Online and Dynamic Algorithms for Set Cover

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

Ravishankar Krishnaswamy
Microsoft Research
India
rakri@microsoft.com

Amit Kumar
IIT Delhi
India
amitk@cse.iitd.ac.in

Debmalya Panigrahi
Duke University
USA
debmalya@cs.duke.edu

ABSTRACT

In this paper, we give new results for the set cover problem in the fully dynamic model. In this model, the set of "active" elements to be covered changes over time. The goal is to maintain a near-optimal solution for the currently active elements, while making few changes in each timestep. This model is popular in both dynamic and online algorithms: in the former, the goal is to minimize the update time of the solution, while in the latter, the recourse (number of changes) is bounded. We present generic techniques for the dynamic set cover problem inspired by the classic greedy and primal-dual offline algorithms for set cover. The former leads to a competitive ratio of $O(\log n_t)$, where n_t is the number of currently active elements at timestep t, while the latter yields competitive ratios dependent on f_t, the maximum number of sets that a currently active element belongs to. We demonstrate that these techniques are useful for obtaining tight results in both settings: update time bounds and limited recourse, exhibiting algorithmic techniques common to these two parallel threads of research.

CCS CONCEPTS

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

KEYWORDS

Set cover, recourse, dynamic algorithms, competitive ratio, online algorithms, vertex cover, graph matching, hypergraph matching.

ACM Reference format:
Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Debmalya Panigrahi. 2017. Online and Dynamic Algorithms for Set Cover. In *Proceedings of 49th Annual ACM SIGACT Symposium on the Theory of Computing, Montreal, Canada, June 2017 (STOC'17),* 14 pages.
DOI: 10.1145/3055399.3055493

1 INTRODUCTION

In the (offline) set cover problem, we are given a universe U of n elements and a family $\mathcal{F}$ of m sets with non-negative costs. The goal is to find a subfamily of sets $\mathcal{S} \subseteq \mathcal{F}$ of minimum cost that covers U. Several techniques achieve an approximation factor of $\ln n$ for this problem, and we cannot achieve $(1-\varepsilon)\ln n$ in unless $P = NP$ [15, 39]. The set cover problem has been popular due to its wide applicability. However, in many applications of this problem, we want to cover some subset $A \subseteq U$ of the universe, and this set changes over time. In general, each update to A inserts or deletes an element, and now we must allow our algorithm to change the solution to restore the feasibility and approximation. We call this the *(fully-)dynamic set cover* problem.

As in all dynamic algorithms, we want to avoid recomputing the solution from scratch, and hence constrain our algorithm to make only "limited" changes. This means it cannot use the offline algorithm off-the-shelf. Two different communities—in online algorithms and in dynamic algorithms—have approached such problems in somewhat different ways. Mainly, they differ in the restrictions they place on the changes we can make after each update in the input. In online algorithms, where decisions are traditionally irrevocable, the dynamic version of the model allows us to change a small number of past decisions while maintaining a good competitive ratio. The number of changes at each step is called its *recourse*. In the context of set cover, at each update, the algorithm is allowed to change a limited number of sets in the solution. In contrast, in dynamic algorithms, the parameter of interest is the running time to implement this change. This running time is usually called the *update time*.

Note the difference between the models: the online model ostensibly does not care about run-times and places only an information-theoretic restriction, whereas the dynamic model places a stronger, computational restriction. Hence, a bound on the update time automatically implies the same bound on recourse, but not the other way around. In most cases, however, this observation cannot be used directly because the recourse bounds one desires (and can achieve) are much smaller than the bounds on update time. This is perhaps the reason that research in these two domains has progressed largely independent of each other; one exception is [28] for the Steiner tree problem. Indeed, for set cover, online algorithms researchers have focused on obtaining (poly)logarithmic approximations in the insert-only model with no recourse [2]. In dynamic algorithms, this problem has been studied as an extension of dynamic matching/vertex cover, and the current results give approximation factors

STOC'17, Montreal, Canada
© 2017 ACM. 978-1-4503-4528-6/17/06…$15.00
DOI: 10.1145/3055399.3055493

that depend on f, the maximum element frequency.[1] In this paper, we bring these two strands of research closer together, both in terms of algorithmic techniques and achieved results. We give new results, and improve existing ones, in both domains, and develop a general framework that should prove useful for other problems in both domains.

1.1 Our Results

Following the literature, our recourse/update time bounds, except where explicitly stated, are *amortized bounds.* In other words, the recourse/update time in a single input step can be higher than the given bounds, but the average over any prefix of input steps obeys these bounds.

Let n_t denote the number of elements that need to be covered at time t, and n denote the maximum value of n_t, i.e., $n = \max_t n_t$. Similarly, let f_t be the maximum frequency of elements active at time t, and $f = \max_t f_t$. Our main results for the setting of online algorithms with recourse are:

Theorem 1.1 (Recourse). *For the fully dynamic set-cover problem, there is a*

(a) polynomial time $O(\log n_t)$-competitive and $O(1)$-recourse per input step deterministic algorithm, and
(b) polynomial time $O(f_t)$-competitive and $O(1)$-recourse per input step randomized algorithm.

Moreover, these can be combined to give a single algorithm that achieves $O(\min(f_t, \log n_t))$-competitiveness with $O(1)$ recourse.

The only prior result known for online set cover is the seminal $O(\log m \log n)$ competitive algorithm for the classical (insertion-only, no recourse) online model with m sets and n elements [2]. Moreover, there is a matching $\Omega(\log m \log n)$ lower bound, assuming $NP \not\subseteq BPP$ [27]. Hence, our result shows that this lower bound breaks down even if we allow only (amortized) constant changes in the solution per input step; moreover, we are able to remove the dependence on the number of sets m altogether from the competitive ratio. This is particularly useful because many problems in combinatorial optimization can be modeled as set cover using exponentially many sets, e.g., the standard way to model non-metric facility location as set cover has exponentially many sets. Observe that our competitive ratio is asymptotically tight assuming $P \neq NP$.

Next, we give our results for dynamic set cover in the update-time setting:

Theorem 1.2 (Update Time). *For the fully-dynamic set cover problem, there is a*

(a) polynomial time $O(\log n_t)$-competitive deterministic algorithm with $O(f \log n)$ update time, and
(b) polynomial time $O(f_t^3)$-competitive deterministic algorithm with $O(f^2)$ update time.

Moreover, we can combine these into a single algorithm that achieves $O(\min(f_t^3, \log n_t))$-competitive algorithm with $O(f(f + \log n))$ update time.

[1]The *frequency* of an element is the number of sets it belongs to; hence $f = 2$ for vertex cover instances, where the elements are edges and the sets are vertices, since an edge/element belongs to two vertices/sets.

To the best of our knowledge, part (a) above is the first result to obtain a competitive ratio independent of the maximum element frequency f, with guarantees on the update time. Indeed, the current-best existing result for dynamic set cover obtains a competitive ratio of $O(f^2)$ with $O(f \log n)$ update time [10]. Our result of part (b) is able to remove the dependence on $\log n$ in the update time, although the competitive ratio worsens from $O(f^2)$ to $O(f^3)$.

For the case of vertex cover where $f = 2$, we obtain an $O(1)$-competitive deterministic algorithm for the *dynamic (weighted) vertex cover* problem, with $O(1)$ update time. The previous-best deterministic algorithm was $(2 + \varepsilon)$-competitive with $O(\varepsilon^{-2} \log n)$ update time [11]. For randomized algorithms, Solomon [37] recently gave an algorithm to maintain maximal matchings with $O(1)$ update time; this gives a 2-competitive algorithm for (unweighted) vertex cover. Our algorithm can be seen as giving (a) a deterministic counterpart to Solomon's result for vertex cover (see [11] for a discussion of the challenges in getting a deterministic algorithm that matches the randomized results in this context), and (b) extending it from unweighted vertex cover to weighted set cover (a.k.a. hypergraph vertex cover).

Non-Amortized Results. Given these amortized bounds on recourse and update time, one may wonder about non-amortized bounds. We give a partial answer to this question, for the case of recourse bounds. We show that if the algorithm were given exponential time, the recourse bound in Theorem 1.1(a) can be made non-amortized. We leave the problem of obtaining polynomial-time algorithms with non-amortized guarantees for these models an interesting open question.

Theorem 1.3. *There is a $O(\log n)$-competitive deterministic algorithm for the dynamic set cover problem, with $O(1)$ non-amortized recourse per input step. If all sets have the same cost (unweighted set cover), then the competitive ratio improves to $O(1)$. This algorithm runs in exponential time.*

1.2 Our Techniques

We grouped our results based on recourse or update time, but the techniques give a different natural grouping. Indeed, the $O(\log n)$-competitive results are obtained by greedy-based algorithms, and the frequency-based results rely on primal-dual techniques.

Greedy Algorithms. The results presented in Theorems 1.1(a) and 1.2(a) are based on a novel *dynamic greedy* framework for the set cover problem. A barrier to making greedy algorithms dynamic is their sequential nature, and element inserts/deletes can play havoc with this. However, we abstract out the intrinsic properties that the analysis of greedy uses, and show these properties can be maintained fast, and with small amounts of recourse. Our algorithm does simple "local" moves, and the analysis uses a fractional token scheme to ensure constant recourse and small run-time.

In more detail, the greedy algorithm chooses sets one-by-one, minimizing the *incremental cost-per-element* covered at each step. The analysis then shows that the number of elements covered at incremental-costs $\approx 2^i(\mathsf{Opt}/n)$ is at most $n/2^i$, which will easily give us the desired $O(\log n)$ bound for approximation factor. So our abstraction in the dynamic setting is then the following: *try to cover as many elements at low incremental costs!* Indeed, if at some time we can find a set S along with some elements $X \subseteq S$ such that the *new incremental-cost* $c_S/|X|$ is at most half the current incremental-cost for each element in X, then we add such a set to the solution, and repeat. Of course, changes beget other changes—if set S now covers element e which was covered by T, the cost of T is now shared among fewer elements, causing their incremental-costs to increase. This can cause cascades of changes. We nevertheless show that this works using a delicate (fractional) token-based argument: each new element brings L tokens with it, and any time a new set is bought, we expend one token. If we make sure that the total tokens remaining is always non-negative, we'd be done. Proving that this can be done with $L = O(1)$ tokens per element is the basis of Theorem 1.1(a). The proof of Theorem 1.2(a) is similar, albeit we need to now argue about running times.

Primal-Dual Algorithms. The results of Theorems 1.1(b) and 1.2(b) are inspired by *primal-dual* algorithms. Offline, we raise some duals until some set constraint becomes tight (so that the set is paid for): we then pick this set in our solution. Elements pay for at most f sets, hence the f-approximation. But if elements disappear and take their dual with them, tight constraints can become slack, and we must take recourse action.

For Theorem 1.2(b), let us use vertex cover to illustrate our ideas. Inspired by previous work [6, 11, 37], a natural idea is to place each vertex v at some *integer level* $\ell(v)$, and set the dual y_e for an edge (u, v) to be $1/2^{\max(\ell(u),\ell(v))}$. To define the solution, then we include all vertices whose dual constraints are approximately tight i.e., $\mathcal{S} = \{v : 1/2 \leq \sum_{e\sim v} y_e \leq 2\}$. The cost is automatically bounded because we only include approximately-tight vertices, and the dual solution is approximately feasible. Now, when edges arrive/depart, these bounds might be violated and then we move the vertices up or down accordingly. To bound the updates, we again use a token scheme, where tokens are added by arrivals/deletions, and are spent on updates. When vertices move, they also transfer some tokens to their neighbors on incident edges, which then use up these tokens when they move, and so on. A key idea in our analysis is to use asymmetry in token transfer—vertices moving down transfer more tokens to their neighbors than vertices moving up. Using this idea, we obtain a *deterministic* constant-competitive vertex cover algorithm with constant update time. These ideas extend to the set cover setting, giving $O(f^3)$-competitive $O(f^2)$ update time algorithms; this is the first result for dynamic set cover with update time independent of n.

Finally, Theorem 1.1(b): getting f-competitiveness and $O(f)$-recourse is easy, but making this $O(1)$-recourse is the challenge. We show that an elegant randomized algorithm due to Pitt [33]—for an uncovered element e, pick a random set covering it from the "harmonic" distribution, and repeat—can be made dynamic with $O(1)$-recourse. We then use dual-fitting to bound the cost.

Combiner Algorithm: We then show how we can dynamically maintain the *best-of-both* solutions with the same recourse or update-time bounds. In the offline setting, getting such an algorithm is trivial as we can simply output the solution with lower cost. However, this is not immediate in the dynamic setting as the lower-cost solution could oscillate between the two solutions we maintain; so we exploit the values of the two competitive-ratio guarantees to design our overall combiner algorithm.

1.3 Related Work

Set cover has been studied both in the offline [39] and online settings [2]; under suitable complexity-theoretic assumptions, both the $O(\log n)$ and $O(f)$-approximations are best possible.

Dynamic algorithms are a vibrant area of research; see [16] for many applications and pointers. Maintaining approximate vertex covers and approximate matchings in a dynamic setting have seen much interest in recent years, starting with [31]; see [6–8, 10–12, 23, 30]. For the exact version of the problem [35] gives polynomial update times, and logarithmic times are ruled out by recent works [1, 24, 26].

The study of online algorithms with recourse goes back at least to the dynamic Steiner tree problem in [25]: motivated by lower bounds for fully-dynamic inputs [4, 5] for online scheduling problems, [3, 17, 32, 34, 36, 38] studied models with job reassignments. Maintaining an exact matching with small recourse was studied by [13, 14, 18]; low-load assignments and flows by [22, 38], Steiner trees by [19, 21, 28, 29]. As far as we know, Łacki et al. [28] are the only previous work trying to bridge the online recourse and dynamic algorithms communities.

Recently and independently, Bhattacharya et al. [9] get a deterministic $O(1)$-competitive, $O(1)$ update-time algorithm for dynamic vertex cover.

1.4 Notation

The input is a set system $(U, \mathcal{F})$; c_S is the cost of set S. In (dynamic) set cover, the input sequence is $\boldsymbol{\sigma} = \langle \sigma_1, \sigma_2, \ldots \rangle$, where request σ_t is either $(e_t, +)$ or $(e_t, -)$. The initial *active set* $A_0 = \emptyset$. If $\sigma_t = (e_t, +)$, then $A_t \leftarrow A_{t-1} \cup \{e_t\}$; if $\sigma_t = (e_t, -)$ then $A_t \leftarrow A_{t-1} \setminus \{e_t\}$. We do not need to know either the universe U or the entire family $\mathcal{F}$ up-front. Indeed, at time t, we know only the elements seen so far, and which sets they belong to. When a new element arrives, it also reveals the sets containing it (which could include new singleton sets). We need to maintain a feasible set cover $\mathcal{S}_t \subseteq \mathcal{F}$, i.e., the sets in $\mathcal{S}_t$ must cover the set of active elements A_t. Define $n_t := |A_t|$ and $n = \max_t n_t$. The *frequency* of an element $e \in U$ is the number of sets of $\mathcal{F}$ it belongs to; let $f_t := \max_{e\in A_t}$ frequency(e) be the maximum frequency of any active element at time t. Let Opt_t be the cost of the optimal set cover for the set system $(A_t, \mathcal{F})$.

Recourse and Update Times. The *recourse* is the number of sets we add or drop from our set cover over the course of the algorithm, as a function of the length of the input sequence. An online algorithm is *α-competitive* with *r (amortized) recourse* if at every time t, the solution $\mathcal{S}_t$ has total cost at most $\alpha \cdot \mathrm{Opt}_t$, and the total recourse in the first t steps is at most $r \cdot t$. Since every dropped set must have been added at some point and we maintain at most n_t sets at time t—at most one for each active element—counting only the number of sets *dropped* until time t changes the recourse only by a constant factor. For r *worst-case recourse*, the number of sets dropped at each time-step must be at most r. In the *update-time* model, we measure the *amount of time* taken to update the solution $\mathcal{S}_{t-1}$ to $\mathcal{S}_t$. We use the terms *amortized* or *worst-case update time* similarly.

2 DYNAMIC GREEDY ALGORITHMS

In this section, we will present algorithms for the dynamic set cover problem that are inspired by the classic greedy algorithm for the offline set cover problem. First, we give a deterministic algorithm with competitive ratio $O(\log n_t)$ that uses $O(\log n)$ (amortized) update time per timestep (establishing Theorem 1.2(a)). We then outline our algorithm in the recourse model: we get $O(\log n_t)$-competitiveness with $O(1)$ (amortized) recourse (establishing Theorem 1.1(a)). These algorithms are based on a common framework where we abstract a greedylike property of set cover solutions that guarantees an $O(\log n_t)$ competitive ratio. The main technical work in these results is in using a token redistribution scheme to show the update time / recourse bounds. Tokens are added to the system in each timestep; for update time, tokens are expended in algorithmic steps for suitably updating the data structures maintained by the algorithm, whereas for recourse, tokens are used to make changes to the solution being maintained by the algorithm.

2.1 Dynamic Greedy for Update Time

We now describe our fully dynamic greedy set cover algorithm in the update-time setting. Our algorithm maintains a solution (denoted $\mathcal{S}_t$ at time t) with sets that cover the active elements A_t. In addition, it also maintains an *assignment* $\varphi_t(e)$ of each active element e to a unique set in $\mathcal{S}_t$ covering it; define $\mathsf{cov}_t(S) := \{e \mid \varphi_t(e) = S\}$ to be the set of elements assigned to S, and S is said to be *responsible for covering* the elements in $\mathsf{cov}_t(S)$. By definition, the sets $\{\mathsf{cov}_t(S) \mid S \in \mathcal{S}_t\}$ are mutually disjoint and their union is A_t. Our algorithm also uses the notions of volume and density:

Volume. At each time, our algorithm maintains for every element a notion of *volume* $\mathsf{vol}(e) > 0$.

Density. Define the *density* of a set S in $\mathcal{S}_t$ as:

$$\rho_t(S) := c(S) / \sum_{e \in \mathsf{cov}_t(S)} \mathsf{vol}(e),$$

the ratio of its cost and the volume of elements it covers.

For the algorithm below, we will set $\mathsf{vol}(e) = 1$ for all e, and hence the density of a set S is the standard notion of per-element-cost of covering elements in $\mathsf{cov}_t(S)$. Later, we will use a different notion of volume in our recourse algorithm. *In fact, our algorithms for the recourse and update-time settings are identical except for the way we define volume of elements.*

Density Levels. We also place each set in $\mathcal{S}_t$ in some *density level.* Each density level ℓ has an associated range $R_\ell := [2^\ell, 2^{\ell+10}]$ of densities. Any set S at level ℓ must have density $\rho_t(S)$ in the interval R_ℓ. We say that element e is at level ℓ if its covering set $\varphi_t(e)$ is at level ℓ.

Note that adjacent density ranges overlap across multiple levels; this range gives the necessary friction which prevents too many changes in our algorithm which helps in bounding both recourse and update time. The algorithm will dynamically make sure that each set $S \in \mathcal{S}_t$ covering a set $\mathsf{cov}_t(S)$ of elements at some time t will be placed in one of its allowed levels. For a level i, we let $A_t(i)$ denote the set of active elements which are assigned to sets at level i; i.e., $A_t(i) := \{e \in A_t \mid \varphi(e) \in \mathcal{L}_t(i)\}$. Define $\mathcal{L}_t(> i)$ and $A_t(> i)$ similarly to denote the collection of sets (and elements) in levels $i+1$ and greater.

We can now define the crucial "greedylike" concept in the dynamic setting — the notion of *stability.*

Stable Solutions. A solution $\mathcal{S}_t$ is *stable* if for every density level ℓ, there is no subset X of elements currently at level ℓ (perhaps covered by different sets) that can all be covered by some set S, such that the density of the resulting set $c(S)/\sum_{e \in X} \mathsf{vol}(e) < 2^\ell$, i.e., the set S (if added to $\mathcal{S}_t$) and these elements X would belong in a *strictly lower density level.*

Loosely, stability means there is no collection of elements that can jointly "defect" to be covered by a set S which charges them less cost-per-element. Now the dynamic algorithm just tries to maintain a stable solution. The algorithm is described in Algorithm 1. When a new element arrives, we add the cheapest set containing it to our solution. Observe that our solution may keep multiple copies of the same set (each copy would be covering disjoint subsets of elements). During each insert/delete operation, the algorithm calls the procedure Stabilitze which ensures that the solution satisfies the stability property as described above.

Algorithm 1 Dynamic($e_t, \pm$)

```
1:  if the operation $\sigma_t$ is $(e_t, +)$ then
2:      let $S$ be the cheapest set containing $e_t$
3:          (so, $\mathsf{cov}_t(S)$ is just $\{e_t\}$.)
4:      let $i$ be highest level s.t. $2^i \le c_S = \rho_t(S)$
5:      add a copy of $S$ to $\mathcal{L}_t(i)$
6:  else if the operation $\sigma_t$ is $(e_t, -)$ then
7:      remove $e_t$ from the set $S = \varphi_t(e)$ covering it;
8:      Let $i$ be the level such that $S \in \mathcal{L}_t(i)$
9:      if $S$ becomes empty then
10:         remove this copy of $S$
11:     else if current density of $S$ is $> 2^{i+10}$ then
12:         move $S$ to highest level $\ell$ s.t. $2^\ell \le \rho_t(S)$
13:     end if
14: end if
15: call Stabilize($t$)
```

Analysis Preliminaries. We now analyze the algorithm. We will show that that the cost of the solutions $\mathcal{S}_t$ are always at most a logarithmic factor off the optimal solution at time step t, and also bound the total amortized recourse. But first, we show that Stabilize terminates in finite steps.

Claim 2.1. *Algorithm* Stabilize *terminates.*

Proof. Suppose some change in the algorithm (either an arrival or departure) triggered a call to Stabilize. Now, we show that the while loop terminates after finitely many steps. Let ℓ_0 be the lowest level of any set when we call Stabilize. Consider the vector $\mathbf{v}_t = (n_t^{\ell_0}, n_t^{\ell_0+1}, n_t^{\ell_0+2}, \ldots)$, where n_t^i is the number of elements at level i in the current solution, i.e., $n_t^i = |A_t(i)|$. We claim that this vector always increases lexicographically when we perform an iteration of the outermost while loop in Algorithm Stabilize. Indeed, when pick a set S (and a corresponding index i), the new level for S is $i^\star \le i$ by definition, and hence all elements in $\mathsf{cov}_t(S)$ move down from a

Algorithm 2 Stabilize(t)

```
1:  while $\exists S \in \mathcal{F}$ and level $i$ s.t. $c_S/|S \cap A_t(i)| < 2^i$ do
2:      add copy of set $S$ to $\mathcal{S}_t$, set $\mathsf{cov}_t(S)$ to $S \cap A_t(i)$
3:      assign $S$ to highest level $i^\star$ s.t. $2^{i^\star} \le \rho_t(S)$
4:      while there exists a set $X$ at level $i$ s.t. $\mathsf{cov}_t(X) \cap \mathsf{cov}_t(S) \neq \emptyset$ do
5:          set $\mathsf{cov}_t(X) \leftarrow \mathsf{cov}_t(X) \setminus \mathsf{cov}_t(S)$
6:          Update $\rho_t(X)$ accordingly.
7:          if $\mathsf{cov}_t(X) = \emptyset$ then
8:              remove $X$ from the solution
9:          else if $\rho_t(X) > 2^{i+10}$ then
10:             Let $\ell$ be highest level s.t. $2^\ell \le \rho_t(X)$
11:             assign $X$ to level $\ell$.
12:         end if
13:     end while
14: end while
```

level $\ge i$ to a level $< i$. Some elements at level above i (covered by sets corresponding to X in the description of the algorithm) may move to levels higher than i, but none of the levels at or below i lose any elements. Hence the vector $\mathbf{v}_t$ increases lexicographically. Since none of the coordinates of this vector can be larger than n_t, this process must terminate finitely. □

The curious reader might wonder whether we need the above proof given that we would in any case need to bound the total update time of the algorithm. However, our update time analysis uses the finite termination of Stabilize and so we need this result. We next show some crucial invariants satisfied by the algorithm:

(i) The current density $\rho_t(S)$ of a set $S \in \mathcal{L}_t(i)$ satisfies $2^i \le \rho_t(S) \le 2^{i+10}$.

(ii) For each level $i \in \mathbb{Z}$, there exists no set $S \in \mathcal{F}$ such that $c_S/|S \cap A_t(i)| < 2^i$, i.e., if we include a new copy of S into the solution and cover the elements in $S \cap A_t(i)$, then all the elements in $S \cap A_t(i)$ will strictly improve their density level (from $\ge i$ to $< i$)

It is now easy to check that the algorithm maintains both the invariants.

Claim 2.2. *The solution $\mathcal{S}_t$ satisfies both the invariants at all times.*

Proof. Suppose the invariants are satisfied by $\mathcal{S}_t$. Then during the next operation, each set that is added (in both the algorithms Dynamic and Stabilize) is placed at a level that satisfies invariant (i). Moreover, the algorithm Stabilize iterates till invariant (ii) is satisfied (and always moves sets to the right levels to satisfy invariant (i)). So since the procedure terminates by Claim 2.1, we know that it satisfies both invariants at the end of time $t+1$ as well. □

Cost Analysis. Next we bound the cost of our solution. Recall that Opt_t denotes the cost of the optimal solution at time t. Let ρ_t denote Opt_t/n_t, and i_t be the index such that $2^{i_t-1} < \rho_t \le 2^{i_t}$.

Lemma 2.3. *The solution $\mathcal{S}_t$ has cost $O(\log n_t)\,\mathrm{Opt}_t$.*[2]

Proof. By Invariant (i), the density of any set at levels i_t or lower is at most $2^{i_t+10} < 2^{11} \cdot \rho_t$. So the total cost of sets in $\mathcal{L}_t(\le i_t)$ is at most $n_t \cdot 2^{11} \cdot \rho_t \le 2^{11} \cdot \mathrm{Opt}_t$.

In order to bound the cost of sets at higher levels, we first claim that for all non-negative integers $i \ge 0$, the number of elements covered by sets at levels i_t+i is at most $n_t/2^i$; i.e., $|A_t(i_t+i)| \le n_t/2^i$. Suppose not. Consider the optimal solution for A_t restricted to the elements in $A_t(i_t+i)$. It is easy to see that there exists a set S in this solution for which $c_S/|S \cap A_t(i_t+i)| \le \mathrm{Opt}_t/|A_t(i_t+i)| < 2^{i+i_t}$. But then this set S and level $i+i_t$ would violate invariant (ii), a contradiction. This proves the claim;

Now, for each such level i_t+i, where $1 \le i \le \log_2 n_t + 2$, the density of sets in the solution at this level is at most 2^{i_t+i+10} (due to invariant (i)). therefore, the total cost of sets in this level (volume times density) is at most $2^{i_t+i+10} \cdot n_t/2^i \le 2^{11} \cdot \mathrm{Opt}_t$.

Finally, the above claim also implies that all elements are covered by level $i_t + \log n_t + 2$. So summing the above cost-per-level bound over levels $i_t, \dots, i_t + \log n_t + 2$ completes the proof. □

Bounding Element-Level Changes. We bound the update time in two parts: in the first part, we (fractionally) assign *tokens* to elements, so that whenever a new set S is created in Stabilize, we will *expend* $\Omega(1)$ tokens from the system and redistribute the remaining tokens to satisfy some token invariants. In addition, we also expend $\Omega(1)$ tokens for *each element which changes its level*. Finally in the third part (in the full version of the paper [20]) we show how to maintain data structures so that the amortized time to update them can be charged to $O(f)$ times the number of tokens expended. So if the total number of tokens injected into the system is at most $O(\log n)$ per element insertion or deletion, we would get our amortized update time to be $O(f \log n)$ per element insertion or deletion. We now proceed with the details of the first part.

In order to clearly describe our token invariant, we define something called the *base level* (denoted by $b(e)$) of an element e to be the largest integer i such that $2^i \le c_{S_e}$ where S_e is the cheapest set covering e.

Claim 2.4. *Each element e is covered by a set at level at least $\mathtt{lo}_t(e) := b(e) - \lceil \log n_t \rceil - 10$ or above.*

Proof. Suppose e is covered by a set S in level i at the beginning of time t. At this time, density of S is at most 2^{i+10}. Since $c_S \ge 2^{b(e)}$, and density of S is at least $\frac{c_S}{n_t}$, it follows that $i \ge b(e) - \lceil \log n_t \rceil - 10$. During the operation at time t, e could change levels. Suppose we add a new set S containing e during Algorithm Stabilize. If S is added to level i, we know that $\rho_t(S) \le 2^{i+1}$. Since $\rho_t(S) \ge c_S/n_t \ge 2^{b(e)}/n_t$, we see that $i \ge b(e) - \lceil \log n_t \rceil - 2$. Any other operation will only move e to a higher level. This proves the claim. □

Our token distribution scheme will ensure that the following property is always satisfied:

[2] In fact we show, in the full version of this paper, that the cost is at most $O(\log \Delta_t)\,\mathrm{Opt}_t$ where Δ_t is the maximum set size $\max_{S \in \mathcal{F}} |S \cap A_t|$ at time t.

Token Invariant: Any element $e \in A_t$ covered at level i has at least $2(i - \mathtt{lo}_t(e) + 1) \geq 2$ tokens.

Token Distribution Scheme: We inject tokens when elements arrive, and redistribute them an element gets assigned to a different set or departs. Crucially, our redistribution would ensure that every time a new set is added to $\mathcal{S}_t$, a constant number of tokens are expended from the system. More formally:

(a) **Element arrival** $\sigma_t = (e_t, +)$**:** We give e_t a total of $2\lceil \log n_t \rceil + 21$ tokens, and e_t immediately *expends* one unit of token for the formation of the singleton set covering it.

(b) **Element departure** $\sigma_t = (e_t, -)$**:** Suppose e was covered by some set S at level i. Then e_t has at least 2 tokens (by Claim 2.4 and the token invariant) and so it expends one token from the system, and equally distributes at least one token to the remaining elements covered by S.

(c) Stabilize **operation:** Suppose we add a set S to the solution $\mathcal{S}_t$ at some level $i^\star$ during an iteration of the **While** loop in Stabilize. Consider any element e now covered by S but earlier covered by some set X_j at level i. Suppose e has $\tau_e \geq 2(i - \mathtt{lo}_t(e) + 1)$ tokens (since it currently satisfies the token invariant). To satisfy e's new token requirement, e retains $\tau'_e = 2(i^\star - \mathtt{lo}_t(e) + 1)$ tokens. Then, *e expends* $1/2(\tau_e - \tau'_e)$ *tokens* from the system toward the creation of this set, and equally distributes the remaining $1/2(\tau_e - \tau'_e)$ tokens among the remaining elements still covered by X, i.e., the set of elements $\mathsf{cov}_t(X) \setminus \mathsf{cov}_t(S)$. Now if a set violates invariant (i) and moves up, each remaining element in $\mathsf{cov}_t(X)$ *expends* 1 *token from the system* (which we show can be done while satisfying the new token invariant for these elements).

(d) **Phase transition:** When n_t becomes a power of two, say 2^k, and suppose the previous (different) power-of-two value of n_t was 2^{k-1}, then we give each element in A_t additional 2 tokens. This is because each element's token invariant has changed (because $\mathtt{lo}_t(e)$ has increased by 1) and it needs two extra tokens.[3]

Claim 2.5. *The total number of tokens introduced into the system is $O(\sum_t (\log n_t))$.*

Proof. New tokens are introduced only when elements arrive, and when n_t becomes a power of two. Every element arrival introduces $O(\log n_t)$ tokens with it. Moreover, when n_t becomes a power-of-two, say, 2^k and the previous power-of-two value of n_t was 2^{k-1}, then we add $2 \cdot 2^k$ tokens into the system in step (d) in the token distribution scheme. But we can *charge* these extra tokens to the 2^{k-1} arrivals which must have happened in the period when n_t increased from 2^{k-1} to 2^k. So each arrival is associated with introducing $O(\log n_t)$ tokens, which completes the proof. □

Claim 2.6. *Whenever an element moves its level up or down, one unit of token is expended from the system.*

[3]If the previous (different) power-of-two value of n_t was 2^{k+1}, then we have seen departures and the token requirement is weaker, so we don't need to give extra tokens in this case.

Proof. This is easy to see, as whenever we create a new copy of any set S, we explicitly make each element in S expend one token in step (c) above. Likewise, when a set moves up levels for violating invariant (i), we make each element expend one token from the system. □

Lemma 2.7. *The* Token Invariant *is always satisfied.*

Proof. We prove this by induction: suppose the invariant holds for all times up to $t-1$ and consider a time-step t. Firstly, we show that when n_t becomes a power-of-two, the token invariant is satisfied. Indeed, suppose we have $n_t = 2^k$, and the previous (different) power-of-two value of n_t was 2^{k-1}. Then the token invariant for each element increases by 2, which we make sure in step (d) in the token distribution scheme. On the other hand, suppose the previous power-of-two was $n_t = 2^{k+1}$, then the token invariant only becomes weaker and so is trivially satisfied.

Next, we show that the invariant holds if the operation σ^t is $(e, +)$. Indeed, we add a new set at level $b(e)$ to cover e and also give it $2\lceil \log n_t \rceil + 21$ tokens in step (a) of the token scheme. It expends one token, and the rest clearly satisfy its token invariant since it is placed at level $b(e)$. Similarly, if the operation is $(e^t, -)$, we simply delete e^t. Since it had at least 2 tokens by Claim 2.4, we are able to make it give 1 token to the remaining elements covered by the set which was covering e^t and also expend 1 token from the system.

Now we show that the operation Stabilize maintains the invariant. Suppose we find a set S during an iteration of the **While** loop of procedure Stabilize. Firstly note that the elements which are covered by the new set have requisite number of tokens since they each drop their level by at least one and we ensure in the token scheme property (c) that they retain sufficiently many tokens to satisfy their token invariant at the new level.

Now we turn our attention to the more challenging scenario when sets move up in level having lost many elements and their current density exceeds the threshold for their current level. So consider the setting when a set S moves up from level i to level $i + \ell$: we now show that each remaining element covered by S can be given $2\ell + 1$ tokens, which will be sufficient for it to satisfy the new token invariant at level $i + \ell$, and also to expend one token which we do in step (c) of the token distribution scheme. To this end, consider the first time t_{init} when a set S was added to level i, and let $\mathsf{cov}_{t_{\mathsf{init}}}(S)$ denote the initial set of elements it covers. Also consider the first time t_{fin} when it violates the invariant (i) for level i, and so it moves to some higher level, say $i + \ell$. Let the set of remaining elements at this time be $\mathsf{cov}_{t_{\mathsf{fin}}}(S)$. We now make some observations regarding these cardinalities.

Firstly, because this set moves up from level i, its current density $c_S / |\mathsf{cov}_{t_{\mathsf{fin}}}(S)|$ must be greater than 2^{i+10}. That is,

$$|\mathsf{cov}_{t_{\mathsf{fin}}}(S)| < c_S / 2^{i+10} \,. \tag{1}$$

Likewise, since it does not move up higher than $i + \ell$, its density $c_S / |\mathsf{cov}_{t_{\mathsf{fin}}}(S)|$ must be strictly less than $2^{i+\ell+1}$ (recall that when a set is relocated, it is placed in the highest level that can accommodate it). Therefore we have,

$$|\mathsf{cov}_{t_{\mathsf{fin}}}(S)| > c_S / 2^{i+\ell+1} \,. \tag{2}$$

Thus we get that $2^{i+\ell+1} > 2^{i+10}$, i.e., $\ell > 9$. Next, we note that when the set first entered level i at time t_{init}, its density $c_S/|\mathsf{cov}_{t_{\mathsf{init}}}(S)|$ must have been less than 2^{i+1} (otherwise we would have placed it at level $i+1$ or above); therefore, we have

$$|\mathsf{cov}_{t_{\mathsf{init}}}(S)| > c_S/2^{i+1}. \tag{3}$$

Similarly, when the set (and the remaining elements it covers) moves from level i to level $i+\ell$ at time t_{fin}, its density is at least $2^{i+\ell}$, i.e.,

$$|\mathsf{cov}_{t_{\mathsf{fin}}}(S)| \le c_S/2^{i+\ell}. \tag{4}$$

So from eqs. (3) and (4), we have that

$$|\mathsf{cov}_{t_{\mathsf{init}}}(S)|/|\mathsf{cov}_{t_{\mathsf{fin}}}(S)| = 2^{\ell-1}.$$

To complete the proof, note that each element in $\mathsf{cov}_{t_{\mathsf{init}}}(S) \setminus \mathsf{cov}_{t_{\mathsf{fin}}}(S)$ left 1 token with the remaining elements $\mathsf{cov}_{t_{\mathsf{fin}}}(S)$ (since it either departed from the instance or moved down at least one level due to some new sets being formed). So the tokens which the remaining elements gained is at least

$$\begin{aligned}|\mathsf{cov}_{t_{\mathsf{init}}}(S)| - |\mathsf{cov}_{t_{\mathsf{fin}}}(S)| &\ge (2^{\ell-1}-1)\cdot|\mathsf{cov}_{t_{\mathsf{fin}}}(S)| \\ &\ge (2\ell+1)\cdot|\mathsf{cov}_{t_{\mathsf{fin}}}(S)|\end{aligned}$$

The last inequality uses $\ell \ge 9$. Hence, when the set (along with the remaining elements) moves up to level $i+\ell$, each of these elements get 2ℓ extra tokens to meet their new token requirement. □

Thus, we have shown that the token invariant holds at all times. Therefore, by Claims 2.5, 2.6 and Theorems 2.3 and 2.7 we get the following theorem.

THEOREM 2.8. *The above algorithm satisfies the following properties:*

1. *At all times t, the solution $\mathcal{S}_t$ is feasible for A_t and has cost $O(\log n_t)$ times the optimal cost.*
2. *Every time a new set is formed, each element expends one token from the system.*
3. *Every time a set changes level, each covered element expends one token from the system.*
4. *The total number of tokens expended till time T is $O(\sum_{t=1}^{T}\log n_t)$.*

Finally, we need some data structures to complete the description of our fully dynamic algorithm. The crucial step to implement fast is in the Stabilize procedure where we find the best density set using elements currently covered at level i for some i. The main observation is that we will look for such sets only when we *move an element to a new level, say i*. Therefore, it should suffice to look at the f sets containing an element whenever we change its level, and see if their current densities $c_S/|A_t(i)| < 2^i$. Since each element expends one token when it changes its level, the total update time would then be within a factor f of the total number of tokens spent by the elements, if we can maintain these data structures consistently.Indeed, implementing this idea requires us to be careful with the data-structure, although we use only simple data-structures such as doubly-linked lists and arrays. One worry could be that we are keeping multiple copies of a set in our solution, and so this could play havoc with the parameter f. But in our implementation, this seemingly higher f never gets used, and we only need to refer to the distinct sets containing an element. We provide details in the full version of this paper [20].

2.2 Dynamic Greedy for Recourse

In this section, we prove Theorem 1.1(i). Our algorithmic framework is identical to the one in Section 2.1, with the crucial change being that we don't treat all elements equally . In fact, for each element $e \in A_t$, we will also define its *volume* $\mathrm{vol}(e,i)$ as a function of e and the level i it is located in. Similarly, for a set of elements $X \subseteq A_t$ and some level i, we extend the definition of volume to $\mathrm{vol}(X,i) = \sum_{e\in X}\mathrm{vol}(e,i)$. Now, for any set $S \in \mathcal{S}_t$ in the current solution which covers the elements $\mathsf{cov}_t(S)$, we define its *level-i-current density* to be $\rho_t(S,i) := c_S/\mathrm{vol}(\mathsf{cov}_t(S),i)$. This now corresponds to the ratio of cost to the *level-i volume* of elements it covers. Finally, to complete the description, we now define the volume of an element e at some level i. Consider any element e and let S_e denote the minimum cost set containing e. Then let $b(e)$ be the highest level i such that $2^i \le c_{S_e}$. Then, the volume $v(e,i)$ of element e at a level i is defined to be $2^{i-b(e)}$. Note that the volume of an element is 1 at the level $b(e)$ and decreases geometrically as it moves to lower indexed levels. For the remainder of the sub-section, $b(e)$ is said to be the *base level* for element e.

Our algorithm (described in Algorithm 3) would always try to find *stable* solutions where no local improvement in the current density of sets is possible. We formalize this with the following two invariants which our algorithm satisfies at the beginning all time-steps:

(i) The current density $\rho_t(S,i)$ of a set $S \in \mathcal{L}_t(i)$ satisfies $2^i \le \rho_t(S,i) \le 2^{i+10}$.
(ii) For each level $i \in \mathbb{Z}$, there exists no set $S \in \mathcal{F}$ such that the elements $S \cap A_t(i)$ can be brought down to level $i-1$ or below by adding a new copy of set S to $\mathcal{S}_t$. Formally, there exists no $S \in \mathcal{F}$ such that $c_S/\mathrm{vol}(S\cap A_t(i),i) < 2^i$.

Algorithm 3 RecourseAlgo($e_t, \pm$)

```
1:  if the operation $\sigma_t$ is $(e_t,+)$ then
2:      let $S$ be the cheapest set containing $e_t$
3:      let $b(e)$ be the highest level $i$ such that $2^i \le c_S$
4:      Add copy of $S$ to $\mathcal{L}_t(b(e))$, set $\mathsf{cov}_t(S) = \{e_t\}$
5:  else if the operation $\sigma_t$ is $(e_t,-)$ then
6:      denote $\varphi_t(e)$ by $S$, set $\mathsf{cov}_t(S) = \mathsf{cov}_t(S)\setminus\{e\}$;
7:      suppose $S$ belongs to level $\mathcal{L}_t(i)$
8:      if $\mathsf{cov}_t(S)$ becomes empty then
9:          remove $S$ from $\mathcal{S}_t$
10:     else if current density $\rho_t(S,i)$ is $> 2^{i+10}$ then
11:         Let $\ell$ be highest level s.t. $2^\ell \le \rho_t(S,\ell)$
12:         move $S$ to level $\ell$.
13:     end if
14: end if
15: call Procedure Stabilize($t$)
```

The analysis of this algorithm involves showing the following properties:

Algorithm 4 Stabilize(t)

```
1: while $\exists S \in \mathcal{F}$ & level $i$ s.t. $c_S/\mathrm{vol}(S \cap A_t(i), i) < 2^i$ do
2:     let $i^\star \le i$ be the highest index such that $2^{i^\star} \le c_S/\mathrm{vol}(S \cap A_t(i), i^\star) \le 2^{i^\star+10}$
3:     add a copy of set $S$ to our solution
4:     set $\mathrm{cov}_t(S)$ to $S \cap A_t(i)$, assign $S$ to the level $i^\star$
5:     while $\exists X \in \mathcal{L}_t(i)$ s.t. $\mathrm{cov}_t(X) \cap \mathrm{cov}_t(S) \neq \emptyset$ do
6:         set $\mathrm{cov}_t(X) \leftarrow \mathrm{cov}_t(X) \setminus \mathrm{cov}_t(S)$
7:         update $\rho_t(X, i)$ accordingly
8:         if $\mathrm{cov}_t(X) = \emptyset$ then
9:             remove $X$ from the solution
10:        else if current density of $X$ is $> 2^{i+10}$ then
11:            Let $\ell$ be highest level s.t. $2^\ell \le \rho_t(X, \ell)$
12:            move $X$ to level $\ell$
13:        end if
14:    end while
15: end while
```

Correctness. For any unstable solution, the sequence of fix operations is finite. Furthermore, any subset can always be placed in some level.

Competitive Ratio. Any stable solution has total cost $O(\log n_t)$ times Opt_t.

Recourse. The number of sets added by this algorithm to the solution, averaged over the element arrivals and departures, is $O(1)$.

The proof of termination of Stabilize is identical to Claim 2.1 and we omit it for avoiding redundancy. Next, we prove the validity of the algorithm by showing that every subset can be placed at some level. In particular, this shows that the term i^* is well defined in the algorithm Stabilize.

Lemma 2.9. *Every set covering a non-empty subset of elements can be placed at some density level.*

Proof. Consider a set S covering a subset X of elements, and suppose it does not satisfy the density condition in invariant (i) for any level. Since $\mathrm{vol}(X, i)$ is an increasing function of i, there must be two adjacent levels i and $i+1$ such that the set S has too low a density for level $i+1$ and too high a density for level i. In other words, the former condition implies that $2^{i+1} > c_S/\mathrm{vol}(X, i+1)$ and $2^{i+10} < c_S/\mathrm{vol}(X, i)$. But note that by the way we have defined vol, we have $\mathrm{vol}(X, i) = 2\mathrm{vol}(X, i+1)$, which in turn implies that

$$2^{i+10} < c_S/\mathrm{vol}(X, i) = 2c_S/\mathrm{vol}(X, i+1) < 2 \cdot 2^{i+1} = 2^{i+2},$$

which gives us the desired contradiction. □

Bounding Cost. Next we bound the cost of our solution. Let Opt_t denotes the cost of the optimal solution at time t. Let ρ_t denote Opt_t/n_t. let i_t be the index such that $2^{i_t - 1} < \rho_t \le 2^{i_t}$.

We first begin with a simple but useful claim about the highest level an element can belong in.

Claim 2.10. *Consider any solution $\mathcal{S}_t$ which satisfies the invariants (i) and (ii). Then, for all elements $e \in A_t$, e will be covered in a level at most $b(e)$. So the volume of e is at most 1.*

Proof. Suppose not, and suppose there exists an element e currently covered in level $\ell \ge b(e) + 1$ in a solution $\mathcal{S}_t$ which satisfies invariants (i) and (ii). Let S be the cheapest set containing e. Therefore, by definition of $b(e)$, we have that $2^{b(e)} \le c_S < 2^{b(e)+1}$, and moreover, that $\mathrm{vol}(e, b(e)) = 1$. But now, we claim that S along with e would violate invariant (ii) at level ℓ. Indeed, we have that $c_S/\mathrm{vol}(e, \ell) \le c_S/2 < 2^{b(e)} < 2^\ell$, which establishes the desired contradiction. □

The next lemma asserts that all density levels lower than i_t can be more or less ignored in calculating the cost of the solution.

Lemma 2.11. *The total cost of all subsets in levels below i_t is $O(\mathrm{Opt}_t)$.*

Proof. Every set S at a level below i_t has density at most $2^{i_t+10} \le 2 \cdot (\mathrm{Opt}_t/n_t) \cdot 2^{10}$. Since the volume of every element at any levels is at most 1 (as it always appears in a level at most its base level from Claim 2.10), and there are at most n_t elements in the current active set A_t, the total cost of sets in all levels including and below level i_t is at most $\mathrm{Opt}_t \cdot 2^{11}$. □

We next bound the cost of sets in any single level.

Lemma 2.12. *The total cost of all subsets in any level $i \ge i_t$ at any time step is $O(\mathrm{Opt}_t)$.*

Proof. Suppose not, and there exists some level $i > i_t$ such that the sum of costs of sets in density level i is strictly greater than $\mathrm{Opt}_t \cdot 2^{10}$. Then, since the density of every set in level i is at most 2^{i+10}, it follows that the total volume of elements in level i is strictly greater than $\mathrm{Opt}_t/2^i$. Since these elements are covered by an optimal solution of total cost Opt_t, it follows that there *exists* some set $S \in \mathcal{F}$ which would have current density in this level strictly less than 2^i, which then contradicts invariant (ii). □

Finally, we show that levels above $i_t + \log n_t + 1$ are empty.

Lemma 2.13. *There are no sets in $\mathcal{S}_t$ at levels above $i_t + \log n_t$.*

Proof. This follows from Claim 2.10. Indeed, note that $\mathrm{Opt}_t \ge 2^{b(e)}$ for all $e \in A_t$ since $2^{b(e)}$ defines a lower bound on the cost of the cheapest set covering e. Therefore, the highest level with any element is at most $\max_{e \in A_t} b(e)$ which is at most $\log \mathrm{Opt}_t \le \log \rho_t + \log n_t \le i_t + \log n_t$. □

Bounding Recourse. Finally, we show the recourse bound. This will be proved via a *token* scheme, where the token invariant is that every element, when it first enters a level, has tokens equal to its volume at the corresponding level; over time, it may gain more tokens as it stays at this level. It uses these extra tokens if it moves to a higher level where our invariant will require it to have a higher token requirement. When an element arrives, the invariant is satisfied because it enters its base level and therefore, requires one token which can be charged to the element arrival. This is the only external injection of tokens into the system. Hence, if we are able to show a) that the token invariant can be maintained and b) whenever a new set is added to the solution, we can *expend* a constant λ amount of (fractional) tokens from the system, then we can claim that the total number of sets ever added (and therefore ever deleted) is at most $1/\lambda$ times the number of element arrivals.

We now describe the token-based argument we use to bound the recourse, starting with the crucial invariant we maintain and then the token distribution scheme.

Token Invariant: Consider any element $e \in A_t$ which is covered at level i in the current solution $\mathcal{S}_t$. Then, it has tokens at least as much as its level-i volume $\mathsf{vol}(e, i)$.

Token Distribution Scheme: As before, We inject tokens when elements arrive, and redistribute them when elements get covered by new sets or depart. Crucially, our redistribution would ensure that every time a new set is added to $\mathcal{S}_t$, a constant λ units of tokens are expended from the system. More formally:

(a) **Element arrival $\sigma_t = (e_t, +)$:** We give e_t *two units of tokens*, and e_t immediately *expends* one unit of token for the formation of the singleton set covering it.

(b) **Element departure $\sigma_t = (e_t, -)$:** Suppose e was covered by some set S at level i. Then e_t equally distributes its tokens (at least $\mathsf{vol}(e_t, i)$) to the remaining elements covered by S.

(c) Stabilize **operation:** Suppose we add a set S to the solution $\mathcal{S}_t$ at some level $i^\star$ during an iteration of the **While** loop in Stabilize. Consider any element e now covered by S but earlier covered by some set X at level i. Suppose e has $\tau_e \geq \mathsf{vol}(e, i)$ tokens (since it currently satisfies the token invariant). To satisfy e's new token requirement, e retains $\mathsf{vol}(e, i^\star)$ tokens. Then, *e expends $1/2(\tau_e - \mathsf{vol}(e, i))$ tokens* from the system toward the creation of this set, and equally distributes the remaining $1/2(\tau_e - \mathsf{vol}(e, i))$ tokens among the remaining elements still covered by X, i.e., the set of elements $\mathsf{cov}_t(X) \setminus \mathsf{cov}_t(S)$.

We now show that the above scheme will maintain the token invariant at all time-steps, and also guarantee a) that the number of tokens brought into the system is bounded by $O(t)$ after t element operations, and b) that at least $\lambda = 2^{-12}$ tokens are expended any time a new set is added to $\mathcal{S}_t$. To this end, we begin with the following easy claims.

Claim 2.14. *The total number of tokens introduced into the system is at most $2t$.*

Proof. The only injection of tokens into the system is on element arrivals, when we introduce 2 tokens into the system per arrival. □

Claim 2.15. *Whenever a new set is created, $\lambda \geq 2^{-11}$ units of tokens are expended from the system.*

Proof. If the new set is added when an element arrives, then we expend one token from the system by definition in the token distribution scheme. So consider the case when a new set S is formed at some time-step t in some level i. We first show that the total level-i volume of the elements $\mathsf{cov}_t(S)$ now covered by S is at least 2^{-10}. Indeed, consider an element $e \in \mathsf{cov}_t(S)$ with base level $b(e)$. Recall that the base level is defined according to the minimum cost set containing e, and so we have $c_S \geq 2^{b(e)}$. But since the set is formed at level i, its current density satisfies $\rho_t(S, i) \leq 2^{i+10}$. It follows that the total level-i volume $\mathsf{vol}(\mathsf{cov}_t(S), i)$ of elements covered by this set S is at least $c_S/\rho_t(S, i) \geq 2^{b(e)-i-10}$. So if $i \leq b(e)$, we get the desired bound on the volume of the new set formed. On the other hand, if $i > b(e)$, then it already has $\mathsf{vol}(e, i) > 1$ and again we have $\mathsf{vol}(\mathsf{cov}_t(S), i) \geq 1$ in this case.

Next, note that when the new set is formed, each element e newly covered by S has dropped its level by at least 1, since this is the criterion for forming new sets in Stabilize. This implies that each such element had at least $\mathsf{vol}(e, i^\star + 1) = 2\mathsf{vol}(e, i^\star)$ tokens before the new set was formed. Then, it retains $\mathsf{vol}(e, i^\star)$ tokens to satisfy its new token requirement, and expends *at least* $1/2(2\mathsf{vol}(e, i^\star) - \mathsf{vol}(e, i^\star)) = 1/2\mathsf{vol}(e, i^\star)$ tokens for the formation of the new set. Since each newly covered element does the same, we get that the total tokens expended is at least $1/2 \sum_{e \in \mathsf{cov}_t(S)} \mathsf{vol}(e, i^\star) \geq 2^{-11}$ from the above volume lower bound. □

Claim 2.16. *The* Token Invariant *is always satisfied.*

Proof. We first show that the invariant is satisfied after a new element arrives, i.e., the operation σ^t is $(e, +)$. By definition, element e gets covered by a set at level $b(e)$, and so, its token invariant requires it to have $\mathsf{vol}(e, b(e) = 1$ tokens which we give it when it arrives. Similarly, if the operation is $(e, -)$, we simply delete e, and there is one fewer token invariant to satisfy. Now we show that the operation Stabilize maintains the invariant. To this end, suppose we find a set S during an iteration of the **While** loop of procedure Stabilize. Firstly note that the elements which are covered by the new set have requisite number of tokens since they each drop their level by at least one and we ensure in the token scheme property (c) that they retain sufficiently many tokens to satisfy their token invariant at the new level (even after expending some tokens).

Now we turn our attention to the more challenging scenario when sets move up in level having lost many elements and their current density exceeds the threshold for their current level. So consider the setting when a set S moves up from level i to level $i + \ell$, and consider the first time t_{init} when a set S was added to level-i, and let $\mathsf{vol}(\mathsf{cov}_{t_{\mathsf{init}}}(S), i)$ denote the initial level-i volume of the elements it covers. Also consider the first time t_{fin} when it violates the invariant (i) for level i, and so it moves to some level, say $i + \ell$. Let its level-i volume of the remaining elements at this time be $\mathsf{vol}(\mathsf{cov}_{t_{\mathsf{fin}}}(S), i)$. We now make some observations regarding these volumes through the corresponding current densities.

Firstly, because this set moves up from level i its level-i current density $c_S/\mathsf{vol}(\mathsf{cov}_{t_{\mathsf{fin}}}(S), i)$ must be greater than 2^{i+10}. That is,

$$\mathsf{vol}(\mathsf{cov}_{t_{\mathsf{fin}}}(S), i) < c_S/2^{i+10} . \tag{5}$$

Likewise, since it does not move up higher than $i + \ell$, its level-$i + \ell + 1$-density $c_S/\mathsf{vol}(\mathsf{cov}_{t_{\mathsf{fin}}}(S), i+\ell+1)$ must be strictly less than $2^{i+\ell+1}$ (recall that when a set is relocated, it is placed in the highest level that can accommodate it). Therefore we have,

$$\mathsf{vol}(\mathsf{cov}_{t_{\mathsf{fin}}}(S), i + \ell + 1) > c_S/2^{i+\ell+1} . \tag{6}$$

Also, $\mathsf{vol}(\mathsf{cov}_{t_{\mathsf{fin}}}(S), i + \ell + 1) = 2^{\ell+1} \cdot \mathsf{vol}(\mathsf{cov}_{t_{\mathsf{fin}}}(S), i)$ by definition of the volume function. Along with Equations (5) and (6), this implies that $2^{i+\ell+1} \cdot 2^{\ell+1} > 2^{i+10}$, i.e., $\ell > 5$.

Next, we note that when the set first entered level i at time t_{init}, its level-$(i+1)$ density $c_S/\mathsf{vol}(\mathsf{cov}_{t_{\mathsf{init}}}(S), i+1)$ must have been less than

2^{i+1} (otherwise we would have placed it at level $i+1$ or above); therefore, we have

$$\mathsf{vol}(\mathsf{cov}_{t_{\mathsf{init}}}(S), i+1) > c_S/2^{i+1}\,, \tag{7}$$

and so it's level-i volume satisfies

$$\mathsf{vol}(\mathsf{cov}_{t_{\mathsf{init}}}(S), i) > c_S/2^{i+2}\,, \tag{8}$$

Similarly, when the set (and the remaining elements it covers) moves from level i to level $i+\ell$ at time t_{fin}, its level-$(i+\ell)$ density is at least $2^{i+\ell}$, i.e.,

$$\mathsf{vol}(\mathsf{cov}_{t_{\mathsf{fin}}}(S), i+\ell) \le c_S/2^{i+\ell}\,, \tag{9}$$

and hence its level-i volume satisfies

$$\mathsf{vol}(\mathsf{cov}_{t_{\mathsf{fin}}}(S), i) \le c_S/2^{i+2\ell}\,. \tag{10}$$

So from eqs. (8) and (10), and the fact that $\ell \ge 5$, we get

$$\mathsf{vol}(\mathsf{cov}_{t_{\mathsf{init}}}(S), i)/\mathsf{vol}(\mathsf{cov}_{t_{\mathsf{fin}}}(S), i) = 2^{2\ell-2} \ge 16 \cdot 2^{\ell}.$$

To complete the proof, note that all the elements in $\mathsf{cov}_{t_{\mathsf{init}}}(S) \setminus \mathsf{cov}_{t_{\mathsf{fin}}}(S)$ together left 1/4 of their $\mathsf{vol}(\mathsf{cov}_{t_{\mathsf{init}}}(S) \setminus \mathsf{cov}_{t_{\mathsf{fin}}}(S), i)$ tokens with the remaining elements $\mathsf{cov}_{t_{\mathsf{fin}}}(S)$. So the tokens which the remaining elements gained is at least

$$\begin{aligned} & 1/4\left(\mathsf{vol}(\mathsf{cov}_{t_{\mathsf{init}}}(S) \setminus \mathsf{cov}_{t_{\mathsf{fin}}}(S), i)\right) \\ = \quad & 1/4\big(\mathsf{vol}(\mathsf{cov}_{t_{\mathsf{init}}}(S), i) - \mathsf{vol}(\mathsf{cov}_{t_{\mathsf{fin}}}(S), i)\big) \\ \ge \quad & 1/4\big(16 \cdot 2^{\ell}\mathsf{vol}(\mathsf{cov}_{t_{\mathsf{fin}}}(S), i) - \mathsf{vol}(\mathsf{cov}_{t_{\mathsf{fin}}}(S), i)\big) \\ \ge \quad & 2^{\ell} \cdot \mathsf{vol}(\mathsf{cov}_{t_{\mathsf{fin}}}(S), i) = \mathsf{vol}(\mathsf{cov}_{t_{\mathsf{fin}}}(S), i+\ell) \end{aligned}$$

Hence, when the set (along with the remaining elements) moves up to level $i+\ell$, these elements have as many tokens as their volume in level $i+\ell$, ensuring that the token invariant holds. □

Thus, we have shown that the token invariant holds at all times. By Claims 2.14, 2.15, 2.16, we get the following lemma.

Lemma 2.17. *The total number of sets added till time T is at most $2T/\lambda$.*

Putting Theorems 2.3 and 2.17 together, we get:

Theorem 2.18. *For any sequence of T insertions or deletions, there is an efficient algorithm which maintains a collection of set cover solutions $\mathcal{S}_t$ such that each $\mathcal{S}_t$ is $O(\log n_t)$-competitive for the active set A_t, and the total recourse (i.e., number of sets added or deleted) is $O(T)$.*

3 DYNAMIC PRIMAL DUAL ALGORITHMS

In the previous section, we saw dynamic algorithms inspired by the greedy analysis of set cover. However, the natural greedy algorithms do not yield competitive ratio better than $O(\log n)$ even for special cases like the vertex cover problem. So we turn our attention to dynamic algorithms based on the *primal-dual* framework, which typically have approximation ratios depending on the parameter f, the maximum number of sets containing any element. Our first result is a deterministic fully-dynamic algorithm in the update time model with $O(f^3)$-competitiveness and an update time of $O(f^2)$ (establishing Theorem 1.2(b)). Note that for vertex cover this gives constant-competitiveness with *deterministic* constant update time. (The randomized version of this result for the special case of *unweighted* vertex cover follows from the recent dynamic maximal-matching algorithm of [37].) Our algorithm follows a similar framework as [10], but we use a more nuanced and apparently novel *asymmetric* token transfer scheme in the analysis to remove the dependence on n in the update time. We then outline our algorithm with improved bounds in the recourse model: we get f-competitiveness with $O(1)$ recourse, establishing Theorem 1.1(b).[4]

3.1 Dynamic Primal Dual for Update Time

Given a set system $(U, \mathcal{F})$, and an element e, let $\mathcal{F}_e$ denote the sets containing e, and let $f_e := |\mathcal{F}_e|$. For simplicity we assume we know f such that $f_e \le f$ for all $e \in A_t$; we discharge this assumption easily in the full version. Like the dynamic greedy algorithms in the earlier sections, our algorithm assigns each set to a *level*, but the intuition now is different. Firstly, only the sets in the solution $\mathcal{S}_t$ were assigned levels in Section 2.1, whereas *all* sets will be assigned levels in the primal-dual framework. Secondly, the intuition of a level in the greedy framework corresponds to the density (or incremental cost-per-element covered) of the sets, whereas the intuition of a level here loosely corresponds to how *quickly a set becomes tight* if we run the standard off-line primal-dual algorithm on the current set of active elements. Sets that become tight sooner are in higher levels. We also define *base levels*, but now these are defined not for elements but for sets.

Set and Element "Levels". At all times, each set $S \in \mathcal{F}$ resides at an integer level, denoted by $\mathsf{level}(S)$. For an element $e \in A_t$, define $\mathsf{level}(e) := \max_{S : e \in S} \mathsf{level}(S)$ to be the largest level of any set covering it.

Set and Element "Dual Values". Given levels for sets and elements, the *dual value* of an element e is defined to be $y(e) := 2^{-\mathsf{level}(e)}$. The dual value of a set $S \in \mathcal{F}$ is the sum of dual values of its elements, $y(S) := \sum_{e \in S \cap A_t} y(e)$.

Recall the dual of the set cover LP:

$$\max\{\textstyle\sum_{e \in A_t} \tilde{y}_e \mid \sum_{e \in A_t \cap S} \tilde{y}_e \le c_S \;\forall S \in \mathcal{F},\; \tilde{y}_e \ge 0\}. \tag{11}$$

Now our solution at time t is simply all sets whose dual constraints are *approximately tight*, i.e., $\mathcal{S}_t = \{S : y(S) \ge c_S/\beta\}$ for $\beta := 32f$. In addition, we will try to ensure that the duals $y(e)$ we maintain will be approximately feasible for (11), i.e., for every set S, $y(S) \le \beta c_S$. Then, bounding the cost becomes a simple application of LP duality.

The main challenge is in maintaining such a dual solution dynamically. We achieve this by defining a *base level* for every set to indicate non-tight dual constraints, and ensuring that all sets strictly *above their base levels* always have approximately tight dual constraints. Formally, the *base level* for set S is defined to be $b(S) := -\lceil \log(\beta c_S) \rceil - 1$, and our solution $\mathcal{S}_t$ consists of *all sets* that are located strictly above their respective base levels, i.e., $\mathcal{S}_t = \{S \in \mathcal{F} : \mathsf{level}(S) > b(S)\}$. (To initialize, each set S is placed at level $b(S)$.) We will ensure that each set resides at level $b(S)$ or higher.

We ensure the approximate tightness and approximate feasibility of dual constraints using the *stability property* below.

[4]In fact, we can get stronger guarantees for both the algorithms to have competitive ratio depend only on f_t, the *maximum frequency at time t* and not $f = \max_t f_t$; details appear in the full version [20].

Stable Solutions. A solution $\mathcal{S}_t$ is *stable* if: for all sets S with $\mathsf{level}(S) > b(S)$ we have $y(S) \in [c_S/\beta, \beta c_S]$, and for all S with $\mathsf{level}(S) = b(S)$ we have $y(S) < \beta c_S$.

We now give the intuition behind this definition of base levels. Suppose a new element e arrives and it is uncovered, i.e., all its covering sets are at their base levels. Then, element e has a dual value of $y(e) = 1/2^{b(S_e)} > 2\beta c_{S_e}$, where S_e is the minimum-cost set containing e. This implies that $y(S_e) > 2\beta c_{S_e}$, and so our algorithm moves S_e up to a higher level and includes it in the solution.

Loosely, the algorithm follows the principle of least effort towards ensuring stability of all sets: if at some point $y(S)$ is too large, S moves up the least number of levels so that the resulting $y(S)$ falls within the admissible range – observe that as S moves up one level, every element e in S for which $\mathsf{level}(e) = \mathsf{level}(S)$ halves its current dual value. Similarly if $y(S)$ is too small, it moves down until $y(S) \geq c_S/\beta$. Indeed, since the competitive ratio is defined purely by the two barriers $1/\beta$ and β, this lazy approach is very natural if the goal is to minimize the number of updates.

Arrival: When e arrives, the current levels for sets define $y(e) := 1/\max_{S:e\in S} 2^{\mathsf{level}(S)}$. Update $y(S)$ for all sets. Run Stabilize.

Departure: Delete e from A_t. Update $y(S)$ for all sets. Run Stabilize.

Stabilize: Repeatedly perform the following steps until the solution is stable: If for some set S at level $\mathsf{level}(S)$ we have $y(S) > \beta c_S$, find the *lowest level* $\ell' > \mathsf{level}(S)$ such that placing S at level ℓ' results in $y(S) \leq \beta c_S$. Analogously, if $y(S) < c_S/\beta$, find the *highest* level ℓ', $b(S) \leq \ell' < \mathsf{level}(S)$, such that placing S at level ℓ' will satisfy the stability condition for S at this level (it can be shown that such a level always exists).

Lemma 3.1 (Stability ⇒ Approximation). *Any stable solution $\mathcal{S}_t$ has cost $O(f^3)\,\mathrm{Opt}_t$.*

Proof. The cost of $\mathcal{S}_t$ is
$$\sum_{S\in\mathcal{S}_t} c_S \leq \beta \sum_{S\in\mathcal{S}_t} y(S) = \beta \sum_{S\in\mathcal{S}_t}\sum_{e\in S\cap A_t} y(e) \leq f\beta \sum_{e\in A_t} y(e).$$
Now since $y(e)/\beta$ is a feasible dual solution (see the set cover dual in (11)), the lemma follows from LP duality. □

3.1.1 *Bounding the Update Time.* As with the analyses of our dynamic greedy algorithms, most updates happen when the solution stabilizes itself at various points. Indeed, even termination of Stabilize is a priori not clear. In any call to Stabilize when a set S moves, let $\mathsf{out}^{\mathsf{old}}(S)$ and $\mathsf{out}^{\mathsf{new}}(S)$ respectively denote the *out-elements* of S at the beginning and end of the move. These are the elements whose level is equal to the level of S, i.e., elements for which all sets containing them reside at levels $\mathsf{level}(S)$ or lower.

Following Solomon [37], we first show that the total update time for maintaining our data structures in an upward move is $O(f\cdot|\mathsf{out}^{\mathsf{new}}(S)|)$, and that in a downward move is $O(f\cdot|\mathsf{out}^{\mathsf{old}}(S)|)$. We provide details of the data structures establishing these bounds in the full version [20].

Given these observations, it suffices to control the sizes of the sets $\mathsf{out}^{\mathsf{new}}(S)$ and $\mathsf{out}^{\mathsf{old}}(S)$, and charge them to distinct arrivals. Again, we use a token scheme for the amortized analysis of these costs.

Token Distribution: When an element arrives, it gives $20f$ tokens to each of the f sets containing it, totaling $O(f^2)$ tokens. When an element departs, it gives 1 token to each of the f sets covering it. When a set moves up from some level ℓ to level ℓ^* in Stabilize, it expends $f\cdot|\mathsf{out}^{\mathsf{new}}(S)|$ tokens to pay for the update-time, and for all $e \in \mathsf{out}^{\mathsf{new}}(S)$ it transfers 1 token to each set $S' \in \mathcal{F}_e$. Similarly when a set moves down in Stabilize, it expends $f\cdot|\mathsf{out}^{\mathsf{old}}(S)|$ tokens, and for all $e \in \mathsf{out}^{\mathsf{old}}(S)$ it transfers $20f$ tokens to each $S' \in \mathcal{F}_e$. *Note the asymmetry between the number of tokens transferred in the two cases!*

Note that at most $20f^2$ tokens are injected per element arrival or departure. Moreover, from the discussion above, the total update time for each up/downward move is $O(1)$ times the number of tokens expended from the system. In what follows, we show that we never expend more tokens than we inject – thus obtaining an amortized $O(f^2)$ bound on update times. To this end, we divide the movement of a set S into *epochs*, where each *up-epoch* is a maximal set of contiguous upward moves of S and a *down-epoch* is a maximal set of contiguous downward moves of S. (Epochs may span different calls to Stabilize.)

Lemma 3.2 (Up-Epochs). *Consider an up-epoch of a set S ending at level ℓ. The total number of tokens expended or transferred by S during this epoch is at most $2^{\ell+8}f^2c_S$. Moreover, the total number of tokens that S gained during this epoch is at least $2^{\ell+8}f^2c_S$.*

Proof. Consider an up-epoch where S moves from $\ell_0 \to \ldots \to \ell_k = \ell$ via a sequence of up-moves. The tokens expended and transferred during the upward move $\ell_{i-1} \to \ell_i$ is $(f+f)\cdot|\mathsf{out}^{\mathsf{new}}(S)| \leq (2f)\beta\, 2^{\ell_i} c_S$. The inequality holds because S satisfies the stability condition when it settles at level ℓ_i, and each out-element has dual value $1/2^{\ell_i}$. Therefore, the *total tokens* transferred/expended in this *entire up-epoch* is at most
$$2f\sum_{i=1}^{k}\beta 2^{\ell_i}c_S \leq 4\beta f 2^{\ell} c_S.$$
Using $\beta = 32f$ proves the first claim.

For the second claim, consider the moment when this epoch started. We first observe that when S had moved down to level ℓ_0 at the end of the previous epoch say at time t_0, then $y_{t_0}(S) < 2c_S/\beta$. Indeed, if this epoch is the first ever epoch for set S then y_S was 0. Else it was preceded by a down-epoch, in which case the final down-move in the previous epoch happened because $y_S < c_S/\beta$ and the down-move can at most double the $y_{t_0}(S)$ value at level ℓ_0. Let $S(t_0)$ be the elements of S which were active at that time. Now suppose at time t_0, we hypothetically placed S in level $\ell - 1$ without changing the levels of other sets. Let $y'_{t_0}(e)$ be the corresponding dual values for elements given by the sets being in these levels, and let $y'_{t_0}(S) := \sum_{e\in S(t_0)} y'_{t_0}(e)$. Clearly, $y'_{t_0}(S) < 2c_S/\beta$ as well, since we are hypothetically placing S at a higher level $\ell - 1$ instead of $\ell_0 \leq \ell - 1$. Now consider the ending time t_1 of the up-epoch, just before we move S from $\ell - 1$ to ℓ. Let $S(t_1)$ be the elements of S active at this time, and let $y_{t_1}(e)$ be the dual values for all elements in $S(t_1)$. Then, $y_{t_1}(S) := \sum_{e \in S(t_1)} y_{t_1}(e)$. Clearly $y_{t_1}(S) > \beta c_S$, else we will not move S from $\ell - 1$ to ℓ.

Note that we placed S at level $\ell - 1$ in both settings, so the increase from $y'_{t_0}(S)$ to $y_{t_1}(S)$ of more than $(\beta - 2)c_S$ can only happen because (a) there are elements in $S(t_1)$ that are not present in $S(t_0)$, or (b) there are elements in $S(t_1) \cap S(t_0)$ that have moved down since time t_0. Each such element can contribute at most $2^{-(\ell-1)}$ to $y_{t_1}(S) - y'_{t_0}(S)$, so there must be at least $2^{\ell-1} \cdot (\beta - 1)c_S \geq 15 f c_S 2^{\ell}$ events in total, and our token distribution scheme now says that S would have collected at least $300 f^2 c_S 2^{\ell} \geq 2^{\ell+8} f^2 c_S$ tokens in this epoch, completing the proof. □

Remark: Note that in the above lemma, the update time is bounded in terms of tokens expended, and this seemingly depends on set cost c_S. At first glance, this might appear strange because update times should be invariant to cost scaling. A closer scrutiny, however, reveals that the update time is indeed scale-free since higher set costs imply lower levels in the algorithm, and vice-versa. In other words, on scaling, the change in c_S is compensated by an opposite change in 2^{ℓ}.

A similar lemma holds for down-epochs.

Lemma 3.3 (Down-Epochs). *Consider a down-epoch for set S starting at level ℓ. The number of tokens expended/transferred by S during this epoch is at most $2^{\ell+1} f c_S$. Moreover, the total number of tokens S gained in the beginning of this epoch is at least $2^{\ell+1} f c_S$.*

The above two lemmas show that for each set S and each epoch, the total number of tokens S expends or transfers is no more than what it receives via transfers or arrivals/departures. This shows that the total tokens never becomes negative, and hence proves the amortized bound.

3.2 Dynamic Primal Dual for Recourse

In this section, we consider the recourse model and give an algorithm with stronger guarantees than the one in the previous section. Our algorithm is inspired by the following offline algorithm for set cover – arrange the elements in some arbitrary order and probe them in this order. We maintain a tentative solution $\mathcal{S}$ which is initially empty. When we probe an element e, two cases arise: (i) the element e is already covered by an element $e \in \mathcal{S}$: in this case, we do nothing, or (ii) there is no such set in $\mathcal{S}$: in this case, we pick a random set from $\mathcal{F}_e$, where a set $S \in \mathcal{F}_e$ is chosen with probability $\frac{1/c_S}{\sum_{S' \in \mathcal{F}_e} 1/c_{S'}}$. This algorithm is $O(f)$-competitive in expectation [33].

We now describe our *dynamic implementation* of this algorithm. Recall that A_t denotes the set of active elements at time t. At all times t, we maintain a partition of A_t into two sets P_t (called the *probed set*) and Q_t (the *unprobed set*). Elements on which we have performed the random experiment outlined above are the ones in the probed set. We also maintain a bijection φ from P_t to $\mathcal{S}_t$, the set cover solution at time t, i.e., for every probed element, there is a unique set in $\mathcal{S}_t$ and vice-versa. Elements can move from Q_t to P_t at a latter point of time (i.e., an element currently in Q_t can be in $P_{t'}$ for some $t' > t$), but once an element is in set P_t it stays in $P_{t'}$ for all $t' \geq t$ (till the element departs).

We now describe the procedures which will be used our algorithm. At certain times, our algorithm may choose to *probe* an unprobed element. This will happen when there is no set in the current solution covering this element.

Probing an element. When an unprobed element $e \in Q_{t-1}$ is probed by the algorithm at time t, it selects a single set that it belongs to, where a set S containing e is chosen with probability $\frac{1/c_S}{\sum_{S' \in \mathcal{F}_e} 1/c_{S'}}$. This chosen set S is added to the current solution of the algorithm: $\mathcal{S}_t := \mathcal{S}_{t-1} \cup \{S\}$. Element e moves from the unprobed set to the probed set: $P_t = P_{t-1} \cup \{e\}$ and $Q_t = Q_{t-1} \setminus \{e\}$. As long as e remains in the instance, i.e., is not deleted, the set S will also remain in the solution. We say that e is responsible for S and denote $\varphi(e) := S$.

Having defined the process of probing elements, we explain how elements are probed. These probes are triggered by element insertions and deletions as described below.

Element Arrivals. Suppose element e arrives at time t. If e is already covered in the current solution $\mathcal{S}_{t-1}$, it is added to the unprobed set (i.e., $Q_t = Q_{t-1} \cup \{e\}$), and the solution remains unchanged ($\mathcal{S}_t = \mathcal{S}_{t-1}$). Else, if e is not covered in the current solution, then it is probed (which adds a set $\varphi(e)$ to $\mathcal{S}_t$ as described above), and we set $P_t = P_{t-1} \cup \{e\}$.

Element Departures. Suppose element e departs from the instance at time t. If e is currently unprobed, then we set $Q_t = Q_t \setminus \{e\}$, but the solution remains unchanged: $\mathcal{S}_t = \mathcal{S}_{t-1}$. Else if e is a probed element, then we set $P_t = P_{t-1} \setminus \{e\}$. In addition, the set $\varphi(e)$ is also removed from the previous solution $\mathcal{S}_{t-1}$. This might lead to some elements in Q_t becoming uncovered in the current solution. We pick the first uncovered element[5] and probe it, which adds a set covering it to $\mathcal{S}$. This set might cover some previously uncovered elements. This process continues, with the first uncovered element in the unprobed set being probed in each iteration. The probing ends once all elements in the unprobed set (and therefore, all elements overall) are covered by the chosen sets. We then define this solution as $\mathcal{S}_t$.

It is easy to see the recourse bound: no set is deleted when elements arrive, and at most 1 set is deleted when an element departs. (Recall that we piggyback set additions on deletions.)

Competitive Ratio. To show the competitive ratio, it is useful to denote the set of elements in S at time t by S_t. Note that S_t evolves over time. The main property that we prove is the following.

Lemma 3.4. *At time t, for any set S_t,*

$$\mathbb{E}\left[\sum_{e \in S_t \cap P_t} c_{\varphi(e)}\right] \leq f \cdot c_{S_t}.$$

Proof. Since the adversary is oblivious of the random choices made by the algorithm, we can consider a fixed input sequence. Further, we condition on the random choices of elements not in S_t (irrespective of whether they are probed or not). We will show the desired bound under this conditioning, and so the result will follow once we remove the conditioning.

The execution of our algorithm can now be seen as a tree. Given the fixed input sequence, and coin tosses of elements not in S_t, we can determine the first element probed by the algorithm. Similarly, given the random choices of the first i elements in S_t which get

[5] We assume an arbitrary but fixed ordering on the elements.

probed, we can determine which element in S_t will be probed next. Therefore, let us define a sequence of random variables $X_1, X_2, \ldots$ and $Y_1, Y_2, \ldots$, where X_i is the i^{th} element of S_t that gets probed, and Y_i is the set selected by this probed element. If we probe less than i elements, then X_i and Y_i are defined as $\perp$. Now observe that given $X_1, Y_1, \ldots, X_{i-1}, Y_{i-1}$, we can determine the identity of X_i. Also, we need to define these random variables for $i \le |S_t|$ only. For notational convenience, let m denote $|S_t|$. Finally, observe that

$$\mathbb{E}\left[\sum_{e \in S_t \cap P_t} c_{\varphi(e)}\right] = \sum_i \mathbb{E}\left[c_{Y_i}\right],$$

where c_{Y_i} is 0 if $Y_i = \perp$. Now we state the induction hypothesis.

$$\text{IH}(i)\text{: } \mathbb{E}\left[\sum_{j=1}^m c_{Y_j} \mid X_1, Y_1, \ldots, X_{i-1}, Y_{i-1}\right] \le f \cdot c_{S_t}.$$

Let us check this for $i = m$. Suppose we are given $X_1, Y_1, \ldots, X_{m-1}, Y_{m-1}$. Then we know the identity of X_m. If $X_m = \perp$, we do not incur any cost, and so the statement holds trivially. Else, suppose $X_m = e \in S_t$. Then the expected cost of Y_m is

$$\sum_{S' : e \in S'} c_{S'} \cdot \frac{1/c_{S'}}{\sum_{S'' : e \in S''} 1/c_{S''}} = \frac{f}{\sum_{S'' : e \in S''} 1/c_{S''}} \le f \cdot c_{S_t}.$$

Now suppose the statement holds for $i+1$, and we want to prove it for i. Suppose we are given the random variables $X_1, Y_1, \ldots, X_{i-1}, Y_{i-1}$. This determines X_i also. Therefore, we can write the desired sum as

$$\mathbb{E}\left[c_{Y_i} \mid X_1, Y_1, \ldots, X_{i-1}, Y_{i-1}, X_i\right] + \\ \mathbb{E}\left[c_{Y_{i+1}} + \ldots + c_{Y_m} \mid X_1, Y_1, \ldots, X_{i-1}, Y_{i-1}, X_i\right].$$

Assume $X_i \ne \perp$, otherwise both terms are 0 are we would be done. Let e denote X_i, and Δ_e denote $\sum_{S' : e \in S'} 1/c_{S'}$. The first term is at most f/Δ_e, the argument being similar to the base case, $i = m$. Let q_e denote the probability that element e chooses set S. So $q_e = \frac{1}{c_{S_t}\Delta_e}$ and we can write the second term as

$$\sum_{S' : e \in S', S' \ne S_t} \mathbb{E}\left[c_{Y_{i+1}} + \ldots + c_{Y_m} \mid X_1, Y_1, \ldots, X_i, Y_i = S'\right] \cdot \\ \mathbb{P}\left[Y_i = S' \mid X_1, Y_1, \ldots, X_{i-1}, Y_{i-1}, X_i\right].$$

Using the induction hypothesis and independence, this can be bounded by

$$\sum_{S' : e \in S', S' \ne S_t} f \cdot c_{S_t} \cdot \mathbb{P}[Y_i = S' | X_i] = f \cdot c_{S_t} \cdot (1 - q_e).$$

So, the overall sum is at most

$$\frac{f}{\Delta_e} + \left(1 - \frac{1}{c_{S_t}\Delta_e}\right) f \cdot c_{S_t} = f \cdot c_{S_t}.$$

This proves the lemma. □

To complete the analysis, we use a dual fitting argument. Define the dual of an element e as $c_{\varphi(e)}/f$ (and 0 if the element is unprobed). The above lemma implies that this dual is feasible in expectation. Note that the dual objective is a $1/f$ fraction of the cost of algorithm's solution. Hence, by standard LP duality, the competitive ratio of the algorithm is f in expectation.

ACKNOWLEDGMENTS

We thank Yuting Ge and Archit Kulkarni for enlightening discussions. Theorem 1.3 was obtained in conversations with Yuting. This work was done in part while the authors were visiting the Simons Institute for the Theory of Computing, as part of the *Algorithms and Uncertainty* program. A. Gupta is supported in part by NSF Awards CCF-1536002, CCF-1540541, and CCF-1617790. D. Panigrahi is supported in part by NSF Awards CCF-1527084 and CCF-1535972.

REFERENCES

[1] Abboud, A., and Williams, V. V. Popular conjectures imply strong lower bounds for dynamic problems. In *FOCS* (2014), pp. 434–443.

[2] Alon, N., Awerbuch, B., Azar, Y., Buchbinder, N., and Naor, J. The online set cover problem. *SIAM J. Comput. 39*, 2 (2009), 361–370.

[3] Andrews, M., Goemans, M. X., and Zhang, L. Improved bounds for on-line load balancing. *Algorithmica 23*, 4 (1999), 278–301.

[4] Azar, Y., Broder, A. Z., and Karlin, A. R. On-line load balancing. *Theoretical Computer Science 130*, 1 (1994), 73–84.

[5] Azar, Y., Kalyanasundaram, B., Plotkin, S. A., Pruhs, K., and Waarts, O. Online load balancing of temporary tasks. In *Proceedings of the 1993 Workshop on Algorithms and Data Structures* (1993).

[6] Baswana, S., Gupta, M., and Sen, S. Fully dynamic maximal matching in $O(\log n)$ update time. *SIAM J. Comput. 44*, 1 (2015), 88–113.

[7] Bernstein, A., and Stein, C. Fully dynamic matching in bipartite graphs. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I* (2015), pp. 167–179.

[8] Bernstein, A., and Stein, C. Faster fully dynamic matchings with small approximation ratios. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016* (2016), pp. 692–711.

[9] Bhattacharya, S., Chakrabarty, D., and Henzinger, M. Deterministic fully dynamic approximate vertex cover and fractional matching in $o(1)$ amortized update time. In *ArXiv Pre-print dated 1^{st} November* (2016).

[10] Bhattacharya, S., Henzinger, M., and Italiano, G. F. Design of dynamic algorithms via primal-dual method. In *Automata, languages, and programming. Part I*, vol. 9134 of *Lecture Notes in Comput. Sci.* Springer, Heidelberg, 2015, pp. 206–218.

[11] Bhattacharya, S., Henzinger, M., and Italiano, G. F. Deterministic fully dynamic data structures for vertex cover and matching. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms* (2015), SIAM, Philadelphia, PA, pp. 785–804.

[12] Bhattacharya, S., Henzinger, M., and Nanongkai, D. New deterministic approximation algorithms for fully dynamic matching. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016* (2016), pp. 398–411.

[13] Bosek, B., Leniowski, D., Sankowski, P., and Zych, A. Online bipartite matching in offline time. In *FOCS* (2014), pp. 384–393.

[14] Chaudhuri, K., Daskalakis, C., Kleinberg, R. D., and Lin, H. Online bipartite perfect matching with augmentations. In *INFOCOM* (2009), pp. 1044–1052.

[15] Dinur, I., and Steurer, D. Analytical approach to parallel repetition. In *STOC'14—Proceedings of the 2014 ACM Symposium on Theory of Computing*. ACM, New York, 2014, pp. 624–633.

[16] Eppstein, D., Galil, Z., and Italiano, G. F. Dynamic graph algorithms. In *Algorithms and theory of computation handbook*. CRC, Boca Raton, FL, 1999, pp. 8–1–8–25.

[17] Epstein, L., and Levin, A. Robust algorithms for preemptive scheduling. In *ESA*, vol. 6942 of *Lecture Notes in Comput. Sci.* Springer, Heidelberg, 2011, pp. 567–578.

[18] Grove, E. F., Kao, M.-Y., Krishnan, P., and Vitter, J. S. Online perfect matching and mobile computing. In *WADS* (1995), pp. 194–205.

[19] Gu, A., Gupta, A., and Kumar, A. The Power of Deferral: Maintaining a Constant-Competitive Steiner Tree Online. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing* (2013), D. Bonesh, J. Feigenbaum, and T. Roughgarden, Eds., STOC '13, ACM, pp. 525–534.

[20] Gupta, A., Krishnaswamy, R., Kumar, A., and Panigrahi, D. Online and dynamic algorithms for set cover. *CoRR abs/1611.05646* (2016).

[21] Gupta, A., and Kumar, A. Online steiner tree with deletions. In *SODA* (Jan 2014), pp. 455–467.

[22] Gupta, A., Kumar, A., and Stein, C. Maintaining assignments online: Matching, scheduling, and flows. In *SODA* (2014), pp. 468–479.

[23] Gupta, M., and Peng, R. Fully dynamic $(1+\epsilon)$-approximate matchings. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA* (2013), pp. 548–557.

[24] Henzinger, M., Krinninger, S., Nanongkai, D., and Saranurak, T. Unifying and strengthening hardness for dynamic problems via the online matrix-vector

multiplication conjecture. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015* (2015), pp. 21–30.

[25] Imase, M., and Waxman, B. M. Dynamic Steiner tree problem. *SIAM J. Discrete Math. 4*, 3 (1991), 369–384.

[26] Kopelowitz, T., Pettie, S., and Porat, E. Higher lower bounds from the 3sum conjecture. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016* (2016), pp. 1272–1287.

[27] Korman, S. On the use of randomization in the online set cover problem. *M.S. thesis, Weizmann Institute of Science* (2005).

[28] Łacki, J., Oćwieja, J., Pilipczuk, M., Sankowski, P., and Zych, A. The Power of Dynamic Distance Oracles: Efficient Dynamic Algorithms for the Steiner Tree. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing* (New York, NY, USA, 2015), STOC '15, ACM, pp. 11–20.

[29] Megow, N., Skutella, M., Verschae, J., and Wiese, A. The power of recourse for online MST and TSP. In *ICALP (1)* (2012), pp. 689–700.

[30] Neiman, O., and Solomon, S. Simple deterministic algorithms for fully dynamic maximal matching. *ACM Trans. Algorithms 12*, 1 (2016), 7.

[31] Onak, K., and Rubinfeld, R. Maintaining a large matching and a small vertex cover. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010* (2010), pp. 457–464.

[32] Phillips, S., and Westbrook, J. On-line load balancing and network flow. *Algorithmica 21*, 3 (1998), 245–261.

[33] Pitt, L. A simple probabilistic approximation algorithm for vertex cover. Tech. Rep. YaleU/DCS/TR-404, Yale University, 1985.

[34] Sanders, P., Sivadasan, N., and Skutella, M. Online scheduling with bounded migration. *Math. Oper. Res. 34*, 2 (2009), 481–498.

[35] Sankowski, P. Faster dynamic matchings and vertex connectivity. In *SODA* (2007), pp. 118–126.

[36] Skutella, M., and Verschae, J. A robust PTAS for machine covering and packing. In *ESA (I)*, vol. 6346 of *LNCS*. Springer, Berlin, 2010, pp. 36–47.

[37] Solomon, S. Fully dynamic maximal matching in constant update time. In *FOCS* (2016).

[38] Westbrook, J. Load balancing for response time. *J. Algorithms 35*, 1 (2000), 1–16.

[39] Williamson, D. P., and Shmoys, D. B. *The design of approximation algorithms*. Cambridge University Press, Cambridge, 2011.