\right) = O(f)$, where the last transition holds due to $\frac{1}{\epsilon} \le \frac{\log C}{\epsilon} < f$. This operation will appear again in the Rebuild subroutine.

(2) If $F \ne \emptyset$, then it is impossible to insert e at an intrinsic level $\mathsf{ilev}(e) \le l$ without violating $\omega(s) < c_s$ for some $s \ni e$. Inserting e at intrinsic level $\mathsf{ilev}(e) = l$ may violate Invariant 2.1(1). Hence we apply subroutine $\mathsf{FixLevel}(e, l)$, which will make e passive at intrinsic level l or higher, but will keep the gap $\mathsf{ilev}(e) - \mathsf{zlev}(e)$ (and the gap $\mathsf{ilev}(e) - \mathsf{lev}(e)$ as well, since it also keeps $\mathsf{zlev}(e) = \mathsf{lev}(e)$) equal to $\lceil \log_{1+\epsilon} \max\{f, \frac{2C}{\epsilon}\}\rceil$, and make sure that all the invariants are satisfied.

Algorithm 3: Insert(e)

```
1  assign zlev($e$) ← $\max_{s \ni e}\{\mathsf{lev}(s)\}$;
2  let $l = \max_{s \ni e}\{\mathsf{lev}(s)\} + \lceil \log_{1+\epsilon} \max\{f, \frac{2C}{\epsilon}\} \rceil$;
3  let $F = \{s \ni e \mid \omega(s) + (1+\epsilon)^{-l} \geq c_s\}$;
4  if $F = \emptyset$ then
5      compute the smallest index $h \geq \mathsf{zlev}(e)$ such that $\omega(s) + (1+\epsilon)^{-h} < c_s, \forall s \ni e$;
6      ilev($e$) ← $h$;
7      foreach $s \ni e$ do
8          $\omega(s) \leftarrow \omega(s) + \omega(e)$;
9  else
10     FixLevel($e, l$);
```

It is left to show that Insert maintains a valid set cover and preserves the invariants. We defer the proof of the following theorem to Section 3.3.2, since our argument relies on the properties of the FixLevel subroutine, which we have not stated yet.

Theorem 3.1. *After the call to* Insert(e)*, T is a set cover and Invariant 2.1(1)(2) are maintained.*

3.3 Fixing levels

High-level idea. When the subroutine FixLevel(e, l) is called, we assign the intrinsic level of a passive or new element e to l. However, this update can increase $\omega(s)$ for $s \ni e$, and hence could violate Invariant 2.1(1). To restore it, the subroutine then goes over each set $s \ni e$ and repeatedly raises s; that is, increases $\mathsf{lev}(s)$ and increases the levels of elements in s if needed, until the invariant is satisfied. Additionally, it makes sure that the gap $d = \mathsf{ilev}(e) - \mathsf{lev}(e)$ remains the same, by increasing $\mathsf{ilev}(e)$ whenever $\mathsf{lev}(e)$ increases. The fact that the gap does not decrease is crucial for the amortized runtime analysis.

The exact definition of the potential functions used in the amortized runtime analysis is provided later in Section 4.1. Intuitively, each unit of "excess weight", that is $\max\{\omega(s) - c_s, 0\}$, has a potential cost, which depends $\mathsf{lev}(s)$. Increasing $\mathsf{lev}(s)$ by one increases the potential cost of one unit of weight by a factor of $1 + \epsilon$. If we had kept $\mathsf{ilev}(e)$ unchanged, $\mathsf{lev}(s)$ could become close to $\mathsf{ilev}(e)$, which would make the potential increase too large. So our rule here is to keep the gap $d = \mathsf{ilev}(e) - \mathsf{lev}(e)$ the same by increasing $\mathsf{ilev}(e)$ whenever $\mathsf{lev}(e)$ increases.

Let F be the set of set $s \ni e$, for which $\omega(s) \geq c_s$ after making the intrinsic level of e to be l. Note that only sets from F need to be raised to restore Invariant 2.1(1). As we will show later in Section 4, the amortized runtime of FixLevel depends on the size of F and the size of the gap $d = \mathsf{ilev}(e) - \mathsf{zlev}(e)$, and is roughly $|F| \cdot f \cdot (1+\epsilon)^{-d}$. Therefore, the smaller F is, the smaller we can make the gap to achieve the same runtime. Later we will use this property to improve the runtime of the Rebuild subroutine.

Whenever we raise s one level up, levels of some elements in s may also increase. Since we maintain the gap $\mathsf{ilev}(e) - \mathsf{lev}(e)$, we may also need to increase $\mathsf{ilev}(e)$, which decreases $\omega(e)$ by a factor of $1 + \epsilon$. The tightness is maintained for s, since we raise s only when $\omega(s, \mathsf{lev}(s) + 1) \geq c_s$, so $\omega(s) \geq c_s/(1 + \epsilon)$ as a result. However, this could also decrease $\omega(s')$ for some other sets $s' \neq s$, and hence s' may become slack. To avoid that, for each element $e' \neq e$ that decreased its weight due to the raise of s, we compensate the loss of $\omega(s')$ incurred by e' by increasing the dead weight $\phi(s')$. As for the losses due to increases of $\mathsf{ilev}(e)$, we need to address them more carefully. Before describing how to implement this approach, there are some technical challenges we would like to explain.

Technical challenges. If we enumerate all sets $s \ni e$ in an arbitrary order, and increase $\mathsf{ilev}(e)$ in each round, then the tightness of some previously visited sets $s' \ni e$ might be violated. To avoid this issue, in the original algorithm of [BHNW21], they enumerated the sets $s \ni e$ by increasing order of difference $c_s - \omega(s)$, which already takes $O(f \log f)$ time; we note that one cannot use a linear-time approximate sorting algorithm for this task. For our deterministic algorithm we can afford to spend the sorting time of $O(f \log f)$, but the update time of our randomized algorithm is asymptotically smaller than that; to circumvent the $\Omega(f \log f)$ sorting time overhead, we will make do with an arbitrary ordering of these sets, and restore tightness of sets $s' \ni e$ by increasing their dead weights.

Each time the intrinsic level $\mathsf{ilev}(e)$ of e changes, we may need to update the weights of sets $s' \ni e$, which already takes $O(f)$ time. However, in some cases, the decrease in potential would not be enough to cover that runtime cost. So in the worst case, the runtime cost could be as high as $O(f^2)$. To avoid this, we will update the contribution of $\omega(e)$ to the weights $\omega(s')$ of other sets $s' \ni e$ in a lazy manner. More specifically, we will do so only when a set $s' \ni e$ is being enumerated — which is when we actually need the updated value $\omega(s')$.

The decrease of $\omega(e)$ might also violate the tightness of some yet unvisited sets $s \ni e$. However, compensating these losses by increasing $\phi(s)$ might be too costly in terms of the potential increase. In practice, the algorithm will make sure that $\mathsf{ilev}(e)$ is not higher than it was before the call (if e existed before the call). We will prove later that this allows us to maintain tightness for all sets.

3.3.1 Description of the FixLevel subroutine

Next, let us describe the algorithm more formally; see Algorithm 4 for the pseudocode of the FixLevel subroutine. Let $\mathsf{ilev}^{\mathrm{old}}(e)$ and $\omega^{\mathrm{old}}(e)$ be the intrinsic level and the weight of element e right before the execution of $\mathsf{FixLevel}(e, l)$; if e is a newly inserted element, then define $\mathsf{ilev}^{\mathrm{old}}(e) = \infty$ (and hence $\omega^{\mathrm{old}}(e) = 0$).

The $\mathsf{FixLevel}(e, l)$ subroutine assumes that $\mathsf{zlev}(e) = \mathsf{lev}(e)$ at the beginning of the call and $\mathsf{lev}^{\mathrm{old}}(e) < l \leq \mathsf{ilev}^{\mathrm{old}}(e)$. The algorithm aims to maintain Invariant 2.1(1). We also note that the algorithm does not explicitly maintain Invariant 2.1(2). The reason for that is that the subroutine is called from the Rebuild subroutine, where Invariant 2.1(2) may be (temporarily) violated. However, the algorithm makes sure that tight sets remain tight, and we will show later that if some conditions hold, no slack set get raised. For the same reasons, we allow passive elements to violate $\mathsf{zlev}(e') \leq \mathsf{lev}(e')$. However, for active elements we still have $\mathsf{ilev}(e') = \mathsf{zlev}(e') = \mathsf{lev}(e')$, and for passive elements we still have $\mathsf{lev}(e') < \mathsf{ilev}(e') \leq \mathsf{zlev}(e') + \lceil \log_{1+\epsilon} \max\{f, \frac{2C}{\epsilon}\} \rceil$.

The algorithm makes sure that $\mathsf{ilev}(e) \leq \mathsf{ilev}^{\mathrm{old}}(e)$ during its execution; we will rely on this property in the proof of correctness. Intuitively, every time we raise a set during FixLevel, some active elements can also increase their level, and hence lose weight. We will compensate each such loss by the increase of dead weights. However, we will not compensate the loss due to increases of $\mathsf{ilev}(e)$, but the inequality $\mathsf{ilev}(e) \leq \mathsf{ilev}^{\mathrm{old}}(e)$ will guarantee that $\omega(e)$ is no smaller than it was at the beginning of the call, so the tightness will be preserved.

Next, let us describe the steps the algorithm makes.

Changing the intrinsic level of e to l. First, make e passive at intrinsic level l by setting $\mathsf{ilev}(e) \leftarrow l$ and then setting $\omega(s) \leftarrow \omega(s) - \omega^{\mathrm{old}}(e) + \omega(e)$ for each set $s \ni e$. If e is not a freshly inserted element, then remove e from $P_l(s)$ for each $s \ni e$; we will add it to the relevant sets in the end, when we finalize the changes, where we have calculated the final value of $\mathsf{ilev}(e)$. Keep a record $d = l - \mathsf{zlev}(e)$ and compute $F = \{s \ni e \mid \omega(s) \geq c_s\}$.

During the algorithm, we will gradually increase $\mathsf{ilev}(e)$ (but never above $\mathsf{ilev}^{\text{old}}(e)$). However, updating all weights and the relevant data structures takes $O(f)$ time. To avoid spending $O(f)$ time for such updates each time $\mathsf{ilev}(e)$ increases, we will update the data structures only once, during the finalization step.

Raising sets $s \ni e$. During the algorithm, we will make sure that the lazy level $\mathsf{zlev}(e)$ is always equal to the actual level $\mathsf{lev}(e)$, and the gap $\mathsf{ilev}(e) - \mathsf{zlev}(e)$ is always equal to d. The algorithm then goes over each set $s \ni e$ and processes it the following way: First, it updates the weight of s according to the current value of $\omega(e)$, since throughout the algorithm execution we may have changed $\omega(e)$ due to increases of $\mathsf{ilev}(e)$, from $(1+\epsilon)^{-l}$ to some possibly lower weight. To update $\omega(s)$ accordingly, we set $\omega(s) \leftarrow \omega(s) - (1+\epsilon)^{-l} + \omega(e)$.

Next, we repair Invariant 2.1(1) for s by increasing $\mathsf{lev}(s)$. Recall that $\omega(s, \mathsf{lev}(s)+1)$ can be computed in constant time, given $\omega(s)$ and $|A_{\mathsf{lev}(s)}(s)|$, by Equation (2).

Raising s to level $\min\{\mathsf{base}(s), \mathsf{zlev}(e)\}$. If $\mathsf{lev}(s)$ is below $\min\{\mathsf{base}(s), \mathsf{zlev}(e)\}$ and we still have $\omega(s, \mathsf{lev}(s)+1) \geq c_s$, then we leverage the fact that there are no elements at intrinsic levels below $\mathsf{base}(s)$ (the proof of that is given later in Claim 3.2). Because of that, we can raise s to $\min\{\mathsf{base}(s), \mathsf{zlev}(e)\}$ in a single shot. Since $\omega(s, \mathsf{lev}(s)+1) \geq c_s$, we have $\omega(s) \geq c_s$, so s is tight regardless of the value of $\phi(s)$. Thus, we zero out $\phi(s)$. Then, we set $\mathsf{lev}(s) \leftarrow \min\{\mathsf{base}(s), \mathsf{lev}(s)\}$. Now there can be passive elements $e' \in s$ such that $\mathsf{ilev}(e') = \mathsf{lev}(s)$, so we need to make them active (note that there are no elements with $\mathsf{ilev}(e') < \mathsf{lev}(s)$, since $\mathsf{ilev}(e')$ must be at least $\mathsf{base}(s)$). To activate elements in $P_{\mathsf{lev}(s)}(s)$, go over each $e' \in P_{\mathsf{lev}(s)}(s)$ and set $\mathsf{zlev}(e') \leftarrow \mathsf{lev}(s)$, and for every set $s' \ni e'$, move e' from $P_{\mathsf{lev}(s)}(s')$ to $A_{\mathsf{lev}(s)}(s')$.

Raising s by one level. At this point, Invariant 2.1(1) may be still violated for s. Thus, we repeatedly increase the level of s by one in a while loop, until $\omega(s, \mathsf{lev}(s)+1) < c_s$ is satisfied. During each iteration, the algorithm does the following steps:

(1) Set $\phi(s) \leftarrow 0$; again, we have $\omega(s) \geq c_s$, so we can safely zero out $\phi(s)$ without breaking the tightness. Define $k = \mathsf{lev}(s)$ and increase $\mathsf{lev}(s)$ by one.

(2) **Raising elements from $A_k(s)$:** The increase of $\mathsf{lev}(s)$ leads to the increase of levels of elements from $A_k(s)$. Thus, we go over all elements $e' \in A_k(s)$ and raise them to level $k+1$. To do so, set $\mathsf{ilev}(e'), \mathsf{zlev}(e') \leftarrow k+1$ (making $\omega(e') = (1+\epsilon)^{-k-1}$), and for each $s' \ni e'$, move e' from $A_k(s')$ to $A_{k+1}(s')$ and update $\omega(s')$ by decreasing it by $\epsilon(1+\epsilon)^{-k-1}$.

    After that, for each $s' \ni e', s' \neq s$, to restore tightness on set $s' \ni e'$, we increase its dead weight $\phi(s')$ by $\epsilon(1+\epsilon)^{-k-1}$.

(3) **Raising e:** If $\mathsf{zlev}(e) = k$, then $\mathsf{lev}(e)$ becomes $k+1$ due to the increase of $\mathsf{lev}(s)$, so we need to update the levels of e. To do it, increase both $\mathsf{zlev}(e)$ and $\mathsf{ilev}(e)$ by one and update $\omega(s)$ accordingly, by setting $\omega(s) \leftarrow \omega(s) - \epsilon(1+\epsilon)^{-k-1-d}$.

(4) **Activating passive elements:** Since we have increased $\mathsf{lev}(s)$ by one, for elements $e' \in P_{\mathsf{lev}(s)}(s)$ we now have $\mathsf{ilev}(e) = \mathsf{lev}(e)$, so we need to activate them. We do it the same way we did when raising s to level $\min\{\mathsf{base}(s), \mathsf{zlev}(e)\}$.

After the while loop terminates, the algorithm keeps a record of the current value of $l_s \leftarrow \mathsf{ilev}(e)$ for the current set s to update $\omega(s)$ to the actual value in the end.

Finalizing the changes. When all sets in $s \ni e$ have been enumerated, since we are being lazy on updating the contribution of e whenever $\mathsf{ilev}(e)$ changes, we need to update the relevant data structures accordingly. To do so, go over all sets $s \ni e$ again to update the weight by setting $\omega(s) \leftarrow \omega(s) - (1+\epsilon)^{-l_s} + \omega(e)$. This update of $\omega(s)$ could decrease it, so s might have become slack. Hence, to restore the tightness for s, we increase its dead weight $\phi(s)$ by $(1+\epsilon)^{-l_s} - \omega(e)$ if $s \in F$. We will show later that if $s \notin F$, then we did not enter the while loop for s, and so every loss of $\omega(s)$ due to raises of active elements was compensated by the increase of $\phi(s)$, and we never decreased $\phi(s)$. Since we are maintaining $\mathsf{ilev}(e) \leq \mathsf{ilev}^{\mathrm{old}}(e)$, the change of $\omega(e)$ could only increase $\omega(s)$. Therefore, if s was tight, it remains tight. Finally, add e to $P_{\mathsf{ilev}(e)}(s)$.

Algorithm 4: $\mathsf{FixLevel}(e, l)$

```
   // We assume that $\mathsf{lev}^{\mathrm{old}}(e) < l \leq \mathsf{ilev}^{\mathrm{old}}(e)$ and $\mathsf{zlev}(e) = \mathsf{lev}(e)$
1  $\mathsf{ilev}(e) \leftarrow h$;
2  foreach $s \ni e$ do
3      $\omega(s) \leftarrow \omega(s) - \omega^{\mathrm{old}}(e) + \omega(e)$;
4  remove $e$ from $P_{\mathsf{ilev}^{\mathrm{old}}(e)}(s)$ for each $s \ni e$ if $e$ is not a freshly inserted element;
5  let $F = \{s \ni e \mid \omega(s) \geq c_s\}$;
6  let $d = \mathsf{ilev}(e) - \mathsf{zlev}(e)$;
7  foreach $s \ni e$ do                                         // raising sets $s \ni e$
8      update the weight $\omega(s) \leftarrow \omega(s) - (1+\epsilon)^{-l} + \omega(e)$;
       // refresh $\omega(s)$ according to up-to-date $\omega(e)$
9      if $\omega(s, \mathsf{lev}(s)+1) \geq c_s$ and $\mathsf{lev}(s) < \min\{\mathsf{base}(s), \mathsf{zlev}(e)\}$ then
10         $\phi(s) \leftarrow 0$;
11         $\mathsf{lev}(s) \leftarrow \min\{\mathsf{base}(s), \mathsf{zlev}(e)\}$;
12         activate all passive elements in $P_{\mathsf{lev}(s)}(s)$;
13     while $\omega(s, \mathsf{lev}(s)+1) \geq c_s$ do
14         $\phi(s) \leftarrow 0$;
15         let $k = \mathsf{lev}(s)$;
16         $\mathsf{lev}(s) \leftarrow k+1$;
17         foreach $e' \in A_k(s)$ do
18             raise $e'$ to level $k+1$;
19             foreach $s' \ni e'$, $s' \neq s$ do
20                 $\phi(s') \leftarrow \phi(s') + \epsilon(1+\epsilon)^{-k-1}$;
21         if $\mathsf{zlev}(e) = k$ then
22             assign $\mathsf{zlev}(e) \leftarrow k+1$, $\mathsf{ilev}(e) \leftarrow k+1+d$;
23             $\omega(s) \leftarrow \omega(s) - \epsilon(1+\epsilon)^{-k-1-d}$;
24         activate all passive elements in $P_{k+1}(s)$;
25     $l_s \leftarrow \mathsf{ilev}(e)$;
26 foreach $s \ni e$ do                                         // finalizing changes
27     $\omega(s) \leftarrow \omega(s) - (1+\epsilon)^{-l_s} + \omega(e)$;
       // refresh $\omega(s)$ according to up-to-date $\omega(e)$
28     if $s \in F$ then
29         $\phi(s) \leftarrow \phi(s) + (1+\epsilon)^{-l_s} - \omega(e)$;
30     add $e$ to $P_{\mathsf{ilev}(e)}(s)$;
```

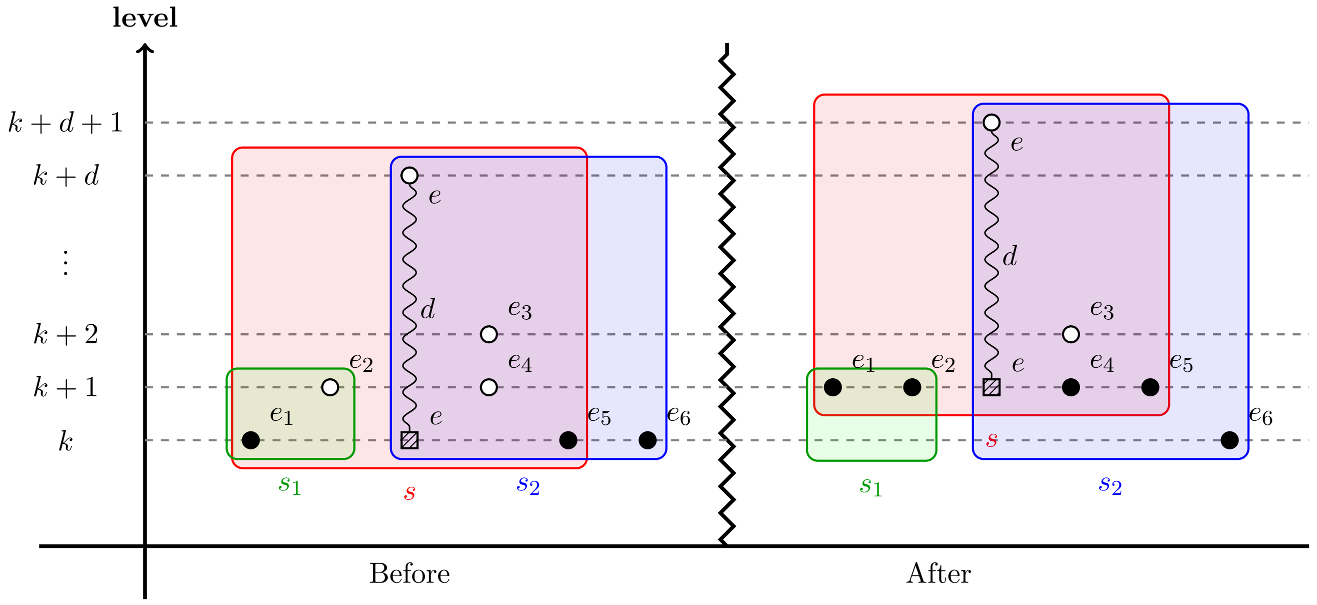

Figure 1: An illustration of a single iteration of the while loop of the $\mathsf{FixLevel}(e, l)$ subroutine, which raises set s by one level. Depicted by black circles are three active elements e_1, e_5, e_6, and depicted by white circles are four passive elements e, e_2, e_3, e_4 (note that e is passive). The small dashed rectangle represents $\mathsf{zlev}(e)$, which is equal to $\mathsf{lev}(e)$. There are three sets: $s = \{e, e_1, e_2, e_3, e_4, e_5\}$ colored red, $s_1 = \{e_1, e_2\}$ colored green, and $s_2 = \{e, e_3, e_4, e_5, e_6\}$ colored blue. The level of all three sets is k. The left and right parts of the figure illustrate the states right before and after the iteration, respectively. During the iteration, e_1 and e_5 were raised from level k to level $k+1$ and they remain active. Elements e_2 and e_4 became active, since they were passive elements at level $k+1$, and since they belong to s. Since the lazy level of e was k, both its lazy and intrinsic levels got raised by one, so the gap between them remains equal to d. The level of s was raised to $k+1$.

3.3.2 Key properties

Recall that for any call to $\mathsf{FixLevel}(e, l)$, we assume Invariant 2.1(1) holds, for every passive element we have $\mathsf{lev}(e') < \mathsf{ilev}(e') \leq \mathsf{zlev}(e') + \lceil \log_{1+\epsilon} \max\{f, \frac{2C}{\epsilon}\} \rceil$, and that $\mathsf{zlev}(e) = \mathsf{lev}(e)$ and $\mathsf{lev}^{\mathrm{old}}(e) < l \leq \mathsf{ilev}^{\mathrm{old}}(e)$. We use the super-script "old" to denote the values of the variables right before the execution of $\mathsf{FixLevel}(e, l)$ started (e.g., $\mathsf{lev}^{\mathrm{old}}(s), \omega^{\mathrm{old}}(s)$). Before we proceed to proving the properties after the call to $\mathsf{FixLevel}(e, l)$, we state some auxiliary observations and claims about what happens during the execution of it.

Observation 3.1. *For any set s' and any element e', the values of $\mathsf{lev}(s')$, $\mathsf{ilev}(e')$ and $\mathsf{zlev}(e')$ can only increase, and the values of $\omega(s')$ and $\omega(e')$ can only decrease during the raising sets $s \ni e$ and finalizing changes steps.*

Observation 3.2. *If $\mathsf{ilev}^{old}(e) = l$, then the call to $\mathsf{FixLevel}(e, l)$ does not make any changes.*

Observation 3.3. *After raising a set s to level $\min\{\mathsf{base}(s), \mathsf{zlev}(e)\}$ or raising s by one level, we have $\mathsf{ilev}(e) - \mathsf{zlev}(e) = d$ and $\mathsf{zlev}(e) = \mathsf{lev}(e)$.*

Claim 3.1. *For any set $s \notin F$, we have $\omega(s, \mathsf{lev}(s) + 1) < c_s$ during the raising sets $s \ni e$ step.*

Proof. If $s \notin F$, then $\omega(s) < c_s$ after setting the intrinsic level of e to l. By Observation 3.1, $\omega(s)$ could only decrease after that, so the entering condition $\omega(s, \mathsf{lev}(s) + 1) \geq c_s$ is never satisfied. $\square$

Observation 3.4. *$\mathsf{lev}(s)$ strictly increases after each iteration of the while loop.*

Claim 3.2. *During the call to $\mathsf{FixLevel}(e, l)$,*

$$\bigcup_{i=0}^{\mathsf{base}(s)-1} A_i(s) \;=\; \bigcup_{i=0}^{\mathsf{base}(s)-1} P_i(s) \setminus \{e\} \;=\; \emptyset.$$

Proof. Suppose for contradiction that there is an element $e' \in s, e' \neq e$ with $\mathsf{ilev}^{\mathrm{old}}(e') \leq \mathsf{base}(s) - 1$. Thus, $\omega^{\mathrm{old}}(e') \geq (1+\epsilon)^{-\mathsf{base}(s)+1}$. But then $\omega^{\mathrm{old}}(s, \mathsf{lev}^{\mathrm{old}}(s)+1) \geq \omega^{\mathrm{old}}(e')/(1+\epsilon) \geq (1+\epsilon)^{-\mathsf{base}(s)} \geq c_s$, which violates Invariant 2.1(1) — a contradiction. Thus,

$$\bigcup_{i=0}^{\mathsf{base}(s)-1} A_i^{\mathrm{old}}(s) = \bigcup_{i=0}^{\mathsf{base}(s)-1} P_i^{\mathrm{old}}(s) \setminus \{e\} = \emptyset.$$

During the call, any element level is non-decreasing by Observation 3.1; therefore,

$$\bigcup_{i=0}^{\mathsf{base}(s)-1} A_i(s) = \bigcup_{i=0}^{\mathsf{base}(s)-1} P_i(s) \setminus \{e\} = \emptyset.$$

$\square$

Next, we show properties after the call to $\mathsf{FixLevel}(e, l)$. First, we state the following observation.

Observation 3.5. *After the call to $\mathsf{FixLevel}(e, l)$, the following holds:*

(1) Element e is passive with $\mathsf{zlev}(e) = \mathsf{lev}(e)$, and the gap $\mathsf{ilev}(e) - \mathsf{zlev}(e)$ is equal to $d = l - \mathsf{lev}^{old}(e)$.

(2) *For any passive element $e' \neq e$, either* $\mathsf{ilev}(e')$ *and* $\mathsf{zlev}(e')$ *remain the same (and hence the gap* $\mathsf{ilev}(e') - \mathsf{zlev}(e')$ *remains the same), or e' becomes active with* $\mathsf{ilev}(e') \geq \mathsf{ilev}^{old}(e')$*.*

(3) *Levels of sets $s \in e$ could have only increased; levels of other sets remain the same.*

(4) *For any set s' such that $e \notin s'$, the value of $\omega(s')$ could have only decreased.*

Recall that for any passive element e' we must have $\mathsf{zlev}(e') \leq \mathsf{lev}(e')$. Since $\mathsf{FixLevel}$ can be invoked from the $\mathsf{Rebuild}$ subroutine, where the inequality $\mathsf{zlev}(e') \leq \mathsf{lev}(e')$ may be temporarily violated, we do not assume that it holds before the call to $\mathsf{FixLevel}(e, l)$. However, we still want to maintain $\mathsf{lev}(e') < \mathsf{ilev}(e') \leq \mathsf{zlev}(e') + \lceil \log_{1+\epsilon} \max\{f, \frac{2C}{\epsilon}\} \rceil$, which we show we do in the following claim.

Claim 3.3. *After the call to* $\mathsf{FixLevel}(e, l)$*, for any passive element e', we have* $\mathsf{lev}(e') < \mathsf{ilev}(e') \leq \mathsf{zlev}(e') + \lceil \log_{1+\epsilon} \max\{f, \frac{2C}{\epsilon}\} \rceil$*.*

Proof. For element e this follows from Observation 3.5(1). Recall that we have $\mathsf{lev}^{\mathrm{old}}(e') < \mathsf{ilev}^{\mathrm{old}}(e') \leq \mathsf{zlev}^{\mathrm{old}}(e') + \lceil \log_{1+\epsilon} \max\{f, \frac{2C}{\epsilon}\} \rceil$ for every passive element $e' \neq e$. Observe that $\mathsf{lev}(e')$ can change only when $\mathsf{lev}(s)$ changes for some $s \ni e$, which can happen only during raising s to level $\min\{\mathsf{base}(s), \mathsf{zlev}(e)\}$ or raising s by one level. In the former case, we have $\mathsf{ilev}(e') \geq \mathsf{base}(s)$ by Claim 3.2, and if $\mathsf{ilev}(e')$ becomes equal to $\mathsf{base}(s)$, then e' gets activated at line 12. In the latter case, $\mathsf{ilev}(e')$ can become equal to $\mathsf{lev}(e')$ due to the increase of $\mathsf{lev}(s)$ by one; in that case, e' gets activated as well at line 24. ☐

The $\mathsf{FixLevel}$ subroutine does not explicitly maintain Invariant 2.1(2). However, we stated that our goal is to make sure that tight sets remain tight and that slack sets are not raised. The following claims show that.

Claim 3.4. *If $d \geq \log_{1+\epsilon} \frac{2C}{\epsilon}$, then* $\mathsf{lev}(s)$ *has not changed for any slack set s after the call to* $\mathsf{FixLevel}(e, l)$*.*

Proof. Since s was slack, we have $\omega^{\mathrm{old}}(s) < c_s/(1+\epsilon)$. Note that $l \geq d$. Therefore, after changing the intrinsic level of e to l, we have

$$\omega(s) < \frac{c_s}{1+\epsilon} + (1+\epsilon)^{-l} \leq c_s.$$

The last inequality holds since $l \geq d \geq \log_{1+\epsilon} \frac{2C}{\epsilon} \geq \log_{1+\epsilon} \frac{2}{\epsilon c_s} \geq \log_{1+\epsilon} \frac{1+\epsilon}{\epsilon c_s}$, for $\epsilon \leq 1$.

Therefore, $s \notin F$. By Claim 3.1, we have $\omega(s, \mathsf{lev}(s)+1) < c_s$, so we do not raise s. ☐

Claim 3.5. *During the execution of* $\mathsf{FixLevel}(e, l)$*, for any set s, the value of* $\omega(s, \mathsf{lev}(s)+1) - \omega(e)$ *can only decrease.*

Proof. By Observation 3.3, we have $\mathsf{ilev}(e) > \mathsf{lev}(e)$ during the execution, so from Equation (1):

$$\omega(s, \mathsf{lev}(s)+1) - \omega(e) = \sum_{e' \in s, e' \neq e} \min\left\{\omega(e'), (1+\epsilon)^{-\max\{\mathsf{lev}(s)+1, \max_{t \mid e' \in t \neq s} \mathsf{lev}(t)\}}\right\}.$$

Observe that changing the intrinsic level of e to l does not affect $\omega(e')$ for $e' \neq e$, so $\omega(s, \mathsf{lev}(s)+1) - \omega(e)$ remains the same. By Observation 3.1, $\mathsf{lev}(s)$ can only increase, and for every $e' \in s$, $\omega(e')$ can only decrease, which can only decrease $\omega(s, \mathsf{lev}(s)+1) - \omega(e)$. ☐

Claim 3.6. *During the execution of* $\mathsf{FixLevel}(e, l)$*, we always have* $\mathsf{ilev}(e) \leq \mathsf{ilev}^{old}(e)$*.*

Proof. After changing the intrinsic level of e to l, we have $\mathsf{ilev}(e) = l \leq \mathsf{ilev}^{\text{old}}(e)$. Observe that $\mathsf{ilev}(e)$ can increase only during an iteration of the while loop, and can only increase by one. Suppose for contradiction that at the beginning of some iteration of the while loop we had $\mathsf{ilev}(e) = \mathsf{ilev}^{\text{old}}(e)$, and thus $\omega(e) = \omega^{\text{old}}(e)$. By Claim 3.5, $\omega^{\text{old}}(s, \mathsf{lev}^{\text{old}}(s) + 1) - \omega^{\text{old}}(e) \geq \omega(s, \mathsf{lev}(s) + 1) - \omega(e)$, so $\omega^{\text{old}}(s, \mathsf{lev}^{\text{old}}(s) + 1) \geq \omega(s, \mathsf{lev}(s) + 1)$. By Invariant 2.1(1), $\omega^{\text{old}}(s, \mathsf{lev}^{\text{old}}(s) + 1) < c_s$, thus $\omega(s, \mathsf{lev}(s) + 1) < c_s$, which contradicts the entering condition of the while loop. □

Claim 3.7. *Any tight set remains tight after the call to* $\mathsf{FixLevel}(e, l)$*.*

Proof. First, observe that during changing the intrinsic level of e to l, weights of sets can only increase, since $l \leq \mathsf{ilev}^{\text{old}}(e)$. Thus, this does not break tightness for any set.

Consider a tight set s. Observe that the tightness for it can be violated only during raising s (the for loop at line 7) or during the finalization step for s (the for loop at line 26). This is so, because during raising some other set s', $\omega(s)$ could change only due to raising elements from $A_k(s')$. However, in that case, the decrease of $\omega(s)$ is immediately compensated by the increase of $\phi(s)$. Therefore, if $e \notin s$, then s remains tight after the call to $\mathsf{FixLevel}(e, l)$.

Next, consider the case $e \in s$. If we do not raise s to level $\min\{\mathsf{base}(s), \mathsf{zlev}(e)\}$ or enter the while loop, then $\omega^*(s)$ could decrease only due to changes of $\omega(e)$. However, we have $\omega(e) \geq \omega^{\text{old}}(e)$ by Claim 3.6, so $\omega^*(s)$ is no smaller than it was at the beginning of the call. Therefore, s remains tight.

If we do not enter the while loop, but raise s to level $\min\{\mathsf{base}(s), \mathsf{zlev}(e)\}$, then observe that we have $\omega(s, \mathsf{lev}(s)+1) \geq c_s$, and hence $\omega(s) \geq c_s$. If we enter the while loop for s, consider the last iteration of it. In that case, we also have $\omega(s) \geq c_s$ at the beginning of that iteration. During it, elements from $A_k(s)$ are raised one level up, and e may also raise one level up, thus decreasing their weights by a factor of $1+\epsilon$. Then, $\omega(s)$ decreases by at most a factor of $1+\epsilon$, so $\omega(s) \geq c_s/(1+\epsilon)$ as a result of this iteration. As we have shown before, $\omega^*(s)$ remains the same during the subsequent iterations. Since we entered the while loop, s must be in F, according to Claim 3.1. Observe that this implies that at the finalization step, the decrease of $\omega(s)$ is compensated by the increase of $\phi(s)$, so s remains tight. □

Finally, recall that $\mathsf{FixLevel}$ aims to maintain Invariant 2.1(1). The following theorem shows that this indeed holds.

Theorem 3.2. *Invariant 2.1(1) holds after the call to* $\mathsf{FixLevel}(e, l)$*.*

Proof. After changing the intrinsic level of e to l, Invariant 2.1(1) could be violated only for sets containing e. The algorithm raises each set $s \ni e$ until the condition of Invariant 2.1(1) is satisfied for s. By Observation 3.1, the weights of sets can only decrease. Therefore, if the condition of Invariant 2.1(1) was satisfied for a set at some point during the call, it is satisfied until the end of the call. It follows that Invariant 2.1(1) holds for every set after the call to $\mathsf{FixLevel}(e, l)$. □

We are ready to prove Theorem 3.1, which shows the correctness of the Insert subroutine, as was stated in Section 3.2.

Theorem 3.1. *After the call to* $\mathsf{Insert}(e)$*, T is a set cover and Invariant 2.1(1)(2) are maintained.*

Proof. We start by analyzing the case $F = \emptyset$. The analysis splits into two subcases.

- The first subcase is that $F = \emptyset$ and $h > \mathsf{lev}(e)$. Let $\omega^{\text{old}}(s)$ be the weight of a set s before the insertion of e. By the minimality of h, there exists $s \ni e$ such that $\omega^{\text{old}}(s) + (1+\epsilon)^{-h+1} \geq c_s$. Dividing both sides by $1+\epsilon$, we get $\frac{1}{1+\epsilon}\omega^{\text{old}}(s) + (1+\epsilon)^{-h} \geq \frac{c_s}{1+\epsilon}$. The left-hand side is at most

$\omega(s) = \omega^{\text{old}}(s) + (1+\epsilon)^{-h}$. Therefore, $\omega(s) \geq c_s/(1+\epsilon)$, so s is tight and the new element e is covered by T.

- The second subcase is that $F = \emptyset$ and $h = \mathsf{lev}(e)$. If $\mathsf{lev}(e) > 0$, then e is covered by a set s with $\mathsf{lev}(s) > 0$. This set is tight by Invariant 2.1(2). Otherwise, $\omega(e) = 1$ and thus any set $s \ni e$ would have $\omega(s) \geq 1$, and so is tight, since $c_s \leq 1$.

In both subcases, Invariant 2.1(1)(2) hold, since weights do not decrease, and $\omega(s) < c_s$ for all $s \ni e$.

Next, let us consider the case where $F \neq \emptyset$. For any $s \in F$, before e was inserted, we had $\omega(s) \geq c_s - (1+\epsilon)^{-l} \geq c_s - \frac{\epsilon}{2C} \geq c_s/(1+\epsilon)$. Hence, all sets in F are tight before we invoke $\mathsf{FixLevel}(e, l)$. By Claim 3.7, sets from F are still tight after the call to $\mathsf{FixLevel}(e, l)$, so e is covered by a tight set. Invariant 2.1(1) holds by Theorem 3.2. To show that Invariant 2.1(2) is also maintained, consider a set s. If it was slack before the insertion, we had $\mathsf{lev}(s) = 0$. After the call to $\mathsf{FixLevel}(e, l)$, it remains at level 0 by Claim 3.4. If s was tight before the insertion, then s remains tight by Claim 3.7. □

3.4 Rebuilding

The $\mathsf{Rebuild}$ subroutine is invoked whenever Invariant 2.1(3) is violated. Let k be the smallest index such that $\phi_{\leq k} > \epsilon\,(c(T_{\leq k}) + f \cdot \omega(E_{\leq k}))$; by Observation 2.1, such an index k can be found in time

$$O\left(\frac{\log C}{\epsilon} + \left|T_{\leq k} \setminus T_{\leq \lceil \log_{1+\epsilon} C \rceil + 1}\right| + |E_{\leq k}|\right).$$

High-level idea. In the $\mathsf{Rebuild}$ procedure we want to eliminate the dead weights of sets up to level k, and then fix the cover and violated invariants by changing levels of elements and sets. If all elements were active (which is the case in the rebuild subroutine of [BHNW21]), we could just raise every set and element below level $k+1$ to level $k+1$, and then invoke the subroutine $\mathsf{WaterFilling}$ for them (refer to Lemma 3.1), which would restore the cover and the invariants by lowering their levels.

However, passive elements pose additional challenges. First, there can be a passive element e with $\mathsf{zlev}(e) \leq k$ and $\mathsf{ilev}(e) > k+1$, which we call a "dirty" element. If we just apply $\mathsf{WaterFilling}$, without considering them, that will cause issues. Invoking $\mathsf{WaterFilling}$ does not guarantee that every set becomes tight, so it might be the case that after the execution of $\mathsf{WaterFilling}$, every set that contains e is slack, so e is not covered by T. Another problem is that we want to bound the time incurred by all the instances of $\mathsf{Rebuild}$ on any passive element e. To do so, we would like to argue that the gap $\mathsf{ilev}(e) - \mathsf{zlev}(e)$ decreases after each call to $\mathsf{Rebuild}$. But even if e is contained in a tight set after the call to $\mathsf{WaterFilling}$, its $\mathsf{lev}(e)$ could decrease as a result, which could force the decrease of $\mathsf{zlev}(e)$ (since we maintain $\mathsf{lev}(e) \geq \mathsf{zlev}(e)$), and hence increase the gap $\mathsf{ilev}(e) - \mathsf{zlev}(e)$.

To overcome these problems, we do the following. First, we raise every set with $\mathsf{lev}(s) \leq k$ to level $k+1$. After that, we raise every "clean" element to level $k+1$; that is, active elements with $\mathsf{lev}(e) \leq k$ and passive with $\mathsf{ilev}(e) \leq k+1$ (note that we need to make passive elements active). Then we raise $\mathsf{zlev}(e)$ to $k+1$ for every "dirty" element, thus decreasing the gap $\mathsf{ilev}(e) - \mathsf{zlev}(e)$. Raising clean elements could decrease weights of some sets. As a result, some elements could stop being covered by a tight set. For clean elements, the cover and the invariants will be repaired by invoking $\mathsf{WaterFilling}$ on them. To repair the cover for dirty elements, we try to decrease $\mathsf{ilev}(e)$ as much as possible if e is not already covered by a tight set, while maintaining $\omega(s) < c_s$ for each $s \ni e$ (and make e active, if necessary). However, it may be still not enough to fix the cover for e.

In that case, we activate e on level $k+1$, we add it to the set of "clean" elements. Finally, we apply $\mathsf{WaterFilling}$ on the set of clean elements, to fix the cover and the invariants for sets that contain them.

However, this approach is not enough by itself, since we cannot afford to go over all sets below level $k+1$ to raise them to level $k+1$. To overcome this hurdle, we observe that many sets are "unnecessary" to restore a valid cover, and would be dropped to level 0 after applying $\mathsf{WaterFilling}$ anyway. Hence we can drop every set below level $k+1$ to level 0 efficiently using the aforementioned idea of *implicit zeroing* (see Section 2.2), and after that $\mathsf{WaterFilling}$ will be applied only on a subset of "necessary" sets below level $k+1$, which we show how to efficiently find.

To bound the runtime, our argument relies on the decrease of the gap $\mathsf{ilev}(e) - \mathsf{zlev}(e)$ after each instance of $\mathsf{Rebuild}$. So far, processing a dirty element required scanning all sets $s \ni e$, and the gap $\mathsf{ilev}(e) - \mathsf{zlev}(e)$ decreased by at least one as a result. To improve the runtime further, we would like to handle dirty elements more efficiently. Our idea here is to try to avoid scanning all the sets $s \ni e$, which takes time $O(f)$. This might be not always possible; in that case, we try to decrease the gap $\mathsf{ilev}(e) - \mathsf{zlev}(e)$ significantly. Suppose we want to increase $\omega(e)$ up to some δ without violating $\omega(s) < c_s$ for any $s \ni e$. There may be some witness set s for which $\omega(s) - \omega(e) + \delta \geq c_s$. If δ is close enough to $\omega(e)$, namely, $\delta - \omega(e) \leq \epsilon \cdot c_s/(1+\epsilon)$, then s must be tight, since $\omega(s) \geq c_s + \omega(s) - \delta \geq c_s/(1+\epsilon)$. Therefore, if there is such a set, then e is covered by a tight set. If there are many witness sets, we can find one by sampling sets $s \ni e$ uniformly at random, which allows us to avoid scanning all sets $s \ni e$; upon failure, we can basically proceed as in the deterministic algorithm in this case, since the sampling fails with small probability.

The other extreme case is when there is no witness set, and then we can increase $\omega(e)$ up to at least δ, and hence decrease the gap $\mathsf{ilev}(e) - \mathsf{zlev}(e)$. The interesting case is when there are a few witness sets, and so the sampling is likely to fail; however, in this case we can apply $\mathsf{FixLevel}$ to decrease the gap. Recall that the amortized runtime cost of invoking $\mathsf{FixLevel}(e, l)$ is roughly $|F| \cdot f \cdot (1+\epsilon)^{-d}$, where $d = l - \mathsf{lev}(e)$ and F is the set of sets $s \ni e$, for which $\omega(s) \geq c_s$ after setting $\mathsf{ilev}(e)$ to l. Notice we take l such that $(1+\epsilon)^{-l} \leq \delta$, then every set in F must be a witness set. As we will argue later, if we define δ carefully, we can achieve a significant decrease of the gap $\mathsf{ilev}(e) - \mathsf{zlev}(e)$, while making sure that the amortized runtime cost stays linear in f.

3.4.1 Description of the Rebuild subroutine

Next, let us describe the algorithm more formally; refer to Algorithm 5 for the pseudocode of the $\mathsf{Rebuild}$ subroutine, and to Algorithms 6 and 8 for the pseudocodes of the auxiliary $\mathsf{HandleDet}$ and $\mathsf{HandleRand}$ subroutines, which are called from within the $\mathsf{Rebuild}$ subroutine.

Definition 3.1. *Consider the moment right before the call to* $\mathsf{Rebuild}(k)$*. An element $e \in E_{\leq k}$ (i.e.,* $\mathsf{zlev}(e) \leq k$*) is called* clean*, if* $\mathsf{ilev}(e) \leq k+1$*, and is called* dirty *otherwise.*

First, go over each element in $E_{\leq k}$ and compute the set D of all dirty elements and the set E of all clean elements among them. We initially define set $S \leftarrow \emptyset$. During the steps of our algorithm, Invariant 2.1(2) may get violated for some sets. In the end of the subroutine, during the post-processing step, we will call $\mathsf{WaterFilling}$ (see line 21 of Algorithm 5), which will restore the invariants for some of the sets, but not necessarily for all sets. Hence we try to collect all possibly affected sets to S, so we are able to deal with them later.

Next, apply implicit zeroing (refer to Section 2.2 for the definition) on all levels in $[0, k]$. Note that this operation may decrease $\mathsf{lev}(e)$ for some elements in $E_{\leq k}$, without modifying $\mathsf{ilev}(e)$ and $\mathsf{zlev}(e)$, thus possibly making them incorrect. However, this will be repaired when we process clean and dirty elements.

Processing clean elements. Let us first process all clean elements the following way. We move each clean element to level $k + 1$, together with all sets containing it; note that all such sets must be below level $k + 1$. More specifically, for each clean element e, we set $\mathsf{ilev}(e), \mathsf{zlev}(e) \leftarrow k + 1$ and go over sets $s \ni e$ to assign $\mathsf{lev}(s) \leftarrow k + 1$, add s to S and update $\omega(s)$ for each $s \ni e$ accordingly. Besides, we also need to update the sets $A_i(s)$ and $P_i(s)$ (which is omitted in the pseudocode).

This repairs the levels of clean elements, but for dirty elements, their lazy levels can be still incorrect for now (i.e., $\mathsf{zlev}(e) > \mathsf{lev}(e)$). Also, by processing clean elements, we could have only decreased the weights of sets. We will make sure that (composite) weights only increase during the subsequent steps, so sets that are tight at this moment will remain tight during the subsequent steps.

Note that after processing clean elements, we have $\mathsf{ilev}(e) \geq k + 1$ for every element. During the subsequent steps, we may raise some sets to level $k + 1$, but because of that, we will not need to raise any element.

Processing dirty elements. Next, we describe how we handle dirty elements. The way we do it depends on whether our algorithm uses randomness or not, which is controlled by the global flag deterministic.

First, initialize set $E' \leftarrow \emptyset$. During processing a dirty element, it could become active at level $k + 1$. The algorithm will make sure that every set above level $k + 1$ is tight; however, sets at level $k + 1$ are not necessarily tight. Thus, we collect such elements to E', so we are able to repair the cover for them later, by invoking WaterFilling on them.

There are two different ways to process an element $e \in D$: one of them uses randomness, while the other does not. We define each of them in the respective subroutine. Refer to Algorithm 6 for the pseudocode of the deterministic subroutine HandleDet(e) and to Algorithm 8 for the randomized subroutine HandleRand(e). To simplify the notation, we assume that k, E', S are passed to them as arguments implicitly.

To process dirty elements, go over each element $e \in D$. If the flag deterministic is set, then we simply call HandleDet(e). Otherwise, we call HandleRand(e) if e is still passive; if not, continue to the next iteration. We need to do this, since in the randomized version, a dirty element could become active during a previous iteration. Refer to Figure 2 for an illustration of processing dirty elements.

Next, we describe the subroutines HandleDet and HandleRand.

The deterministic subroutine HandleDet(e) (for processing an element $e \in D$). In the HandleDet(e) subroutine, we want to raise $\mathsf{zlev}(e)$ to at least $k + 1$ and make sure that e will be covered by a tight set. Due to the way we process clean elements, some sets on positive levels might become slack, and hence e could become not covered by T. In that case, we can try to decrease $\mathsf{ilev}(e)$ (but not below $k + 1$) until some set $s \ni e$ becomes tight. If this is not possible, then we can make e active at level $k + 1$ and add it to E' and all sets $s \ni e$ to S; for them, the cover will be repaired in the end, using the WaterFilling subroutine. Later in the runtime analysis, we will rely on the fact that each dirty element strictly decreases its gap $\mathsf{ilev}(e) - \mathsf{zlev}(e)$.

First, we check if there is a tight set $s \ni e$. If there is such a set s, then we assign $\mathsf{lev}(s) \leftarrow \max\{k + 1, \mathsf{lev}(s)\}$ (thus, raising it to level $k + 1$ if it was below) and set $\mathsf{zlev}(e)$ to $\mathsf{lev}(s)$, making it at least $k + 1$ and restoring $\mathsf{zlev}(e) \leq \mathsf{lev}(e)$.

Otherwise, if all sets $s \ni e$ are slack, we call the DeclLev(e) subroutine, which tries to restore the cover for e by decreasing $\mathsf{ilev}(e)$ as much as possible, adding e to E' and all sets $s \ni e$ to S upon failure.

Algorithm 5: Rebuild(k)

```
let $D$ be the set of all dirty elements, and $E$ be the set of all clean elements;
$S \leftarrow \emptyset$;
apply implicit zeroing for all levels in $[0, k]$;
foreach $e \in E$ do                                   // processing clean elements
    assign lev$(s) \leftarrow k+1$ and add $s$ to $S$ for each $s \ni e$;
    foreach $s \ni e$ do
        $\omega(s) \leftarrow \omega(s) - \omega(e) + (1+\epsilon)^{-k-1}$;
    ilev$(e)$, zlev$(e) \leftarrow k+1$;
$E' \leftarrow \emptyset$;
foreach $e \in D$ do                                   // processing dirty elements
    if deterministic then
        HandleDet$(e)$;
    else
        if $e$ is still passive then
            HandleRand$(e)$;
// post-processing
let $\widehat{E}$ be the elements from $E \cup E'$ that are not contained in a tight set;
set lev$(s) \leftarrow 0$ for every slack set $s \in S$;
let $\widehat{S}$ be the collection of all the sets that contain elements from $\widehat{E}$;
let $\widehat{k} = \min\left\{k+1, \left\lceil \log_{1+\epsilon} \frac{2C \cdot |\widehat{E}|}{\epsilon} \right\rceil \right\}$;
move all elements and sets in $\widehat{E}, \widehat{S}$ to level $\widehat{k}$;
WaterFilling$(\widehat{k}, \widehat{S}, \widehat{E})$;
```

Algorithm 6: HandleDet(e)

```
if there is a tight set $s \ni e$ then
    set lev$(s) \leftarrow \max\{k+1, \text{lev}(s)\}$;
    update zlev$(e) \leftarrow$ lev$(s)$;
else
    DeclLev$(e)$;
```

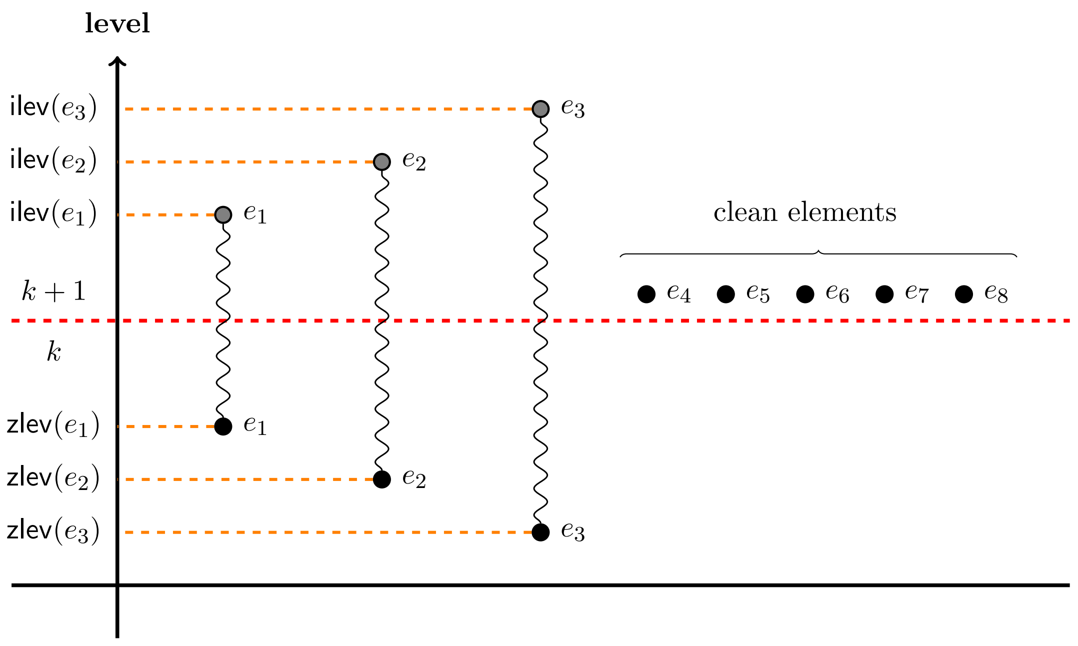

Figure 2: In this illustration, the dirty elements are e_1, e_2, e_3, and the clean elements, which have been raised to level $k+1$, are e_4, e_5, e_6, e_7, e_8. The $\mathsf{Rebuild}(k)$ procedure reduces the gaps $\mathsf{ilev}(e_i) - \mathsf{zlev}(e_i), \forall i \in \{1,2,3\}$. In the deterministic algorithm, the gaps decrease by at least one; in the randomized algorithm, the gaps decrease exponentially (i.e., from d to $\log d$).

The $\mathsf{DeclLev}(e)$ subroutine. The $\mathsf{DeclLev}(e)$ subroutine tries to decrease $\mathsf{ilev}(e)$ (and hence increase $\omega(e)$) as much as possible, while maintaining $\omega(s) < c_s$ for each $s \ni e$. In order to maintain the correctness of levels, it does not decrease $\mathsf{ilev}(e)$ below $\mathsf{lev}(e)$ and $k+1$. Because of that, it is not necessary that at least one of the sets $s \ni e$ becomes tight as a result of decreasing $\mathsf{ilev}(e)$. However, this can only happen when $\mathsf{ilev}(e)$ reaches $k+1$. In that case, we add e to E' and move each set $s \ni e$ to level $k+1$ and add it to S.

To simplify the notation, we assume that k, E', S are passed to the subroutine implicitly. Also, we assume that $\omega(s) < c_s$ for each $s \ni e$ at the beginning of the call. Refer to Algorithm 7 for the pseudocode of the $\mathsf{DeclLev}(e)$ subroutine. The subroutine makes the following steps.

First, compute $l = \max\{k+1, \max_{s \ni e}\{\mathsf{lev}(s)\}\}$, which is the lower bound for $\mathsf{ilev}(e)$. Find the smallest index $h \in [l, \mathsf{ilev}(e)]$ such that $\forall s \ni e$, $\omega(s) - \omega(e) + (1+\epsilon)^{-h} < c_s$. Note that we can find h using binary search in $O(f)$ time, if $\mathsf{ilev}(e) \le \mathsf{zlev}(e) + \lceil \log_{1+\epsilon} \max\{f, \frac{2C}{\epsilon}\} \rceil$, as we argued in the description of the Insert subroutine. We will later show that this condition, i.e., $\mathsf{ilev}(e) \le \mathsf{zlev}(e) + \lceil \log_{1+\epsilon} \max\{f, \frac{2C}{\epsilon}\} \rceil$ is preserved during the execution of $\mathsf{Rebuild}(k)$. After that, we update $\mathsf{ilev}(e)$. To do it, we first set $\omega(s) \leftarrow \omega(s) - \omega(e) + (1+\epsilon)^{-h}$ for each $s \ni e$, and then set $\mathsf{ilev}(e) \leftarrow h$, making $\omega(e) = (1+\epsilon)^{-h}$.

After that, all sets $s \ni e$ may be still slack. Note that this can only happen if all sets $s \ni e$ are at level $k+1$ or below, since we make sure that sets above level $k+1$ are tight. In that case, we add e to E', assign $\mathsf{lev}(s) \leftarrow k+1$ and add s to S for each $s \ni e$, and update $\mathsf{zlev}(e)$ to $\mathsf{lev}(e)$ by setting $\mathsf{zlev}(e) \leftarrow k+1$.

Otherwise, if there is a tight set $s \ni e$, then set $\mathsf{lev}(s) \leftarrow \max\{k+1, \mathsf{lev}(s)\}$ and update $\mathsf{zlev}(e)$ to $\mathsf{lev}(e)$ by setting $\mathsf{zlev}(e) \leftarrow \max_{s \ni e}\{\mathsf{lev}(s)\}$. As a result, $\mathsf{zlev}(e) \ge k+1$ after the call.

Besides, we also need to update the sets $A_i(s)$ and $P_i(s)$ (which is omitted in the pseudocode).

To do it, move e from $P_{\mathsf{ilev}^{\text{old}}(e)}(s)$ to $P_{\mathsf{ilev}(e)}(s)$ (or $A_{\mathsf{ilev}(e)}(s)$ if e has become active) for each $s \ni e$, where $\mathsf{ilev}^{\text{old}}(e)$ is the value of $\mathsf{ilev}(e)$ at the beginning of the call.

Algorithm 7: $\mathsf{DeclLev}(e)$

```
1  let $l = \max\{k+1, \max_{s \ni e}\{\mathsf{lev}(s)\}\}$;
2  find the smallest index $h \in [l, \mathsf{ilev}(e)]$ such that $\forall s \ni e$, $\omega(s) - \omega(e) + (1+\epsilon)^{-h} < c_s$;
3  foreach $s \ni e$ do
4      $\omega(s) \leftarrow \omega(s) - \omega(e) + (1+\epsilon)^{-h}$;
5  $\mathsf{ilev}(e) \leftarrow h$;
6  if all $s \ni e$ are slack then
7      add $e$ to $E'$;
8      assign $\mathsf{lev}(s) \leftarrow k+1$ and add $s$ to $S$ for each $s \ni e$;
9      $\mathsf{zlev}(e) \leftarrow k+1$.
10 else
11     find a tight set $s \ni e$;
12     set $\mathsf{lev}(s) \leftarrow \max\{k+1, \mathsf{lev}(s)\}$;
13     update $\mathsf{zlev}(e) \leftarrow \max_{s \ni e}\{\mathsf{lev}(s)\}$;
```

The randomized subroutine $\mathsf{HandleRand}(e)$ **(for processing an element** $e \in D$**).** In the $\mathsf{HandleRand}(e)$ subroutine, we try to avoid spending $O(f)$ time on scanning all the sets $s \ni e$, and upon failure, we aim to decrease the gap $\mathsf{ilev}(e) - \mathsf{zlev}(e)$ significantly. To do so, we consider the set $\widehat{F} = \{s \ni e \,|\, \omega(s) - \omega(e) + \delta \geq c_s\}$ for some $\delta > \omega(e)$ which we define later. That is, the set of sets $s \ni e$ such that $\omega(s) \geq c_s$ if we increase $\omega(e)$ to δ.

Later we will show that every set in $\widehat{F}$ is tight (see Lemma 3.3). Accordingly, if $\widehat{F} \neq \emptyset$, then e is already covered by T. However, computing $\widehat{F}$ explicitly would take time $O(f)$. We can test whether $\widehat{F} \neq \emptyset$, without computing it explicitly, by sampling sets $s \ni e$ randomly. With some probability, the random sampling may fail to find a set $s \in \widehat{F}$; in that case, we compute $\widehat{F}$ explicitly. Notice that if $\widehat{F}$ is large enough, the probability that we will proceed to computing $\widehat{F}$ is low. Otherwise, if $\widehat{F}$ is small, the expected runtime can be as large as $\Omega(f)$. However, in that case, we argue that we can decrease the gap $\mathsf{ilev}(e) - \mathsf{zlev}(e)$ significantly, so there can be only a few such "expensive" calls. More specifically, if $\widehat{F} = \emptyset$, we can call $\mathsf{DeclLev}(e)$, which will decrease $\mathsf{ilev}(e)$, so as a result, $\omega(e)$ becomes at least δ (or e becomes active). Otherwise, if $\widehat{F}$ is small but not empty, we can apply the $\mathsf{FixLevel}$ subroutine to decrease the gap significantly, which we argue would be cheap enough in that case (we prove that later in Lemma 4.3).

For the randomized version, we assume $f > \frac{2C}{\epsilon}$. Otherwise, if $f \leq \frac{2C}{\epsilon}$, we can simply use the deterministic version, for which the runtime bound is not worse in that case.

Recall that we check if e is still passive at line 14 of Algorithm 5, before processing it. This is necessary in the randomized version of $\mathsf{Rebuild}$, since e could have been activated when processing some previous element from D (note that in that case we do not have the problem with incorrect $\mathsf{zlev}(e)$ anymore).

First, we check if $\mathsf{ilev}(e) - k - 1 \leq \max\{\frac{200}{\epsilon^2}, 1 + 2\log_{1+\epsilon}\frac{2C}{\epsilon}\}$. If that is the case, we fall back to calling $\mathsf{HandleDet}(e)$.

Otherwise, the $\mathsf{HandleRand}(e)$ subroutine performs the following steps. First, find the index η

such that[3]

$$(5\log)^{(\eta+1)}_{1+\epsilon} f \le \mathsf{ilev}(e) - k - 1 \le (5\log)^{(\eta)}_{1+\epsilon} f\,.$$

We will argue later in Claim 3.21 that such an index exists. Note that we can find the index in constant time by computing it at the outset for every value of $\mathsf{ilev}(e) - k - 1$. Next, define

$$\delta = \min\left\{ \left(\frac{1}{(5\log)^{(\eta)}_{1+\epsilon} f} \right)^4, \left(\frac{\epsilon}{2C}\right)^2 \right\} \cdot (1+\epsilon)^{-k-1}\,.$$

Random sampling a witness set. Repeatedly take uniformly random samples of sets $s \ni e$ for $50 \left\lceil \frac{f}{(5\log)^{(\eta)}_{1+\epsilon} f} \right\rceil$ times, and check if $\omega(s) - \omega(e) + \delta \ge c_s$. If there exists a set s for which $\omega(s) - \omega(e) + \delta \ge c_s$, then s is tight (which we prove later in Lemma 3.3), and hence e is covered by a tight set. To update the lazy level of e, assign $\mathsf{lev}(s) \leftarrow \max\{k+1, \mathsf{lev}(s)\}$ and $\mathsf{zlev}(e) \leftarrow \mathsf{lev}(s)$. After that, exit the subroutine.

Handling the element when the random sampling step fails. Reaching this step means we have failed to find a witness set. In that case, compute the set $\widehat{F}$ of all $s \ni e$ such that $\omega(s) - \omega(e) + \delta \ge c_s$, or in other words, the set of all witness sets. We will prove later in Lemma 3.4 that $\delta > \omega(e)$. After that, the execution splits into three branches, depending on the size of $\widehat{F}$:

(1) If $|\widehat{F}| = 0$, then it means that we can safely increase $\omega(e)$ to δ without breaking $\omega(s) < c_s$ for any set $s \ni e$. Hence we call $\mathsf{DecILev}(e)$.

(2) If $0 < |\widehat{F}| \le \left((5\log)^{(\eta)}_{1+\epsilon} f\right)^2$, then we use the $\mathsf{FixLevel}$ subroutine to decrease the gap $\mathsf{ilev}(e) - \mathsf{zlev}(e)$ significantly. We will later show that the amortized cost of invoking it in this case is small enough.

    Go over each $s \ni e$ to assign $\mathsf{lev}(s) \leftarrow \max\{k+1, \mathsf{lev}(s)\}$ if s is tight. Note that there is at least one tight set $s \ni e$, since sets in $\widehat{F}$ are tight. After that, update $\mathsf{zlev}(e) \leftarrow \max_{s\ni e}\{\mathsf{lev}(s)\}$, since $\mathsf{FixLevel}$ requires $\mathsf{zlev}(e) = \mathsf{lev}(e)$. Then, define $l = \min\{\mathsf{ilev}(e), \mathsf{zlev}(e) + \widehat{d}\}$, where:

$$\widehat{d} = \left\lceil \log_{1+\epsilon} \max\left\{ \left((5\log)^{(\eta)}_{1+\epsilon} f\right)^4, \left(\frac{2C}{\epsilon}\right)^2 \right\} \right\rceil\,.$$

    Invoke the subroutine $\mathsf{FixLevel}(e, l)$ and apply the implicit zeroing for sets on level 0. This is because some sets at level 0 can gain dead weight due to the call to $\mathsf{FixLevel}$; however, as we will show later, these sets are not needed for the cover, so we can safely zero the dead weights for them.

(3) Otherwise, $|\widehat{F}| > \left((5\log)^{(\eta)}_{1+\epsilon} f\right)^2$. Recall that every set in $\widehat{F}$ is tight. Thus, we pick an arbitrary set $s \in \widehat{F}$ and assign $\mathsf{lev}(s) \leftarrow \max\{k+1, \mathsf{lev}(s)\}$ and $\mathsf{zlev}(e) \leftarrow \mathsf{lev}(s)$.

[3]Let $(5\log)^{(\eta)}$ denote the $(5\log)$ function, iterated for η times.

Algorithm 8: $\mathsf{HandleRand}(e)$

```
if $\mathsf{ilev}(e) - k - 1 \le \max\{\frac{200}{\epsilon^2}, 1 + 2\log_{1+\epsilon}\frac{2C}{\epsilon}\}$ then
    $\mathsf{HandleDet}(e)$;
else
    find index $\eta$ such that $(5\log)^{(\eta+1)}_{1+\epsilon} f \le \mathsf{ilev}(e) - k - 1 \le (5\log)^{(\eta)}_{1+\epsilon} f$;
    let $\delta = \min\left\{\left(\frac{1}{(5\log)^{(\eta)}_{1+\epsilon} f}\right)^4, \left(\frac{\epsilon}{2C}\right)^2\right\} \cdot (1+\epsilon)^{-k-1}$;
    for $50\left\lceil f/(5\log)^{(\eta)}_{1+\epsilon} f\right\rceil$ times do
        uniformly sample $s \ni e$;
        if $\omega(s) - \omega(e) + \delta \ge c_s$ then
            set $\mathsf{lev}(s) \leftarrow \max\{k+1, \mathsf{lev}(s)\}$;
            update $\mathsf{zlev}(e) \leftarrow \mathsf{lev}(s)$;
            return
    compute $\widehat{F} = \{s \ni e \mid \omega(s) - \omega(e) + \delta \ge c_s\}$;
    if $|\widehat{F}| = 0$ then
        $\mathsf{DeclLev}(e)$;
    else if $0 < |\widehat{F}| \le \left((5\log)^{(\eta)}_{1+\epsilon} f\right)^2$ then
        let $\widehat{d} = \left\lceil \log_{1+\epsilon} \max\left\{\left((5\log)^{(\eta)}_{1+\epsilon} f\right)^4, \left(\frac{2C}{\epsilon}\right)^2\right\}\right\rceil$;
        assign $\mathsf{lev}(s) \leftarrow \max\{k+1, \mathsf{lev}(s)\}$ for each tight $s \ni e$;
        update $\mathsf{zlev}(e) \leftarrow \max_{s \ni e}\{\mathsf{lev}(s)\}$;
        let $l = \min\{\mathsf{ilev}(e), \mathsf{zlev}(e) + \widehat{d}\}$);
        $\mathsf{FixLevel}(e, l)$;
        apply implicit zeroing for sets at level 0;
    else
        take an arbitrary $s \in \widehat{F}$;
        set $\mathsf{lev}(s) \leftarrow \max\{k+1, \mathsf{lev}(s)\}$;
        update $\mathsf{zlev}(e) \leftarrow \mathsf{lev}(s)$;
```

Post-processing. After we have processed elements from D, some elements from $E \cup E'$ may have become covered by tight sets. Consequently, as a post-processing step, we define $\widehat{E}$ as the set of elements from $E \cup E'$ not covered by a tight set. We will argue later that every set at a level greater than $k+1$ is tight, so every element in $\widehat{E}$ is at level $k+1$. Note that there may be sets in S, which are slack and at positive levels, so the condition of Invariant 2.1(2) is violated for them. To restore it, we assign $\mathsf{lev}(s) \leftarrow 0$ for every such a set $s \in S$. Next, we define $\widehat{S}$ to be the collection of all sets that contain elements from $\widehat{E}$; note that $\widehat{S}$ can be computed in $O(f|\widehat{E}|)$ time, by going over each element from $\widehat{E}$.

For the rest, we need to use the following subroutine WaterFilling from [BHNW21, BK19].

Lemma 3.1 ([BHNW21, BK19]). *There is a deterministic subroutine* $\mathsf{WaterFilling}(\widehat{k}, \widehat{S}, \widehat{E})$ *which takes as input a collection of sets* $\widehat{S}$ *and a collection of active elements* $\widehat{E}$ *on level* $\widehat{k}$*, such that for each* $s \in \widehat{S}$*, we have* $\omega(s) < c_s, \phi(s) = 0$*. In the end, all elements in* $\widehat{E}$ *are still active, and the subroutine places each set* $s \in \widehat{S}$ *at level* $\mathsf{lev}(s)$ *such that (1)* $\omega(s) < c_s, \phi(s) = 0$*, and (2) if* $\mathsf{lev}(s) > 0$ *then* $\omega(s) \geq \frac{c_s}{1+\epsilon}$*. The runtime is* $O(f|\widehat{E}| + \widehat{k})$*.*

We cannot directly apply $\mathsf{WaterFilling}(k+1, \widehat{S}, \widehat{E})$, since the runtime would depend on k. Instead, we follow the idea from [BHNW21] and move all elements and sets in $\widehat{E}, \widehat{S}$ to level $\widehat{k} = \min\left\{k+1, \left\lceil \log_{1+\epsilon} \frac{2C\cdot|\widehat{E}|}{\epsilon} \right\rceil\right\}$. The following lemma claims that directly moving elements and sets to level $\widehat{k}$ keeps $\omega(s) < c_s$ for every set $s \in \widehat{S}$, which is required in the conditions of Lemma 3.1. So in the final step, we can safely invoke $\mathsf{WaterFilling}(\widehat{k}, \widehat{S}, \widehat{E})$.

Lemma 3.2 ([BHNW21]). *After moving sets and elements in* $\widehat{S}, \widehat{E}$ *to level* $\widehat{k}$*, each set* $s \in \widehat{S}$ *has weight less than* c_s*.*

3.4.2 Key properties

Now we will prove some properties of Rebuild which are common for both the deterministic and the randomized versions. As we did for the FixLevel subroutine, we use the superscript "old" to denote the values of the variables right before the execution of $\mathsf{Rebuild}(k)$. Recall that we assume that at the beginning of the call, Invariant 2.1(1)(2) hold and every element is in the correct state; i.e. every active element satisfies $\mathsf{ilev}(e) = \mathsf{zlev}(e) = \mathsf{lev}(e)$, and every passive element satisfies $\mathsf{zlev}(e) \leq \mathsf{lev}(e)$ and $\mathsf{lev}(e) < \mathsf{ilev}(e) \leq \mathsf{zlev}(e) + \lceil \log_{1+\epsilon} \max\{f, \frac{2C}{\epsilon}\} \rceil$.

Properties after processing clean elements. First, we show some properties after processing clean elements (see the for loop at line 4 of Algorithm 5). Similarly to $T_{\leq i}$ and $E_{\leq i}$, let $T_{>i}$ be the set of tight sets s such that $\mathsf{lev}(s) > i$ and $E_{>i}$ be the set of elements e such that $\mathsf{zlev}(e) > i$.

Observation 3.6. *During the implicit zeroing and processing clean elements, every clean element becomes active at level* $k+1$*; for every other element,* $\mathsf{zlev}(e)$ *and* $\mathsf{ilev}(e)$ *remain the same.*

Claim 3.8. *During the implicit zeroing and processing clean elements, only sets with* $\mathsf{lev}^{\mathrm{old}}(s) \leq k$ *are affected, and for them* $\mathsf{lev}(s)$ *becomes* 0 *or* $k+1$*.*

Proof. After the implicit zeroing, all such sets are at level 0. Observe that only sets that contain clean elements are affected by processing clean elements. Since $\mathsf{lev}^{\mathrm{old}}(e) \leq k$ for each clean element, such sets are initially at level k or below, and all of them are raised to level $k+1$. □

Since sets at level $k+1$ or above are not affected, we can make the following two corollaries.

Corollary 3.1. *After processing clean elements, elements from $E_{>k}^{old}$ are covered by $T_{>k}$.*

Corollary 3.2. *After processing clean elements, any slack set s either has $\mathsf{lev}(s) = 0$, or $s \in S$ and $\mathsf{lev}(s) = k + 1$.*

Claim 3.9. *After processing clean elements, for each element e, we have $\mathsf{ilev}(e) \geq k + 1$ if e is active, and $\mathsf{ilev}(e) > k + 1$ if e is passive.*

Proof. Observe that by Observation 3.6, it holds for all elements that are not clean. Every clean element e becomes active and $\mathsf{ilev}(e)$ becomes $k + 1$ during processing clean elements. □

Claim 3.10. *After processing clean elements, for each passive element holds $\mathsf{lev}(e) < \mathsf{ilev}(e) \leq \mathsf{zlev}(e) + \lceil \log_{1+\epsilon} \max\{f, \frac{2C}{\epsilon}\} \rceil$, and for each active element holds $\mathsf{ilev}(e) = \mathsf{zlev}(e) = \mathsf{lev}(e)$.*

Proof. Observe that it holds for every clean element. By Observation 3.6, the conditions hold for every other element, for which $\mathsf{lev}(e)$ has remained the same. By Claim 3.8, $\mathsf{lev}(e)$ could have changed only for dirty elements. Consider a dirty element e. By Claim 3.9, we have $\mathsf{ilev}(e) > k+1$. By Claim 3.8, if $\mathsf{lev}(e)$ has changed, then $\mathsf{lev}(e) \leq k + 1$. Thus, the inequality $\mathsf{lev}(e) < \mathsf{ilev}(e)$ holds. □

Claim 3.11. *After processing clean elements, we have $\omega(s) < c_s$ for every $s \notin T_{>k}^{old}$.*

Proof. Observe that $\omega(s) = \omega^{\mathrm{old}}(s, k + 1)$ after processing clean elements, by Equation (1). By Invariant 2.1(1), $\omega^{\mathrm{old}}(s, \mathsf{lev}^{\mathrm{old}}(s) + 1) < c_s$. Since $\mathsf{lev}^{\mathrm{old}}(s) \leq k$, we get $\omega(s) = \omega^{\mathrm{old}}(s, k + 1) \leq \omega^{\mathrm{old}}(s, \mathsf{lev}^{\mathrm{old}}(s) + 1) < c_s$. □

Claim 3.12. *After processing clean elements, we have $\omega(s) < c_s$, $\mathsf{lev}(s) = 0$ and $\phi(s) = 0$ for each set s such that $\mathsf{lev}(s) < k + 1$.*

Proof. By Claim 3.8, only sets with $\mathsf{lev}^{\mathrm{old}}(s) < k + 1$ can have $\mathsf{lev}(s) < k + 1$ after processing clean elements. For such sets, we have $\mathsf{lev}(s) = 0$ and $\phi(s) = 0$ after the implicit zeroing. Observe that during processing clean elements, $\mathsf{lev}(s)$ can only increase to $k + 1$ and $\phi(s)$ remains the same. Finally, we have $\omega(s) < c_s$ by Claim 3.11. □

Claim 3.13. *Invariant 2.1(1) holds after processing clean elements.*

Proof. By Claim 3.8, we only need to prove it for sets s such that $\mathsf{lev}^{\mathrm{old}}(s) \leq k$. Observe that for them, we have $\omega(s) < c_s$ by Claim 3.11. Therefore, Invariant 2.1(1) holds. □

Properties during processing dirty elements. Next, we show some properties after each iteration of the for loop that processes dirty elements (see the for loop at line 10 of Algorithm 5).

First, we state some properties that we will show are maintained after processing each element $e \in D$.

Property 3.1. *Invariant 2.1(1) holds.*

Property 3.2. *For each element e', we have $\mathsf{ilev}(e') \geq k + 1$ if e' is active, and $\mathsf{ilev}(e') > k + 1$ if e' is passive.*

Property 3.3. *Each passive e' satisfies $\mathsf{lev}(e') < \mathsf{ilev}(e') \leq \mathsf{zlev}(e') + \lceil \log_{1+\epsilon} \max\{f, \frac{2C}{\epsilon}\} \rceil$ if it is passive, and satisfies $\mathsf{ilev}(e') = \mathsf{zlev}(e') = \mathsf{lev}(e')$ if it is active.*

Property 3.4. *Any set s such that $\mathsf{lev}(s) < k + 1$, has $\omega(s) < c_s$, $\mathsf{lev}(s) = 0$ and $\phi(s) = 0$.*

Next, we state some claims about the iterations of the for loop. We will prove all of them simultaneously.

Claim 3.14. *Processing an element $e \in D$ maintains Properties 3.1 to 3.4.*

Note that all the conditions of Properties 3.1 to 3.4 hold after processing clean elements by Claims 3.9, 3.10, 3.12 and 3.13.

Recall that due to the implicit zeroing, the condition $\mathsf{zlev}(e) \leq \mathsf{lev}(e)$ can be temporarily violated for a dirty element e. The following claim states that this condition is repaired after processing e.

Claim 3.15. *If an element $e \in D$ has not become active, then $\mathsf{zlev}(e) \leq \mathsf{lev}(e)$ after processing it.*

The following claim shows that processing a dirty element is "local", in a sense, that it does not affect the lazy and intrinsic levels of other passive elements (except possibly making them active).

Claim 3.16. *During processing an element $e \in D$, every passive element $e' \neq e$ either becomes active or $\mathsf{zlev}(e')$ and $\mathsf{ilev}(e')$ remain the same.*

Claim 3.17. *Any tight set stays tight after processing an element $e \in D$.*

Claim 3.18. *Any slack set s either has $\mathsf{lev}(s) = 0$, or $s \in S$ and $\mathsf{lev}(s) = k + 1$.*

Recall that the purpose of S was to collect all sets for which Invariant 2.1(2) might have been violated. Claim 3.18 shows that S indeed contains all such sets. Also, notice that Corollary 3.2 implies that the conditions of Claim 3.18 hold after processing clean elements.

Claim 3.19. *During processing an element $e \in D$, levels of sets can only increase.*

Claim 3.20. *If during processing an element $e \in D$, it was not added to E', then e is covered by a set $s \in T_{>k}$ after processing e.*

Claim 3.21. *If during processing an element $e \in D$ we call $\mathsf{HandleRand}(e)$ and reach the computation of η, then there exists η such that*

$$(5\log)^{(\eta+1)}_{1+\epsilon} f \leq \mathsf{ilev}(e) - k - 1 \leq (5\log)^{(\eta)}_{1+\epsilon} f\,.$$

Before we proceed to the proof of Claims 3.14 to 3.20, let us first prove two lemmas about the properties of δ defined in the algorithm.

Lemma 3.3. *During a call to $\mathsf{HandleRand}(e)$, each set s such that $\omega(s) - \omega(e) + \delta \geq c_s$ is tight.*

Proof. For such s we have $\omega(s) - \omega(e) \geq c_s - \delta \geq c_s - \frac{\epsilon}{2C} \geq \frac{c_s}{1+\epsilon}$, so s is tight. □

Lemma 3.4. *During a call to $\mathsf{HandleRand}(e)$, we have $\delta > \omega(e)$.*

Proof. Observe that if we enter the case where we compute δ, we have $\mathsf{ilev}(e) - k - 1 > 1 + 2\log_{1+\epsilon}\frac{2C}{\epsilon}$. By definition, $\delta \geq \left(\frac{\epsilon}{2C}\right)^2 \cdot (1+\epsilon)^{-k-1}$. Recall that $\omega(e) = (1+\epsilon)^{-\mathsf{ilev}(e)}$. Thus,

$$\omega(e) = (1+\epsilon)^{-\mathsf{ilev}(e)} < (1+\epsilon)^{-2\log_{1+\epsilon}\frac{2C}{\epsilon} - k - 1} = \left(\frac{\epsilon}{2C}\right)^2 \cdot (1+\epsilon)^{-k-1} \leq \delta.$$

□

Proof of Claims 3.14 to 3.21. Consider the iteration that processes an element $e \in D$ and assume the claims hold for each of the previous iterations. Observe that if e is already active, then the algorithm makes no changes, so Claims 3.14 to 3.19 hold vacuously. Observe that e could become active only due to the call to $\mathsf{FixLevel}$ for some previously processed $e' \in D$. At the iteration for e', we had $\mathsf{ilev}(e) > k+1$ by Property 3.2. By Observation 3.5(2), e becomes active with $\mathsf{lev}(e) > k+1$ after that call, and hence there is a set $s \ni e$ with $\mathsf{lev}(s) = \mathsf{lev}(e) > k+1$, which must be tight by Claim 3.18. This set has been tight since then by Claim 3.17, and its level could have only increased by Claim 3.19.

Otherwise, e is passive, and we call $\mathsf{HandleDet}(e)$ or $\mathsf{HandleRand}(e)$. First, we prove Claim 3.21. We need to show that $\mathsf{ilev}(e)-k-1$ is not too large and not too small. According to Observation 3.6 and Claim 3.16, $\mathsf{ilev}(e) = \mathsf{ilev}^{\text{old}}(e)$ and $\mathsf{zlev}(e) = \mathsf{zlev}^{\text{old}}(e) \leq k$, since e is dirty. Therefore, $\mathsf{ilev}(e) - k - 1$ is bounded by $\mathsf{ilev}^{\text{old}}(e) - \mathsf{zlev}^{\text{old}}(e) - 1 \leq \lceil \log_{1+\epsilon} \max\{f, \frac{2C}{\epsilon}\} \rceil - 1$. Recall that in the randomized version we assume $f > \frac{2C}{\epsilon}$. Thus, $\mathsf{ilev}(e) - k - 1 \leq \lceil \log_{1+\epsilon} f \rceil - 1 \leq 5 \log_{1+\epsilon} f$ and η is at least one. Observe that we enter the branch where we compute η only when $\mathsf{ilev}(e) - k - 1 > \frac{200}{\epsilon^2}$. By Lemma 2.1, $(5\log)_{1+\epsilon}(y) \leq \sqrt{\epsilon/5} \cdot y$ for any $y \geq \frac{200}{\epsilon^2}$. Hence, such η exists.

To prove the remaining claims, consider the following three cases and their subcases:

(1) We call $\mathsf{HandleDet}(e)$ or call $\mathsf{HandleRand}(e)$ and fall back to calling $\mathsf{HandleDet}(e)$. Then we branch into two cases:

   (a) There is a tight set $s \ni e$. Observe that if $\mathsf{lev}(s) \geq k+1$, then $\mathsf{lev}(s)$ remains the same. If $\mathsf{lev}(s) < k+1$, then we have $\omega(s) < c_s$. Therefore, Properties 3.1 and 3.4 are preserved. Observe that Property 3.2 is preserved as well. Notice that this does not break Property 3.3, since for any active element e', we have $\mathsf{lev}(e') \geq k+1$, thus $\mathsf{lev}(e')$ is not affected; for any passive element e', we have $\mathsf{ilev}(e') > k+1$, and $\mathsf{lev}(e')$ can increase only to $k+1$. By Claim 3.16 and Observation 3.6, $\mathsf{zlev}(e) = \mathsf{zlev}^{\text{old}}(e) \leq k$ before the iteration, so $\mathsf{zlev}(e)$ can only increase. Therefore, Property 3.3 is preserved, and hence Claim 3.14 holds. Claims 3.15 to 3.20 can be easily verified by the description of the algorithm.

   (b) All sets $s \ni e$ are slack. In that case, we call $\mathsf{DecILev}(e)$. Observe that in the $\mathsf{DecILev}(e)$ subroutine, steps before the branching can only increase $\omega(s)$ and do not break the tightness for any set. Thus, Claim 3.17 holds. Observe that Claims 3.15, 3.16, 3.18 and 3.20 hold as well.

   Suppose we enter the case where all sets $s \ni e$ are slack. Then it must be $\mathsf{lev}(s) \leq k+1$ for all $s \ni e$, since each slack set is either at level 0, or at level $k+1$. Observe that this also implies that levels of sets can only increase, and thus Claim 3.19 holds.

   Since initially we had $\omega(s) < c_s$ for sets $s \ni e$, and due to the way we update the weights, we have $\omega(s) < c_s$ for all $s \ni e$ in the end, and hence Invariant 2.1(1) holds, so Property 3.1 is preserved. Since levels of sets can only increase, and they increase to $k+1$, Property 3.4 is preserved. Observe that $\mathsf{zlev}(e)$ becomes equal to $\mathsf{lev}(e)$ and $\mathsf{ilev}(e) \geq \mathsf{zlev}(e) \geq k+1$, so Properties 3.2 and 3.3 are preserved as well.

(2) We call $\mathsf{HandleRand}(e)$ and do not fall back to $\mathsf{HandleDet}(e)$, and the random sampling step finds a witness set s. In that case, s is tight by Lemma 3.3, and so the proof is identical to case (1)a.

(3) We call $\mathsf{HandleRand}(e)$ and do not fall back to $\mathsf{HandleDet}(e)$, and the random sampling step fails. In that case, we compute $\widehat{F}$ and enter the branching. Consider each branch separately:

(a) If $|\widehat{F}| = 0$, then we call $\mathsf{DeclLev}(e)$. By Lemma 3.4, $\delta > \omega(e)$, so we have $\omega(s) < c_s$ for all sets $s \ni e$. The rest of the proof is identical to case (1)b (with the exception that not all sets $s \ni e$ are necessary slack).

(b) If $0 < |\widehat{F}| \le \left((5\log)^{(\eta)}_{1+\epsilon} f\right)^2$, then we call $\mathsf{FixLevel}(e, l)$. Recall that $\mathsf{FixLevel}(e, l)$ assumes the following: $\mathsf{lev}(e) < l \le \mathsf{ilev}(e)$, we have $\mathsf{zlev}(e) = \mathsf{lev}(e)$, Invariant 2.1(1) holds and for every passive element e' we have $\mathsf{ilev}(e') > \mathsf{lev}(e')$. By the same argument as in case (1)a, all the properties hold after setting $\mathsf{lev}(s) \leftarrow \max\{k+1, \mathsf{lev}(s)\}$ for all $s \in e$ and updating $\mathsf{zlev}(e) \leftarrow \max_{s \ni e}\{\mathsf{lev}(s)\}$. Hence the last three assumptions hold. By Property 3.3, we have $\mathsf{ilev}(e) > \mathsf{lev}(e)$. Thus, by the definition of l, $\mathsf{lev}(e) < l \le \mathsf{ilev}(e)$, and hence the first assumption holds as well.

The properties are preserved by the call to $\mathsf{FixLevel}(e, l)$ and the subsequent implicit zeroing of sets at level 0. First, notice that if $l = \mathsf{ilev}(e)$, then the call makes no changes by Observation 3.2. Otherwise, Property 3.1 is preserved by Theorem 3.2. Property 3.2 is preserved by Observation 3.5(2). Property 3.3 is preserved by Observation 3.5(1)(2) and by Claim 3.3. Next, we show that Property 3.4 is preserved. By Observation 3.5(3), only sets $s \ni e$ can change their level, and their levels can only increase. By Claim 3.4, only tight set can change their level. Observe that all tight sets $s \ni e$ are at level at least $k+1$ before the call to $\mathsf{FixLevel}(e, l)$. Due to the implicit zeroing, $\phi(s) = 0$ for all sets at level 0. Therefore, Property 3.4 is preserved.

Claims 3.15 to 3.17 and 3.19 hold by Observation 3.5 and Claim 3.7. Claim 3.18 holds, since $\mathsf{FixLevel}$ does not raise slack sets by Claim 3.4. Finally, we show that Claim 3.20 holds. Observe that $\widehat{F} \neq \emptyset$ by the entering condition. By Lemma 3.3, every set in $\widehat{F}$ is tight, and it remains tight after the call to $\mathsf{FixLevel}(e, l)$ by Claim 3.7.

(c) If $|\widehat{F}| \ge \left((5\log)^{(\eta)}_{1+\epsilon} f\right)^2$, then we take an arbitrary set $s \in \widehat{F}$. By Lemma 3.3, s is tight, so the proof is identical to case (1)a.

□

Corollary 3.3. *At the beginning of the iteration for $e \in D$, if e is still passive, then* $\mathsf{zlev}(e) = \mathsf{zlev}^{old}(e) \le k$ *and* $\mathsf{ilev}(e) = \mathsf{ilev}^{old}(e)$.

Proof. Since e is dirty, $\mathsf{zlev}^{\mathrm{old}}(e) \le k$, and by Observation 3.6, $\mathsf{ilev}(e)$ and $\mathsf{zlev}(e)$ remain the same after processing clean elements. By Claim 3.16, they remain the same until this iteration. □

Next, let us show that $\widehat{E}$ collects all elements that are not covered by a tight set. We prove the following corollaries.

Corollary 3.4. *After processing dirty elements, every element $e \in E^{old}_{>k}$ is covered by $T_{>k}$.*

Proof. By Claim 3.8, e is covered by $T^{\mathrm{old}}_{>k}$ after processing clean elements, and all these sets are tight. During processing dirty elements, sets from $T^{\mathrm{old}}_{>k}$ remain tight, and their level can only increase by Claims 3.17 and 3.19. □

Corollary 3.5. *After processing dirty elements, every element $e \in D \setminus E'$ is covered by $T_{>k}$.*

Proof. By Claim 3.20, e is covered by a set $s \in T_{>k}$ after the iteration that processes it. By Claims 3.17 and 3.19, s is tight and has $\mathsf{lev}(s) \ge k+1$ after the subsequent iterations. □

Properties after $\mathsf{Rebuild}(k)$. First, we make the following observation about the post-processing steps.

Observation 3.7. *During the post-processing steps,* $\mathsf{ilev}(e)$ *and* $\mathsf{zlev}(e)$ *remain the same for each passive element* e*.*

Next, we show that we apply $\mathsf{WaterFilling}$ on all elements not covered by a tight set.

Claim 3.22. *Before elements from* $\widehat{E}$ *and sets from* $\widehat{S}$ *are moved to level* $\widehat{k}$ *and* $\mathsf{WaterFilling}(\widehat{k}, \widehat{S}, \widehat{E})$ *is invoked,* $\widehat{E}$ *contains all the elements that are currently not covered by a tight set, and* $\widehat{S}$ *contains all the sets that contain elements from* $\widehat{E}$*. Every other element* $e \notin \widehat{E}$ *is covered by* $T_{>k}$*.*

Proof. Every element $e \in E^{\text{old}}_{>k} \cup (D \setminus E')$ is covered by $T_{>k}$ by Corollaries 3.4 and 3.5. Notice that $T_{>k} \cap \widehat{S} = \emptyset$, since sets in $\widehat{S}$ are slack. Therefore, if e is not covered by a tight set, then $e \in (E \cup E')$. Recall that $\widehat{E}$ is the set of elements from $E \cup E'$ that are not covered by a tight set, and $\widehat{S}$ is the collection of all the sets that contain elements from $\widehat{E}$. Observe that whenever we process an element $e \in E$ or add e to E', we add all sets $s \ni e$ to S. Therefore, if $e \in (E \cup E') \setminus \widehat{E}$, then there is a tight set $s \ni e$. ☐

Now we are ready to prove that the $\mathsf{Rebuild}$ subroutine maintains Invariant 2.1(1)(2) and a valid cover.

Theorem 3.3. *Invariant 2.1(1)(2) hold after the call to* $\mathsf{Rebuild}(k)$*.*

Proof. After processing dirty elements, Invariant 2.1(1) holds by Claim 3.14. Observe that the post-processing steps do not change $\omega(s)$ for $s \notin \widehat{S}$. Since only slack sets are dropped to level 0 during the post-processing, Invariant 2.1(1) holds for $s \notin \widehat{S}$. For any $s \in \widehat{S}$, Invariant 2.1(1) will hold after the call to $\mathsf{WaterFilling}(\widehat{k}, \widehat{S}, \widehat{E})$ by Lemma 3.1.

After processing dirty elements, all slack sets at levels above 0 are in S by Claim 3.18. All slack sets in S are dropped to level 0 at line 17, so Invariant 2.1(2) holds for them after that. By the definitions of $\widehat{E}$ and $\widehat{S}$, it must be $\widehat{S} \subseteq S$. For sets from $\widehat{S}$, Invariant 2.1(2) can be violated after moving them to level $\widehat{k}$, but it will be restored afterward, due to the call to $\mathsf{WaterFilling}(\widehat{k}, \widehat{S}, \widehat{E})$, by Lemma 3.1. ☐

Theorem 3.4. T *is a valid set cover for* $\mathcal{U}$ *after the call to* $\mathsf{Rebuild}(k)$*.*

Proof. By Claim 3.22, $\widehat{E}$ contains all elements that are not in any tight set, and $\widehat{S}$ contains all the sets that contain such elements. Every element from $\mathcal{U} \setminus \widehat{E}$ is covered by a tight set from $\mathcal{S} \setminus \widehat{S}$, since all the sets in $\widehat{S}$ are slack. Then, after the call to $\mathsf{WaterFilling}(\widehat{k}, \widehat{S}, \widehat{E})$, all elements will be covered by tight sets by Lemma 3.1. ☐

During the post-processing steps, we drop slack sets from S to level 0, which could potentially decrease $\mathsf{lev}(e)$, and hence violate $\mathsf{zlev}(e) \leq \mathsf{lev}(e)$ for passive elements, or $\mathsf{zlev}(e) = \mathsf{lev}(e)$ for active elements. In the following claim, we show that at the end of the execution of the $\mathsf{Rebuild}(k)$ subroutine, no element will be in an incorrect state.

Theorem 3.5. *After the call to* $\mathsf{Rebuild}(k)$*, for each passive element* e*, we have* $\mathsf{zlev}(e) \leq \mathsf{lev}(e)$ *and* $\mathsf{lev}(e) < \mathsf{ilev}(e) \leq \mathsf{zlev}(e) + \lceil \log_{1+\epsilon} \max\{f, \frac{2C}{\epsilon}\} \rceil$*; for each active element* e*, we have* $\mathsf{ilev}(e) = \mathsf{zlev}(e) = \mathsf{lev}(e)$*.*

Proof. Notice that by Claim 3.14, we have that after processing dirty elements.

The inequality $\mathsf{lev}(e) < \mathsf{ilev}(e) \le \mathsf{zlev}(e) + \lceil \log_{1+\epsilon} \max\{f, \frac{2C}{\epsilon}\} \rceil$ holds for every passive element e, since the post-processing steps can only decrease $\mathsf{lev}(e)$, and they do not affect $\mathsf{ilev}(e)$ and $\mathsf{zlev}(e)$ by Observation 3.7.

Next, we prove the inequality $\mathsf{zlev}(e) \le \mathsf{lev}(e)$ for every passive element e. If $e \in E^{\text{old}}_{>k}$, then $\mathsf{lev}(e)$ and $\mathsf{zlev}(e)$ are the same after the implicit zeroing and processing clean elements by Observation 3.6. If $e \in D \backslash E'$, then by Claim 3.15, $\mathsf{zlev}(e) \le \mathsf{lev}(e)$ after the iteration that processes e. By Claims 3.16 and 3.19, $\mathsf{ilev}(e)$ remains the same and $\mathsf{lev}(e)$ could only increase during the subsequent iterations. During the post-processing steps, dropping slack sets from S to level 0 at line 17 might break the inequality. However, by Corollaries 3.4 and 3.5, e is covered by $T_{>k}$, and by Claim 3.18, every slack set that was dropped was at level $k + 1$. Therefore, $\mathsf{lev}(e)$ is not affected.

Finally, we prove that $\mathsf{ilev}(e) = \mathsf{zlev}(e) = \mathsf{lev}(e)$ for every active element e. By Properties 3.2 and 3.3, it holds after processing dirty elements and $\mathsf{lev}(e) \ge k + 1$. If $e \in \widehat{E}$, then the equality will hold after the call to $\mathsf{WaterFilling}$. Otherwise, e is covered by $T_{>k}$, and as we have shown before, $\mathsf{lev}(e)$ is not affected by dropping slack sets from S to level 0 at line 17. $\square$

4 Update time analysis

4.1 Potential functions

Following [BHNW21], we define the *up*, *down* and *lift potentials*. In addition, we introduce two new types of potential, which we call the *passive potential* and the *clean potential*; the definitions of these five types of potentials are given below, along with some intuitive explanations behind the definitions. We define the *total potential* of the set system, denoted by Φ, as the sum of all types of potentials across all elements and sets.

- **Up potential.** Define $\alpha_i = 2f\left(\frac{3}{\epsilon^3} + \frac{\log C}{\epsilon^2}\right)(1+\epsilon)^{i+1}$, for any index $0 \le i \le L$. Then, the *up potential* of s is defined as $\Phi_{\text{up}}(s) = \max\{\omega(s) - c_s, 0\} \cdot \alpha_{\mathsf{lev}(s)}$.

  Intuition. This type of potential, which is gained by element insertions, is used to cover the costs of raising active elements by the $\mathsf{FixLevel}$ subroutine. Whenever we violate Invariant 2.1(1), the process of raising active elements releases a sufficiently large amount of potential to cover the costs. Notice that potential release due to the loss of one unit of weight can cover the increase of down potential due to increase of f units of dead weight.

- **Down potential.** Define $\beta_i = \frac{2}{\epsilon^2}(1+\epsilon)^{i+1}$ for any index $0 \le i \le L$. Then, the *down potential* of s is defined as $\Phi_{\text{down}}(s) = \phi(s) \cdot \beta_{\mathsf{lev}(s)}$.

  Intuition. This type of potential is used to cover the costs of the $\mathsf{Rebuild}$ subroutine. We gain it whenever we lose weight of some elements due to element deletions or raising levels of elements. Whenever Invariant 2.1(3) is violated, we have gained large enough down potential to cover the costs of the $\mathsf{Rebuild}$ subroutine, which aims at restoring the invariant.

- **Lift potential.** Each set s has a *lift potential* of $\Phi_{\text{lift}}(s) = L - \max\{\mathsf{lev}(s), \mathsf{base}(s)\}$.

  Intuition. This type of potential is used to cover the costs of raising levels of sets above their base levels due to the $\mathsf{FixLevel}$ subroutine. Initially, we have the maximum amount L of that potential for every set. After that, it is restored due to the $\mathsf{Rebuild}$ subroutine. It is needed to cover the cases where we do not raise any active elements, and hence do not decrease the up potential.

- **Passive potential.** Each element e has a *passive potential* $\Phi(e) = f$ if e is passive, and $\Phi(e) = 0$ otherwise.

  Intuition. This type of potential is used to cover the costs of activating an element. This potential is gained by element insertions.

- **Clean potential.** Each set s such that $\phi(s) \neq 0$ has a *clean potential* $\Phi_{\text{clean}}(s) = \frac{1}{\epsilon^2} + \frac{\log C}{\epsilon}$, and $\Phi_{\text{clean}}(s) = 0$ otherwise.

  Intuition. This potential is used to cover the costs associated with the $\mathsf{Rebuild}$ subroutine. Sometimes, the decrease of down potential is not enough to cover the $O(\frac{1}{\epsilon^2} + \frac{\log C}{\epsilon})$ term in the runtime cost. But in that case, we argue it can be covered by the decrease of clean potential.

Initially, when $\mathcal{U} = \emptyset$, by definition of our potential functions, the set system has potential at most $mL = O(m \log(Cn))$. This is because the lift potential is at most L for each set, $\omega(s), \phi(s) = 0$ for each set $s \in \mathcal{S}$, and there are no passive elements. We note that the amortized update time (associated with potential function Φ) of an element update is defined as the sum of the potential change $\Delta\Phi$ due to the update and the actual time spent by the update algorithm; for technical convenience, we shall assume that any unit of potential can cover $O(1)$ units of time. Consequently, in amortized analysis via the potential function method, we would like cheap operations (such as element deletions in our case) to increase the total potential, but not by too much, whereas costly operations (such as the $\mathsf{Rebuild}$ subroutine) should decrease the total potential in roughly the same amount as their actual running time.

4.2 Deletion

By the algorithm's description, the $\mathsf{Delete}(e)$ subroutine takes $O(f)$ time. Hence, it suffices to bound the potential increase following an execution of the $\mathsf{Delete}(e)$ subroutine.

Observe that the up potential may only decrease, and the lift and passive potentials for elements $e' \neq e$ remain unchanged; $\Phi(e)$ may only decrease. To bound the increase in down potential, note that for each $s \ni e$ that was tight before the deletion of element e, its dead weight increases by at most $\omega(e)$. We know that $\mathsf{lev}(e) \geq \mathsf{lev}(s)$; if e is passive, then we also have $\mathsf{ilev}(e) > \mathsf{lev}(e) \geq \mathsf{lev}(s)$. Therefore, $\omega(e) \leq (1+\epsilon)^{-\mathsf{lev}(s)}$, and the increase of $\Phi_{\text{down}}(s)$ is at most

$$\omega(e) \cdot \beta_{\mathsf{lev}(s)} \leq (1+\epsilon)^{-\mathsf{lev}(s)} \cdot \frac{2}{\epsilon^2} \cdot (1+\epsilon)^{\mathsf{lev}(s)+1} \leq \frac{2(1+\epsilon)}{\epsilon^2}.$$

The clean potential may also increase due to the increase of $\phi(s)$; the increase of $\Phi_{\text{clean}}(s)$ is bounded by $\frac{1}{\epsilon^2} + \frac{\log C}{\epsilon}$. Hence, the total increase of Φ after the deletion of e (and the amortized runtime cost) is $O\left(\frac{f}{\epsilon^2} + \frac{f \log C}{\epsilon}\right)$.

4.3 Fixing levels

Recall that we assume Invariant 2.1(1) holds before the execution of $\mathsf{FixLevel}(e, l)$ started. As we did for the properties, use the super-script "old" to denote the values of the variables before the execution of $\mathsf{FixLevel}(e, l)$ started. We only need to consider the case where l is strictly smaller than the old value of $\mathsf{ilev}^{\text{old}}(e)$; otherwise, since Invariant 2.1(1) held before, none of the while loops on line 13 would be triggered, and the algorithm would make no changes by Observation 3.2. The total amortized runtime of the procedure in this case would be $O(f)$.

During the call, we may activate passive elements in $P_{\mathsf{lev}(s)}(s)$ for some set s (see lines 12 and 24 in the pseudocode of the $\mathsf{FixLevel}$ subroutine). The activation of a passive element e' takes time

$O(f)$. However, this runtime cost can be charged to the clearance of its passive potential $\Phi(e')$; indeed, note that the passive potential of any passive element is at least f and that of any active elements is 0.

For the rest, we will only be concerned with other steps in the subroutine. Consider the moment just before an iteration of the while loop (or before entering the branching at line 9). Let $\omega^{\text{new}}(s)$, $\Phi_{\text{up}}^{\text{new}}(s)$ and $\mathsf{lev}^{\text{new}}(s)$ be the weight, the up potential and the level of s right after this iteration (or the branching at line 9) respectively. Define $\widehat{\omega}(s) = \omega(s) - \omega(e)$ and $\widehat{\omega}^{\text{new}}(s) = \omega^{\text{new}}(s) - \omega^{\text{new}}(e)$.

First, we bound the potential increase due to raising s to level $\min\{\mathsf{base}(s), \mathsf{zlev}(e)\}$ (i.e., when we enter the if statement at line 9). It is easy to see that $\Phi_{\text{down}} + \Phi_{\text{clean}}$ does not increase, since we zero out $\phi(s)$. However, Φ_{up} may increase.

Claim 4.1. *If $s \in F$, then the increase of $\Phi_{up}(s)$ due to raising s to level $\min\{\mathsf{base}(s), \mathsf{zlev}(e)\}$ is at most $\omega^{new}(e) \cdot \alpha_{\mathsf{lev}^{new}(s)} - \omega(e) \cdot \alpha_{\mathsf{lev}(s)}$.*

Proof. Let $k = \mathsf{lev}(s)$ and $k' = \mathsf{lev}^{\text{new}}(s)$. By Invariant 2.1(1) and Claim 3.5, we have $\omega(s, \mathsf{lev}(s) + 1) - \omega(e) \leq \omega^{\text{old}}(s, \mathsf{lev}^{\text{old}}(s) + 1) < c_s$. By Claim 3.2, there are no elements in $A_k(s)$, so from Equation (2), we have $\omega(s, \mathsf{lev}(s) + 1) = \omega(s)$. Hence $\widehat{\omega}(s) = \omega(s) - \omega(e) < c_s$. Observe that no weight of an element is changed, so $\omega(s) = \omega^{\text{new}}(s)$ and $\widehat{\omega}(s) = \widehat{\omega}^{\text{new}}(s)$. By the entering condition, $\omega(s) = \omega^{\text{new}}(s) = \omega(s, \mathsf{lev}(s) + 1) \geq c_s$. Therefore,

$$\begin{aligned}\Phi_{\text{up}}^{\text{new}}(s) - \Phi_{\text{up}}(s) &= (\widehat{\omega}^{\text{new}}(s) + \omega^{\text{new}}(e) - c_s) \cdot \alpha_{k'} - (\widehat{\omega}(s) + \omega(e) - c_s) \cdot \alpha_k \\ &= (\widehat{\omega}(s) - c_s) \cdot (\alpha_{k'} - \alpha_k) + \omega^{\text{new}}(e) \cdot \alpha_{k'} - \omega(e) \cdot \alpha_k \\ &\leq \omega^{\text{new}}(e) \cdot \alpha_{k'} - \omega(e) \cdot \alpha_k.\end{aligned}$$

□

Next, we analyze the potential increase due to a single iteration of the while loop. Consider the iteration of the for loop that raises a set $s \ni e$. Note that if $s \notin F$, then we do not enter the while loop by Claim 3.1. So the interesting case is when $s \in F$. We are going to prove some claims about that iteration. Recall that k is the level of s at the beginning of an iteration of the while loop.

Lemma 4.1. *At the beginning of each iteration of the while loop, we have $\omega(s) - \omega(e) - |A_k(s)| \cdot \epsilon(1+\epsilon)^{-k-1} < c_s$.*

Proof. Since we assumed that Invariant 2.1(1) has held before the execution of $\mathsf{FixLevel}$, we have $\omega^{\text{old}}(s, \mathsf{lev}^{\text{old}}(s) + 1) < c_s$. By Claim 3.5, $\omega(s, k+1) - \omega(e) \leq \omega^{\text{old}}(s, \mathsf{lev}^{\text{old}}(s) + 1) - \omega^{\text{old}}(e)$. Thus, $\omega(s, k+1) - \omega(e) < c_s$. Since $\omega(s, k+1) = \omega(s) - |A_k(s)| \cdot \epsilon(1+\epsilon)^{-k-1}$ by Equation (2), we get the desired inequality. □

Claim 4.2. *If $s \in F$, then the increase of $\Phi_{up}(s)$ after a single iteration of the while loop is at most*

$$-2f \cdot |A_k(s)| \cdot \left(\frac{3}{\epsilon^2} + \frac{\log C}{\epsilon}\right) + \omega^{new}(e) \cdot \alpha_{k+1} - \omega(e) \cdot \alpha_k. \tag{3}$$

For any other set $s' \neq s$ the up potential $\Phi_{up}(s')$ does not increase.

Proof. If $\omega^{\text{new}}(s) \leq c_s$, then $\Phi_{\text{up}}^{\text{new}}(s) = 0$. So by the while loop condition that $\omega(s, k+1) \geq c_s$ and by Equation (2), we know that the up potential change is equal to

$$-\Phi_{\text{up}}(s) = -(\omega(s) - c_s) \cdot \alpha_k \leq -(\omega(s) - \omega(s, k+1)) \cdot \alpha_k \leq -2f \cdot |A_k(s)| \cdot \left(\frac{3}{\epsilon^2} + \frac{\log C}{\epsilon}\right).$$

Otherwise, we have $\omega^{\text{new}}(s) > c_s$. Then,

$$\Phi_{\text{up}}^{\text{new}}(s) - \Phi_{\text{up}}(s) = (\widehat{\omega}^{\text{new}}(s) + \omega^{\text{new}}(e) - c_s) \cdot \alpha_{k'} - (\widehat{\omega}(s) + \omega(e) - c_s) \cdot \alpha_k.$$

By the algorithm, all elements from $A_k(s)$ are raised to level $k+1$, and so their weights decrease by the factor of $1+\epsilon$. Hence, we get $\widehat{\omega}^{\text{new}}(s) = \widehat{\omega}(s) - |A_k(s)| \cdot \epsilon(1+\epsilon)^{-k-1}$. Notice that $\alpha_{k+1} = (1+\epsilon)\alpha_k$. Therefore,

$$\begin{aligned}\widehat{\omega}^{\text{new}}(s) \cdot \alpha_{k+1} - \widehat{\omega}(s) \cdot \alpha_k &= \widehat{\omega}(s) \cdot (1+\epsilon)\alpha_k - |A_k(s)| \cdot \epsilon(1+\epsilon)^{-k} \cdot \alpha_k - \widehat{\omega}(s) \cdot \alpha_k \\ &= (\widehat{\omega}(s) - |A_k(s)| \cdot (1+\epsilon)^{-k}) \cdot \epsilon\alpha_k \\ &= (\omega(s) - \omega(e) - |A_k(s)| \cdot (1+\epsilon)^{-k}) \cdot \epsilon\alpha_k.\end{aligned}$$

Using Lemma 4.1, we can bound it by $(c_s - |A_k(s)| \cdot (1+\epsilon)^{-k-1}) \cdot \epsilon\alpha_k$. Then,

$$\begin{aligned}\Phi_{\text{up}}^{\text{new}}(s) - \Phi_{\text{up}}(s) &\le -|A_k(s)| \cdot (1+\epsilon)^{-k-1} \cdot \epsilon\alpha_k + \omega^{\text{new}}(e) \cdot \alpha_{k+1} - \omega(e) \cdot \alpha_k \\ &= -2f \cdot |A_k(s)| \cdot \left(\frac{3}{\epsilon^2} + \frac{\log C}{\epsilon}\right) + \omega^{\text{new}}(e) \cdot \alpha_{k+1} - \omega(e) \cdot \alpha_k.\end{aligned}$$

□

The total down potential Φ_{down} and the total clean potential Φ_{clean} may change after an iteration of the while loop, and specifically increase, due to changes of dead weights and the increase of $\mathsf{lev}(s)$.

Claim 4.3. *If $s \in F$, then the overall increase of $\Phi_{down} + \Phi_{clean}$ after a single iteration of the while loop is at most $(f-1) \cdot |A_k(s)| \cdot \left(\frac{3}{\epsilon^2} + \frac{\log C}{\epsilon}\right)$.*

Proof. Observe that $\phi(s)$ is zeroed out at the beginning of the iteration, and later it does not increase, so $\Phi_{\text{down}}(s) = 0$. By the algorithm, each element $e' \in A_k(s)$ incurs an increase of each $\phi(s')$ by at most $\epsilon(1+\epsilon)^{-k-1}$. So the overall increase of dead weights for $s' \ni e', s' \ne s$ is at most $(f-1) \cdot |A_k(s)| \cdot \epsilon(1+\epsilon)^{-k-1}$. As at the beginning of the iteration, we have $\mathsf{lev}(s') \le \mathsf{lev}(e') = k$ for $s' \ne s$, and $\mathsf{lev}(s')$ is unchanged during it, the total increase of down potential due to these sets is bounded by

$$(f-1) \cdot |A_k(s)| \cdot \epsilon(1+\epsilon)^{-k-1} \cdot \beta_k = (f-1) \cdot |A_k(s)| \cdot \frac{2}{\epsilon}.$$

For each s', the increase of $\Phi_{\text{clean}}(s')$ is bounded by $\frac{1}{\epsilon^2} + \frac{\log C}{\epsilon}$; therefore, the total increase of clean potential due to these sets is at most

$$(f-1) \cdot |A_k(s)| \cdot \epsilon(1+\epsilon)^{-k-1} \cdot \beta_k = (f-1) \cdot |A_k(s)| \cdot \left(\frac{1}{\epsilon^2} + \frac{\log C}{\epsilon}\right).$$

□

Claim 4.4. *After the call to $\mathsf{FixLevel}(e, l)$, the sum $\Phi_{up} + \Phi_{down} + \Phi_{clean}$ increases by at most*

$$3|F| \cdot \left(\frac{3f}{\epsilon^3} + \frac{f \log C}{\epsilon^2}\right) \cdot (1+\epsilon)^{-d+1} + |F| \cdot \left(\frac{1}{\epsilon^2} + \frac{\log C}{\epsilon}\right).$$

Proof. Consider a set $s \ni e$. If $s \notin F$, then $\Phi_{\text{up}}(s)$ does not increase after refreshing $\omega(s)$ according to the up-to-date $\omega(e)$, since it can only decrease $\omega(s)$ by Claim 3.6. By Claim 3.1, we do not raise s, so $\Phi_{\text{up}} + \Phi_{\text{down}} + \Phi_{\text{clean}}$ does not increase after this iteration.

Now consider the case when $s \in F$. When $\omega(s)$ is refreshed at the beginning of an iteration of the for loop, $\Phi_{\text{up}}(s)$ increases (compared to the beginning of the call) by at most $\omega(e) \cdot \alpha_{\mathsf{lev}(s)} - \omega^{\text{old}}(e) \cdot \alpha_{\mathsf{lev}(s)}$, and $\Phi_{\text{down}}(s)$ and $\Phi_{\text{clean}}(s)$ remain unchanged.

Consider an iteration of the while loop for s. According to Claim 4.2, the first term $-2f \cdot |A_k(s)| \cdot \left(\frac{3}{\epsilon^2} + \frac{\log C}{\epsilon}\right)$ in Equation (3) is enough to cover the increase of $\Phi_{\text{down}} + \Phi_{\text{clean}}$ which is at most $(f-1) \cdot |A_k(s)| \cdot \left(\frac{3}{\epsilon^2} + \frac{\log C}{\epsilon}\right)$ by Claim 4.3.

Summing the terms $\omega^{\text{new}}(e) \cdot \alpha_{k+1} - \omega(e) \cdot \alpha_k$ from Equation (3) over all iterations of the while loop, together with the increase due to raising s to level $\min\{\mathsf{base}(s), \mathsf{zlev}(e)\}$ from Claim 4.1, and the initial increase in $\Phi_{\text{up}}(s)$ due to refreshing $\omega(s)$ according to the up-to-date $\omega(s)$, the result is bounded by the value of $\omega(e) \cdot \alpha_{\mathsf{lev}(s)}$ at the moment after the last iteration of the while loop. Since $\mathsf{ilev}(e) \geq \mathsf{lev}(s) + d$, this value is at most $2f\left(\frac{3}{\epsilon^3} + \frac{\log C}{\epsilon^2}\right) \cdot (1+\epsilon)^{-d+1}$. Therefore, the total increase of $\Phi_{\text{up}} + \Phi_{\text{down}} + \Phi_{\text{clean}}$ is at most

$$2f\left(\frac{3}{\epsilon^3} + \frac{\log C}{\epsilon^2}\right) \cdot (1+\epsilon)^{-d+1}.$$

During the finalization step, $\omega(s)$ can only decrease, so $\Phi_{\text{up}}(s)$ does not increase. If $s \in F$, we also update the value of $\phi(s)$, so $\Phi_{\text{down}}(s)$ and $\Phi_{\text{clean}}(s)$ can increase. Consider the moment when we assigned $l_s \leftarrow \mathsf{ilev}(e)$. At this moment we had $\mathsf{ilev}(e) \geq \mathsf{lev}(s) + d$. Observe that since that, $\mathsf{lev}(s)$ remains unchanged until the end. Therefore, the increase of $\Phi_{\text{down}}(s)$ is at most

$$\begin{aligned}\beta_{\mathsf{lev}(s)} \cdot (1+\epsilon)^{-l_s} &\leq \frac{2}{\epsilon^2} \cdot (1+\epsilon)^{\mathsf{lev}(s)+1} \cdot (1+\epsilon)^{-\mathsf{lev}(s)-d} \\ &< f\left(\frac{3}{\epsilon^3} + \frac{\log C}{\epsilon^2}\right) \cdot (1+\epsilon)^{-d+1}.\end{aligned}$$

The increase of $\Phi_{\text{clean}}(s)$ is at most $\frac{1}{\epsilon^2} + \frac{\log C}{\epsilon}$.

To conclude, the overall increase in potential is at most

$$3|F| \cdot \left(\frac{3f}{\epsilon^3} + \frac{f \log C}{\epsilon^2}\right) \cdot (1+\epsilon)^{-d+1} + |F| \cdot \left(\frac{1}{\epsilon^2} + \frac{\log C}{\epsilon}\right).$$

□

Now we are ready to bound the amortized runtime cost of the $\mathsf{FixLevel}$ subroutine. The steps outside the while loop and activating passive elements take $O(f)$ time in total. We already have shown that the runtime cost of activating passive elements could be charged to the clearance of their passive potentials. The runtime of a single iteration of the while loop is $O(f \cdot |A_k(s)| + 1)$. If $A_k(s) \neq \emptyset$, then we can cover this runtime cost by the decrease in potential, according to Claims 4.2 and 4.3.

Next, consider the case $A_k(s) = \emptyset$. If $k \geq \mathsf{base}(s)$, then we can cover the $O(1)$ runtime cost by the decrease of $\Phi_{\text{lift}}(s)$. Notice that if $k < \mathsf{base}(s)$ at the first iteration of the while loop, then we have $\mathsf{zlev}(e) = \mathsf{lev}(s)$, and during each iteration, both $\mathsf{zlev}(e)$ and $\mathsf{lev}(s)$ will increase by one. Indeed, if we had $\mathsf{lev}(s) < \mathsf{zlev}(e)$ before we entered the while loop, we must have entered the if statement at line 9, after which $\mathsf{lev}(s)$ becomes equal to $\mathsf{zlev}(e)$. Therefore, after each iteration, where $k < \mathsf{base}(s)$, we have $\mathsf{zlev}(e) = \mathsf{lev}(s)$ and both of them increase by one. Notice that

$\mathsf{base}(s) \le \lceil \log_{1+\epsilon} C \rceil$ for each set s. Hence, there are $O(\frac{\log C}{\epsilon}) = O(f)$ such iterations, and the total runtime we spend on them is $O(f)$.

Observe that for any element $e' \neq e$, the passive potential $\Phi(e')$ could only decrease, and Φ_{lift} does not increase. $\Phi(e)$ increase by at most f, if e is a freshly inserted element. The increase of $\Phi_{\text{up}} + \Phi_{\text{down}} + \Phi_{\text{clean}}$ is bounded by Claim 4.4.

Therefore, we conclude our analysis by the following theorem.

Theorem 4.1. *The amortized runtime cost of* $\mathsf{FixLevel}(e, l)$ *is bounded by*

$$3|F| \cdot \left(\frac{3f}{\epsilon^3} + \frac{f \log C}{\epsilon^2}\right) \cdot (1+\epsilon)^{-d+1} + |F| \cdot \left(\frac{1}{\epsilon^2} + \frac{\log C}{\epsilon}\right) + O(f).$$

4.4 Rebuilding

As before, we use the superscript "old" to denote the values of the variables right before $\mathsf{Rebuild}(k)$ started. Similarly to [BHNW21], we will argue that due to any call to $\mathsf{Rebuild}(k)$, we have released a large amount of potential to compensate for the update time. There is a significant difference, however: the release of down potential allows us to compensate the costs associated with active elements only. Thus, the majority of our argument is devoted to dealing with passive elements.

Lemma 4.2 ([BHNW21])**.** *We have the following lower bound on the down potential* $\Phi_{down}(T_{\le k})$

$$\sum_{s \in T_{\le k}} 2\max\{\mathsf{lev}(s) - \mathsf{base}(s) + 1, 0\} + \frac{2f}{\epsilon}|A_{\le k}|$$

Proof. By definition, k is the minimum index such that:

$$\phi_{\le k} \; > \; \epsilon \cdot \left(c(T_{\le k}) + f \cdot \omega(E_{\le k})\right), \tag{4}$$

$$\phi_{\le i} \; \le \; \epsilon \cdot \left(c(T_{\le i}) + f \cdot \omega(E_{\le i})\right), \forall 0 \le i < k. \tag{5}$$

Therefore, we have:

$$\begin{aligned}
\Phi_{\text{down}}(T_{\le k}) &= \sum_{i=0}^{k} \phi_i \cdot \beta_i = \beta_k \cdot \phi_{\le k} - \sum_{i=0}^{k-1} (\beta_{i+1} - \beta_i) \cdot \phi_{\le i} \\
&> \beta_k \cdot \epsilon \cdot (c(T_{\le k}) + f \cdot \omega(E_{\le k})) - \sum_{i=0}^{k-1} (\beta_{i+1} - \beta_i) \cdot \epsilon \cdot (c(T_{\le i}) + f \cdot \omega(E_{\le i})) \\
&= \epsilon \sum_{i=0}^{k} \beta_i \cdot c(T_i) + \epsilon f \cdot \sum_{i=0}^{k} \beta_i \cdot \omega(E_i) \quad \text{(by Abel transformation)} \\
&\ge \epsilon \sum_{i=0}^{k} \sum_{s \in T_i} c_s \cdot \frac{2(1+\epsilon)^{i+1}}{\epsilon^2} + \epsilon f \cdot \sum_{i=0}^{k} \sum_{e \in A_i} \omega(e) \cdot (1+\epsilon)^{i+1} \cdot \frac{2}{\epsilon^2} \\
&\ge \sum_{i=0}^{k} \sum_{s \in T_i} \frac{2(1+\epsilon)^{i - \mathsf{base}(s)}}{\epsilon} + f \cdot \sum_{i=0}^{k} \sum_{e \in A_i} (1+\epsilon)^{-i} \cdot (1+\epsilon)^{i+1} \cdot \frac{2}{\epsilon} \\
&\ge \sum_{s \in T_{\le k}} 2\max\{\mathsf{lev}(s) - \mathsf{base}(s) + 1, 0\} + \frac{2f}{\epsilon}|A_{\le k}|,
\end{aligned}$$

where the first inequality follows from Equation (4) and Equation (5); the last two inequalities hold, since $c_s \ge (1+\epsilon)^{-\mathsf{base}(s)-1}$ and $(1+\epsilon)^x \ge 1 + \epsilon x \ge \epsilon(1+x)$. □

First, let us analyze the runtime cost of the steps inside $\mathsf{Rebuild}(k)$, together with finding k, excluding the steps inside the calls to $\mathsf{HandleDet}(e)$ and $\mathsf{HandleRand}(e)$. By Observation 2.1, the runtime cost of finding k, implicitly zeroing the sets up to level k, and initializing set E is

$$O\left(\frac{\log C}{\epsilon} + |T_{\leq k} \setminus T_{\leq \lceil \log_{1+\epsilon} C \rceil + 1}| + |E_{\leq k}|\right).$$

Processing clean elements takes $O\left(f|E|\right)$ time; then we spend $O\left(|D|\right) = O\left(|E_{\leq k}|\right)$ time on processing dirty elements. The time the algorithm spends on the post-processing steps before invoking $\mathsf{WaterFilling}$ is $O\left(f(|E|+|E'|) + |S| + f|\widehat{E}| + |\widehat{E}| + |\widehat{S}|\right) = O\left(f(|E|+|E'|)\right)$, since $\widehat{E} \subseteq E \cup E'$, $\widehat{S} \subseteq S$ and $|S| \leq f(|E|+|E'|)$. Finally, invoking $\mathsf{WaterFilling}(\widehat{k}, \widehat{S}, \widehat{E})$ takes time $O\left(f|\widehat{E}| + \widehat{k}\right) = O\left(f|\widehat{E}| + \frac{|\widehat{E}|}{\epsilon} + \frac{1}{\epsilon^2} + \frac{\log C}{\epsilon}\right) = O\left(f|\widehat{E}| + \frac{1}{\epsilon^2} + \frac{\log C}{\epsilon}\right)$, according to Lemma 3.1 and since $f > \frac{\log C}{\epsilon}$ and $\widehat{k} \leq \left\lceil \log_{1+\epsilon} \frac{2C \cdot |\widehat{E}|}{\epsilon} \right\rceil$.

Therefore, the total runtime cost is

$$O\left(\frac{1}{\epsilon^2} + \frac{\log C}{\epsilon} + \left|T_{\leq k} \setminus T_{\leq \lceil \log_{1+\epsilon} C \rceil + 1}\right| + |E_{\leq k}| + f(|E|+|E'|)\right).$$

According to Lemma 4.2, the runtime cost $O\left(|T_{\leq k} \setminus T_{\leq \lceil \log_{1+\epsilon} C \rceil + 1}|\right)$ can be charged to the elimination of down potential $\Phi_{\text{down}}(T_{\leq k})$, since the first term is at least:

$$\begin{aligned}\sum_{s \in T_{\leq k}} \max\{\mathsf{lev}(s) - \mathsf{base}(s) + 1, 0\} &\geq \sum_{s \in T_{\leq k} \setminus T_{\lceil \log_{1+\epsilon} C \rceil + 1}} \max\{\mathsf{lev}(s) - \mathsf{base}(s) + 1, 0\} \\ &\geq \sum_{s \in T_{\leq k} \setminus T_{\lceil \log_{1+\epsilon} C \rceil + 1}} 1 = \left|T_{\leq k} \setminus T_{\leq \lceil \log_{1+\epsilon} C \rceil + 1}\right|\end{aligned}$$

Observe that every element in $E \cup E'$ either was active before the call to $\mathsf{Rebuild}(k)$, or was passive but became active during it. The runtime cost $O(f)$ induced by each active element can be charged to the elimination of $\Phi_{\text{down}}(T_{\leq k})$, while the cost induced by passive elements can be charged to the clearance of their passive potentials. Using the same argument, we can cover the runtime cost $O(1)$ spent on processing each dirty element, for which we do not call $\mathsf{HandleDet}(e)$ or $\mathsf{HandleRand}(e)$ (because it became active during a previous iteration). For other dirty elements, the $O(1)$ runtime cost can be transferred to the respective call to $\mathsf{HandleDet}(e)$ or $\mathsf{HandleRand}(e)$.

The runtime cost $O(\frac{1}{\epsilon^2} + \frac{\log C}{\epsilon})$ can be charged to the elimination of $\Phi_{\text{clean}}(T_{\leq k})$. Since $\phi_{\leq k} > 0$, then $\phi(s) > 0$ for at least one set s with $\mathsf{lev}(s) \leq k$. Therefore $\Phi_{\text{clean}}(T_{\leq k})$ has decreased by at least $\frac{1}{\epsilon^2} + \frac{\log C}{\epsilon}$.

In the meantime, since $\mathsf{WaterFilling}$ might decrease the levels of the sets in $\widehat{S}$, we might have increased the lift potentials of $s \in \widehat{S}$ by at most $\max\{\mathsf{lev}^{\text{old}}(s) - \mathsf{base}(s) + 1, 0\}$. Fortunately, such potential increases can also be paid for by the first term of $\Phi_{\text{down}}(T_{\leq k})$ from Lemma 4.2. As for Φ_{up}, it does not increase during that steps, since for every set s for which $\mathsf{lev}(s)$ or $\omega(s)$ were changed, either $s \in \widehat{S}$, or we have $\mathsf{lev}(s) = 0$ as a result. In the former case, we have $\omega(s) < c_s$ after the call to $\mathsf{WaterFilling}(\widehat{k}, \widehat{S}, \widehat{E})$ by Lemma 3.1. In the latter case, it was either because $s \in S$ and slack, or because it was affected by the implicit zeroing, in which case we have $\omega(s) < c_s$, since Property 3.4 holds after processing dirty elements.

It remains to analyze the runtime cost of the calls to $\mathsf{HandleDet}(e)$ and $\mathsf{HandleRand}(e)$. We will charge that costs to the insertion of e.

Deterministic rebuilding. The runtime cost of $\mathsf{HandleDet}(e)$ is $O(f)$. As for the potential increase, notice that only sets below level $k+1$ can increase their level. Any such set s has $\phi(s) = 0$ by Property 3.4. Therefore, Φ_{down} and Φ_{clean} do not increase. Φ_{up} does not increase as well. Observe that if there is a tight set $s \ni e$, then no weight of a set has changed, and if $\mathsf{lev}(s)$ is changed, then $\omega(s) < c_s$ by Property 3.4. Otherwise, if all sets are slack, then the call to $\mathsf{DeclLev}(e)$ could increase the weights of the sets; however, it makes sure that $\omega(s) < c_s$ for all $s \ni e$ as a result. Φ_{lift} does not increase, since sets can only increase their levels.

Next, we will show that the gap $\mathsf{ilev}(e) - \mathsf{zlev}(e)$ decreases by at least one each time we call $\mathsf{HandleRand}(e)$.

Claim 4.5. *The call to* $\mathsf{HandleDet}(e)$ *decreases the gap* $\mathsf{ilev}(e) - \mathsf{zlev}(e)$ *by at least one, and the size of the gap becomes at most* $\mathsf{ilev}^{old}(e) - k - 1$.

Proof. By Corollary 3.3, we have $\mathsf{zlev}(e) = \mathsf{zlev}^{\text{old}}(e) \le k$, and $\mathsf{ilev}(e) = \mathsf{ilev}^{\text{old}}(e)$ before the call. Observe that during the call, $\mathsf{zlev}(e)$ becomes at least $k+1$, and $\mathsf{ilev}(e)$ does not increase. Therefore, the gap $\mathsf{zlev}(e) - \mathsf{ilev}(e)$ decreases by at least one and becomes at most $\mathsf{ilev}^{\text{old}}(e) - k - 1$. □

Notice that the gap $\mathsf{ilev}(e) - \mathsf{zlev}(e)$ is affected only by calls to $\mathsf{HandleDet}(e)$. This follows from Observations 3.6 and 3.7 and Claim 3.16. Observe that calls to Delete do not affect the gap either. Calls to Insert do not affect the gap by Observation 3.5(2). By Claim 4.5, the gap decreases by at least one after each call to $\mathsf{HandleDet}(e)$. Since after e is inserted, we have $\mathsf{ilev}(e) \le \mathsf{zlev}(e) + \lceil \log_{1+\epsilon} \max\{f, \frac{2C}{\epsilon}\} \rceil$, there are at most $\lceil \log_{1+\epsilon} \max\{f, \frac{2C}{\epsilon}\} \rceil$ instances of $\mathsf{HandleDet}(e)$. As we have shown, the amortized runtime cost of one instance of $\mathsf{HandleDet}(e)$ is $O(f)$; therefore, the total runtime spent on the calls to $\mathsf{HandleDet}(e)$ is

$$O\left(\frac{f \log f}{\epsilon} + \frac{f \log C}{\epsilon} + \frac{f}{\epsilon^2}\right).$$

Randomized rebuilding. Consider the call to $\mathsf{HandleRand}(e)$. If the random sampling step succeeds, then the runtime cost is $O(f/(5\log)^{(\eta)}_{1+\epsilon} f)$; otherwise, it is $O(f)$ (excluding the call to $\mathsf{FixLevel}$). Notice that if the random sampling step fails, and we enter the cases for $|\widehat{F}| = 0$ or $|\widehat{F}| > \left((5\log)^{(\eta)}_{1+\epsilon} f\right)^2$, the potential does not increase, by the same argument as for the deterministic version. The same holds for the case, when the random sampling succeeds. Next, consider the case when the random sampling fails, and we enter the case where we call $\mathsf{FixLevel}$. By the same argument, the steps before the call do not increase the potential. We bound the amortized runtime cost of calling $\mathsf{FixLevel}(e, l)$ by the following lemma.

Lemma 4.3. *The amortized runtime cost of the call to* $\mathsf{FixLevel}(e, l)$ *during* $\mathsf{HandleRand}(e)$ *is* $O\left(\frac{f}{\epsilon^2} + \frac{f \log C}{\epsilon}\right)$ *if* $\eta < 3$, *and* $O\left(\frac{f}{\epsilon^2} + \frac{\log C}{\epsilon} \log^2 f\right)$ *otherwise.*

Proof. Consider F defined in the call to $\mathsf{FixLevel}(e, l)$. By definition, F is a collection of all sets $s \ni e$ such that $\omega(s) - \omega(e) + (1+\epsilon)^{-\mathsf{zlev}(e) - d} > c_s$. Since $\mathsf{zlev}(e)$ becomes at least $k+1$ right before the call to $\mathsf{FixLevel}(e, l)$, we have $(1+\epsilon)^{-\mathsf{zlev}(e) - \widehat{d}} \le \delta$, and hence $F \subseteq \widehat{F}$. Therefore, according to Theorem 4.1, the amortized runtime cost of the call to $\mathsf{FixLevel}$ can be bounded by

$$3|\widehat{F}| \cdot \left(\frac{3f}{\epsilon^3} + \frac{f \log C}{\epsilon^2}\right) \cdot (1+\epsilon)^{-\widehat{d}+1} + |\widehat{F}| \cdot \left(\frac{1}{\epsilon^2} + \frac{\log C}{\epsilon}\right) + O(f) \tag{6}$$

Since $|\widehat{F}| \leq \left((5\log)^{(\eta)}_{1+\epsilon} f\right)^2$ and $\widehat{d} = \left\lceil \log_{1+\epsilon} \max\left\{\left((5\log)^{(\eta)}_{1+\epsilon} f\right)^4, \left(\frac{2C}{\epsilon}\right)^2\right\}\right\rceil$, we can bound the first term of Equation (6):

$$3|\widehat{F}| \cdot \left(\frac{3f}{\epsilon^3} + \frac{f\log C}{\epsilon^2}\right) \cdot (1+\epsilon)^{-\widehat{d}+1} \leq 6\left(\frac{3f}{\epsilon^3} + \frac{f\log C}{\epsilon^2}\right) / \frac{2C}{\epsilon} < 12f/\epsilon^2$$

By Lemma 2.1, $|\widehat{F}| \leq \left(\epsilon/5 \cdot 5\log_{1+\epsilon} f\right)^2 = O(\log^2 f)$ when $\eta \geq 3$. Therefore, we can bound the second term of Equation (6):

$$|\widehat{F}| \cdot \left(\frac{1}{\epsilon^2} + \frac{\log C}{\epsilon}\right) = O(\log^2 f) \cdot \left(\frac{1}{\epsilon^2} + \frac{\log C}{\epsilon}\right) = O\left(\frac{f}{\epsilon^2}\right) + O\left(\frac{\log C}{\epsilon}\log^2 f\right)$$

When $\eta < 3$, we can use a trivial bound $|\widehat{F}| \leq f$, and bound the second term:

$$|\widehat{F}| \cdot \left(\frac{1}{\epsilon^2} + \frac{\log C}{\epsilon}\right) \leq \frac{f}{\epsilon^2} + \frac{f\log C}{\epsilon}$$

□

For an element e, consider all the instances of $\mathsf{HandleRand}(e)$. We will again argue that the gap $\mathsf{ilev}(e) - \mathsf{zlev}(e)$ decreases after each such call. By the same argument as for the deterministic version, the gap is affected by these calls only.

First, we bound the total runtime of the instances of $\mathsf{HandleRand}(e)$ where we fall back to $\mathsf{HandleDet}(e)$. Consider the first such an instance of $\mathsf{HandleRand}(e)$. It was called from within $\mathsf{Rebuild}(k)$ for some k. Observe that we call $\mathsf{HandleDet}(e)$ whenever $\mathsf{ilev}(e) - k - 1 \leq \frac{200}{\epsilon^2}$ or $\mathsf{ilev}(e) - \mathsf{zlev}(e) \leq 1 + 2\log_{1+\epsilon}\frac{2C}{\epsilon}$. If the first inequality holds, then, by Claim 4.5, the gap $\mathsf{ilev}(e) - \mathsf{zlev}(e)$ becomes at most $\mathsf{ilev}(e) - k - 1 \leq \frac{200}{\epsilon^2}$ as a result. The gap $\mathsf{ilev}(e) - \mathsf{zlev}(e)$ strictly decreases after each call to $\mathsf{HandleDet}(e)$, and steps outside it do not increase the gap. Therefore, there are at most $O(\max\{1 + 2\log_{1+\epsilon}\frac{2C}{\epsilon}, 1 + \frac{200}{\epsilon^2}\}) = O(\frac{\log C}{\epsilon} + \frac{1}{\epsilon^2})$ such instances of $\mathsf{HandleRand}(e)$, and each such an instance takes time $O(f)$. Therefore, the total time spent on such instances is $O(\frac{f}{\epsilon^2} + \frac{f\log C}{\epsilon})$.

Next, consider all the instances of $\mathsf{HandleRand}(e)$ where we do not fall back to $\mathsf{HandleDet}(e)$ and group them by the value of η during them.

Claim 4.6. *Each call to* $\mathsf{HandleRand}(e)$ *decreases the gap* $\mathsf{ilev}(e) - \mathsf{zlev}(e)$ *by at least one, and the size of the gap becomes at most* $\mathsf{ilev}^{old}(e) - k - 1$.

Proof. If we fall back to $\mathsf{HandleDet}(e)$, then this holds by Claim 4.5. Otherwise, by Corollary 3.3, we have $\mathsf{zlev}(e) = \mathsf{zlev}^{\mathrm{old}}(e) \leq k$, and $\mathsf{ilev}(e) = \mathsf{ilev}^{\mathrm{old}}(e)$ before the call. If the random sampling step succeeds, then $\mathsf{zlev}(e)$ becomes at least $k+1$, and $\mathsf{ilev}(e)$ remains unchanged. If it fails, then we branch into three cases. If the first and the last case, $\mathsf{zlev}(e)$ becomes at least $k+1$ and $\mathsf{ilev}(e)$ does not increase. In the second case, $\mathsf{zlev}(e)$ becomes at least $k+1$ as well, and then we call to $\mathsf{FixLevel}(e, l)$, where $l \leq \mathsf{ilev}(e)$, which preserves the gap by Observation 3.5(1). □

Claim 4.7. *For any element e, the total amortized runtime spent on the instances of* $\mathsf{HandleRand}(e)$ *where we do not fall back to* $\mathsf{HandleDet}(e)$ *with the same value of η is bounded in expectation by* $O\left(\frac{f}{\epsilon^2} + \frac{f\log C}{\epsilon}\right)$ *if $\eta < 3$, and* $O\left(\frac{f}{\epsilon^2} + \frac{\log C}{\epsilon}\log^2 f\right)$ *otherwise.*

Proof. First, notice that for a fixed value of η, there are at most $(5\log)^{(\eta)}_{1+\epsilon} f$ instances of $\mathsf{HandleRand}(e)$. Indeed, consider the first among them. By the definition of η, at the beginning of it, we have

$\mathsf{ilev}(e) - k - 1 \le (5\log)_{1+\epsilon}^{(\eta)} f$. By Claim 4.6, the gap $\mathsf{ilev}(e) - \mathsf{zlev}(e)$ becomes at most $\mathsf{ilev}(e) - k - 1$ after it.

The runtime cost of the random sampling step is $O(f/(5\log)_{1+\epsilon}^{(\eta)} f)$, so the total runtime spent on it for all such instances is $O(f)$. For the rest, assume that the random sampling step fails. In that case, we spend time $O(f)$ on computing $\widehat{F}$. Next, the analysis splits into cases, depending on the size of $\widehat{F}$.

If $0 < |\widehat{F}| \le \left((5\log)_{1+\epsilon}^{(\eta)} f\right)^2$, then after the call to $\mathsf{FixLevel}(e, l)$, the gap $\mathsf{ilev}(e) - \mathsf{zlev}(e)$ is at most

$$\left\lceil \log_{1+\epsilon} \max\left\{ \left((5\log)_{1+\epsilon}^{(\eta)} f\right)^4, \left(\frac{2C}{\epsilon}\right)^2 \right\} \right\rceil \le \max\left\{ (5\log)_{1+\epsilon}^{(\eta+1)} f, 1 + 2\log_{1+\epsilon} \frac{2C}{\epsilon} \right\} \tag{7}$$

In that case, we use Lemma 4.3 to bound the runtime cost of $\mathsf{FixLevel}(e, l)$.

If $\widehat{F} = \emptyset$, then we call $\mathsf{DeclLev}(e)$. Due to that call, either e becomes active, or $\mathsf{ilev}(e)$ becomes at most $\lceil -\log_{1+\epsilon} \delta \rceil$. Recall that

$$\delta = \min\left\{ \left(\frac{1}{(5\log)_{1+\epsilon}^{(\eta)} f}\right)^4, \left(\frac{\epsilon}{2C}\right)^2 \right\} \cdot (1+\epsilon)^{-k-1}.$$

Observe that after the call, we have $\mathsf{zlev}(e) \ge k + 1$. Thus, we get the same bound on the gap as in Equation (7).

Therefore, if $|\widehat{F}| \le \left((5\log)_{1+\epsilon}^{(\eta)} f\right)^2$, then either the gap $\mathsf{ilev}(e) - \mathsf{zlev}(e)$ becomes at most $(5\log)_{1+\epsilon}^{(\eta+1)} f$, and hence in the next instance the value of η will be strictly larger, or the gap becomes at most $1 + 2\log_{1+\epsilon} \frac{2C}{\epsilon}$, in which case it is the last instance of $\mathsf{HandleRand}(e)$ where we do not fall back to $\mathsf{HandleDet}(e)$.

Otherwise, if $|\widehat{F}| > \left((5\log)_{1+\epsilon}^{(\eta)} f\right)^2$, the algorithm spends time $O(f)$. However, the probability that the random sampling step fails would be at most

$$\left(1 - |\widehat{F}|/f\right)^{50 \cdot \left\lceil f/(5\log)_{1+\epsilon}^{(\eta)} f \right\rceil} \le \left(\frac{1}{e}\right)^{10 (5\log)_{1+\epsilon}^{(\eta)} f} \le \frac{1}{(5\log)_{1+\epsilon}^{(\eta-1)} f} \le \frac{1}{(5\log)_{1+\epsilon}^{(\eta)} f}.$$

Therefore, the expected time cost of such a call is $O(f/(5\log)_{1+\epsilon}^{(\eta)} f)$.

As we have shown in the beginning of the proof, there are at most $(5\log)_{1+\epsilon}^{(\eta)} f$ instances with such a value of η. Hence the total expected runtime is

$$O\left(\frac{f}{(5\log)_{1+\epsilon}^{(\eta)} f} \cdot (5\log)_{1+\epsilon}^{(\eta)} f \right) = O(f).$$

□

For any passive element e, we have $\mathsf{ilev}(e) \le \mathsf{zlev}(e) + \lceil \log_{1+\epsilon} \max\{f, \frac{2C}{\epsilon}\} \rceil$. Therefore, the gap $\mathsf{ilev}(e) - \mathsf{zlev}(e)$ is bounded by $\lceil \log_{1+\epsilon} f \rceil$, since we assume $f > \frac{2C}{\epsilon}$ for the randomized version. Therefore, the maximum possible value of η is bounded by $(5\log)_{1+\epsilon}^{*}(f)$. By Lemma 2.1, $(5\log)_{1+\epsilon}^{(3)}(y) \le \epsilon/5 \cdot 5\log_{1+\epsilon} y \le 2\ln y$ for large enough y; otherwise, if y is small, then $(5\log)_{1+\epsilon}^{*}(y) \le$

3. Thus $(5\log)^*_{1+\epsilon}(f) = O(\log^* f)$, and hence the total runtime the algorithm spends on instances of $\mathsf{HandleRand}(e)$ is

$$O\left(\frac{f}{\epsilon^2}\log^* f + \frac{\log C}{\epsilon}\log^2 f \log^* f + \frac{f}{\epsilon^2} + \frac{f \log C}{\epsilon}\right) = O\left(\frac{f}{\epsilon^2}\log^* f + \frac{f\log C}{\epsilon}\right).$$

We conclude our analysis with the following theorem.

Theorem 4.2. *In the case of the randomized algorithm, the amortized expected total time spent for each passive element e on the calls to* $\mathsf{HandleRand}(e)$ *is* $O\left(\frac{f}{\epsilon^2}\log^* f + \frac{f\log C}{\epsilon}\right)$.

4.5 Insertion

The algorithm spends $O(f)$ runtime before it enters the branching. Next, let us consider the branches separately.

- In the case $F = \emptyset$, the algorithm spends $O(f)$ time on finding h and $O(f)$ time on the remaining steps. For the potential increase, notice that we have $\omega(s) < c_s$ for all $s \ni e$ after the insertion of e, and hence $\Phi_{\text{up}}(s) = 0$. $\Phi_{\text{lift}}(s)$ remains unchanged, since the levels of the sets remain the same. $\Phi_{\text{down}}(s)$ and $\Phi_{\text{clean}}(s)$ remain unchanged as well, since $\phi(s)$ does not change.
- In the case $F \neq \emptyset$, we can apply Theorem 4.1 with the trivial bound $|F| \le f$ and $d = \mathsf{ilev}(e) - \mathsf{zlev}(e) \ge \log_{1+\epsilon} f$.

In both cases, since we have added a new element e to the system, $\Phi(e)$ increases by at most f. Since we charge the runtime costs associated to the calls to $\mathsf{HandleDet}(e)$ and $\mathsf{HandleRand}(e)$ to $\mathsf{Insert}(e)$, we obtain the following theorem.

Theorem 4.3. *The amortized runtime cost of* $\mathsf{Insert}(e)$ *is* $O\left(\frac{f\log f}{\epsilon} + \frac{f}{\epsilon^3} + \frac{f \log C}{\epsilon^2}\right)$ *for the deterministic algorithm, and* $O\left(\frac{f}{\epsilon^2}\log^* f + \frac{f}{\epsilon^3} + \frac{f\log C}{\epsilon^2}\right)$ *for the randomized algorithm.*

4.6 Total runtime

To conclude the proof of Theorem 1.1, let $\lambda \in \{\lambda_{\text{rand}}, \lambda_{\text{det}}\}$ be the upper bound on the amortized update time, where $\lambda_{\text{rand}} = \Theta\left(\frac{f}{\epsilon^2}\log^* f + \frac{f}{\epsilon^3} + \frac{f}{\epsilon^2}\log C\right)$, and $\lambda_{\text{det}} = \Theta\left(\frac{1}{\epsilon} f\log f + \frac{f}{\epsilon^3} + \frac{f\log C}{\epsilon^2}\right)$ for the deterministic algorithm. Let Γ be the total number of element updates. Without loss of generality, we can assume that at the end of the update sequence, we have $\mathcal{U} = \emptyset$; otherwise, we can add Γ artificial deletions which does not change the asymptotic runtime bound.

- **Preprocessing.** Initially $\mathcal{U} = \emptyset$, and all sets are slack and on level 0.

  If the algorithm is deterministic, then for each $0 \le i \le L$, initialize pointers to (currently empty) sets $S_i, T_i, E_i, A_i(s), P_i(s)$ and store them in a random-accessible array. Add each set to S_0. This takes $O(\frac{1}{\epsilon} m \log(Cn))$ time and space.

  If the algorithm is randomized, then initialize randomized dynamic hash tables to store pointers to (currently empty) sets $S_i, T_i, E_i, A_i(s), P_i(s)$ [DKM+94], and add each set to S_0. This takes time $O(m)$.

- **Updates.** As we have proved in previous subsections, for each update, the amortized update time is bounded as $\Delta\Phi + \text{runtime} \le \lambda$.

Let Φ^{init} be the total potential at the beginning, and let Φ^{end} be the total potential at the end. Taking the summation of the preprocessing procedure and all updates, the total update time is bounded asymptotically by $\Gamma \cdot \lambda + \Phi^{\text{init}} - \Phi^{\text{end}}$. Since $\mathcal{U}$ is empty both at the beginning and at the end, we have $\Phi^{\text{init}} = \Phi^{\text{end}}$, which finalizes the proof.

References

[AAG$^+$19] Amir Abboud, Raghavendra Addanki, Fabrizio Grandoni, Debmalya Panigrahi, and Barna Saha. Dynamic set cover: improved algorithms and lower bounds. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 114–125, 2019.

[AS21] Sepehr Assadi and Shay Solomon. Fully dynamic set cover via hypergraph maximal matching: An optimal approximation through a local approach. 204:8:1–8:18, 2021.

[BCH17] Sayan Bhattacharya, Deeparnab Chakrabarty, and Monika Henzinger. Deterministic fully dynamic approximate vertex cover and fractional matching in $O(1)$ amortized update time. In *Integer Programming and Combinatorial Optimization: 19th International Conference, IPCO 2017, Waterloo, ON, Canada, June 26-28, 2017, Proceedings*, pages 86–98. Springer, 2017.

[BCPS23] Sayan Bhattacharya, Martín Costa, Nadav Panski, and Shay Solomon. Nibbling at long cycles: Dynamic (and static) edge coloring in optimal time. *CoRR (to appear at SODA'24)*, abs/2311.03267, 2023.

[BGK$^+$22] Sayan Bhattacharya, Fabrizio Grandoni, Janardhan Kulkarni, Quanquan C. Liu, and Shay Solomon. Fully dynamic $(\Delta +1)$-coloring in $O(1)$ update time. *ACM Trans. Algorithms*, 18(2):10:1–10:25, 2022.

[BGM17] Sayan Bhattacharya, Manoj Gupta, and Divyarthi Mohan. Improved algorithm for dynamic b-matching. In *25th Annual European Symposium on Algorithms (ESA)*, volume 87 of *LIPIcs*, pages 15:1–15:13, 2017.

[BHI15] Sayan Bhattacharya, Monika Henzinger, and Giuseppe F Italiano. Design of dynamic algorithms via primal-dual method. In *Automata, Languages, and Programming: 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, pages 206–218. Springer, 2015.

[BHN19] Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. A new deterministic algorithm for dynamic set cover. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 406–423. IEEE, 2019.

[BHNW21] Sayan Bhattacharya, Monika Henzinger, Danupon Nanongkai, and Xiaowei Wu. Dynamic set cover: Improved amortized and worst-case update time. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2537–2549. SIAM, 2021.

[BK19] Sayan Bhattacharya and Janardhan Kulkarni. Deterministically Maintaining a $(2+\epsilon)$-Approximate Minimum Vertex Cover in $O(1/\epsilon^2)$ Amortized Update Time. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1872–1885. SIAM, 2019.

[DKM$^+$94] Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer Auf Der Heide, Hans Rohnert, and Robert E Tarjan. Dynamic perfect hashing: Upper and lower bounds. *SIAM Journal on Computing*, 23(4):738–761, 1994.

[DS14] Irit Dinur and David Steurer. Analytical approach to parallel repetition. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 624–633, 2014.

[GKKP17] Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Debmalya Panigrahi. Online and dynamic algorithms for set cover. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 537–550, 2017.

[HP20] Monika Henzinger and Pan Peng. Constant-time dynamic $(\Delta{+}1)$-coloring. In *37th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 154 of *LIPIcs*, pages 53:1–53:18, 2020.

[KR08] Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within $2-\varepsilon$. *Journal of Computer and System Sciences*, 74(3):335–349, 2008.

[PD06] Mihai Patrascu and Erik D Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM Journal on Computing*, 35(4):932–963, 2006.

[PS16] David Peleg and Shay Solomon. Dynamic $(1+\epsilon)$-approximate matchings: A density-sensitive approach. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 712–729, 2016.

[PT11] Mihai Pătraşcu and Mikkel Thorup. Don't rush into a union: take time to find your roots. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 559–568, 2011.

[Sol16] Shay Solomon. Fully dynamic maximal matching in constant update time. In *IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 325–334, 2016.

[SU23] Shay Solomon and Amitai Uzrad. Dynamic $((1+\epsilon)\ln n)$-Approximation Algorithms for Minimum Set Cover and Dominating Set. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 1187–1200, 2023.

[SW18] Shay Solomon and Nicole Wein. Improved dynamic graph coloring. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms (ESA)*, volume 112 of *LIPIcs*, pages 72:1–72:16, 2018.

[WS11] David P Williamson and David B Shmoys. *The design of approximation algorithms.* Cambridge university press, 2011.

Notation	**Definition**
$\omega(e)$	The weight of element $e \in \mathcal{U}$. $\omega(e) = (1+\epsilon)^{-\mathsf{ilev}(e)}$.
$\omega(s)$	The total weight of set $s \in \mathcal{S}$. $\omega(s) = \sum_{e \in s} \omega(e)$.
$\phi(s)$	The dead weight of set $s \in \mathcal{S}$.
ϕ	The total dead weight. $\phi = \sum_{s \in \mathcal{S}} \phi(s)$.
ϕ_i	The total dead weight of sets on level i. $\phi_i = \sum_{s, \mathsf{lev}(s)=i} \phi(s)$.
$\phi_{\leq i}$	The total dead weight of sets on level i and below. $\phi_{\leq i} = \sum_{s, \mathsf{lev}(s) \leq i} \phi(s)$.
$\omega^*(s)$	The composite weight of $s \in \mathcal{S}$. $\omega^*(s) = \omega(s) + \phi(s)$.
$\omega(s, i)$	The weight of set $s \in \mathcal{S}$ at level i. It is the weight of s if it were raised to level i. $\omega(s,i) = \sum_{e \in s} \min \left\{ \omega(e), (1+\epsilon)^{-\max\{i, \max_{t \mid e \in t \neq s} \mathsf{lev}(t)\}} \right\}$.
L	The maximum level of a set, i.e. each set is assigned a level $\mathsf{lev}(s) \in [L]$. $L = \lceil \log_{1+\epsilon}(Cn) \rceil + 1$.
Tight set	A set $s \in \mathcal{S}$ is tight if $\omega^*(s) \geq \frac{c_s}{1+\epsilon}$.
Slack set	A set $s \in \mathcal{S}$ which is not tight, i.e. $\omega^*(s) < \frac{c_s}{1+\epsilon}$.
T	The collection of all tight sets.
T_i	The collection of all tight sets at level i, i.e. a collection of $s \in T$ such that $\mathsf{lev}(s) = i$
S_i	The collection of all sets at level i, i.e. a collection of $s \in \mathcal{S}$ such that $\mathsf{lev}(s) = i$
$\mathsf{lev}(s)$	The level of set $s \in \mathcal{S}$. $\mathsf{lev}(s) \in [L]$.
$\mathsf{lev}(e)$	The level of element $e \in \mathcal{U}$. $\mathsf{lev}(e) = \max_{s \ni e}\{\mathsf{lev}(s)\}$.
$\mathsf{zlev}(e)$	The lazy level of element $e \in \mathcal{U}$. $\mathsf{zlev}(e) \leq \mathsf{lev}(e)$.
$\mathsf{ilev}(e)$	The intrinsic level of element $e \in \mathcal{U}$. $\omega(e) = (1+\epsilon)^{-\mathsf{ilev}(e)}$ and $\mathsf{lev}(e) < \mathsf{ilev}(e) \leq \mathsf{zlev}(e) + \lceil \log_{1+\epsilon} \max\{f, \frac{2C}{\epsilon}\} \rceil$.
$\mathsf{base}(s)$	The base level of set $s \in \mathcal{S}$. $\mathsf{base}(s) = \lfloor \log_{1+\epsilon} 1/c_s \rfloor$.
Active element	If an element e is active, then the value $\mathsf{lev}(e)$ is correctly maintained (i.e. $\mathsf{ilev}(e) = \mathsf{zlev}(e) = \mathsf{lev}(e)$), and $\omega(e) = (1+\epsilon)^{-\mathsf{lev}(e)}$.
Passive element	If an element e is passive, then we have $\mathsf{zlev}(e) \leq \mathsf{lev}(e)$ and $\mathsf{ilev}(e) > \mathsf{lev}(e)$.
A_i	The set of all active element of level i (i.e. $\mathsf{lev}(e) = i$)
P_i	The set of passive elements of intrinsic level i, (i.e. $\mathsf{ilev}(e) = i$).
$A_i(s)$, $P_i(s)$	$A_i \cap s$ and $P_i \cap s$ respectively.
E_i	The set of elements e such that $\mathsf{zlev}(e) = i$.
$A_{\leq i}, P_{\leq i}, E_{\leq i}, S_{\leq i}, T_{\leq i}$	$A_{\leq i} = \bigcup_{k=0}^{i} A_k$. The rest are defined similarly.
Dirty element	During a rebuild on a level k, an element $e \in E_{\leq k}$ is called dirty, if e is passive and $\mathsf{ilev}(e) > k+1$.
Clean element	During a rebuild on a level k, an element $e \in E_{\leq k}$ is clean if it is not dirty.

Table 2: Some of the notations used in this paper.