Dynamic Dominating Set in Uniformly Sparse Graphs

Anton Bukov
Tel Aviv University, Israel

Shay Solomon
Tel Aviv University, Israel

Abstract

In the dynamic *minimum dominating set (MDS)* problem, the goal is to efficiently maintain an approximate MDS in an n-vertex graph with vertex costs in $[1/C, 1]$ undergoing edge insertions and deletions. In STACS'19 [17] it was shown that an $O(\log n)$-approximate MDS can be maintained in *unweighted graphs* with $O(\Delta \cdot \log n)$ update time, where Δ is an upper bound on the maximum degree throughout the update sequence, and in STOC'23 [27] this was extended to weighted graphs and improves the approximation guarantee to $(1+\epsilon)\ln\Delta$.

Is it possible to achieve $\text{poly}(\log n)$ update time without any dependence on Δ, for any nontrivial graph family? This basic question has remained open even in **forests** and even for **unweighted instances**. The *arboricity* $\alpha = \alpha(G)$ of a graph G is the minimum number of edge-disjoint forests whose union is G, and is a standard measure of sparsity. While α is bounded by Δ in any graph, various real-world graph families exhibit a significant gap between α and Δ.

In this work, we show that one can maintain an $O(\alpha)$-approximate MDS with update time $O(\alpha \cdot \log(Cn))$, for dynamic graphs whose *arboricity* is bounded by α throughout the update sequence. This replaces the dependence on Δ in prior update bounds with α, while also improving the approximation guarantee for bounded-arboricity graphs. In particular, for any graph family of constant arboricity, such as planar graphs, bounded treewidth graphs, and more generally graphs excluding a fixed minor, our algorithm gives an $O(1)$-approximation with $O(\log(Cn))$ update time. To achieve this result, our algorithm departs from prior *greedy-based* approaches, relying instead on the *primal-dual framework* and new structural insights specific to bounded arboricity graphs.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases dynamic algorithms, minimum dominating set, arboricity, sparse graphs, approximation algorithms, primal-dual methods

Digital Object Identifier 10.4230/LIPIcs.ESA.2026.109

Related Version *Full Version*: `https://arxiv.org/abs/2607.24514`

Funding *Anton Bukov*: Funded by the European Union (ERC, DynOpt, 101043159).
Shay Solomon: Funded by the European Union (ERC, DynOpt, 101043159). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them. Also funded by a grant from the United States-Israel Binational Science Foundation (BSF), Jerusalem, Israel, and the United States National Science Foundation (NSF).

1 Introduction

This work studies the classic problem of *minimum dominating set (MDS)*, where we are given an n-vertex m-edge graph $G = (V, E)$ in which each vertex v has a cost $c_v \in [\frac{1}{C}, 1]$. A subset of vertices $D \subseteq V$ is called a *dominating set* if, for every vertex $v \in V$, either $v \in D$ or v has a neighbor in D. The goal is to compute a dominating set of minimum total cost.

This problem is closely related to the *minimum set cover (MSC)* problem: given a set system $(\mathcal{U}, \mathcal{S})$, where $\mathcal{U}$ is a universe of n elements and $\mathcal{S}$ is a family of m sets, each with cost in $[\frac{1}{C}, 1]$, the goal is to compute a minimum-cost subfamily covering $\mathcal{U}$. The two problems admit approximation-preserving reductions and have been studied in many cases together.

A classical *greedy* algorithm achieves logarithmic approximations for both problems. More precisely, letting $\Delta = \Delta(G)$ denote the maximum degree of G and D the maximum set size, the greedy algorithm gives an $H(\Delta + 1)$-approximate MDS and an $H(D)$-approximate MSC, where $\forall k, H(k) := \sum_{i=1}^{k} \frac{1}{i}$. A *primal-dual (PD)* algorithm yields a $(\Delta + 1)$-approximate MDS and an f-approximate MSC, where f is the *frequency* of the set system.

The greedy approach is essentially optimal for both problems, as one cannot achieve better approximations (to within a factor of $1 - \epsilon$) unless P = NP [30, 12, 11]. The PD approach behaves differently: an $(f - \epsilon)$-approximation cannot be achieved in polynomial time under the unique games conjecture [18], while the $(\Delta + 1)$-approximation obtained for MDS by the PD algorithm is exponentially worse than the greedy guarantee. However, in **uniformly sparse graphs** the situation changes, and the PD approach can be used to obtain significantly improved approximations for MDS, which is the focus of this work.

The MDS problem has been studied extensively in various settings and computational models. For brevity, the literature survey that follows is restricted to previous work on the MDS problem either for uniformly sparse graphs or in the dynamic setting.

Uniformly sparse graphs

A standard notion that captures sparsity in a uniform manner is *arboricity*. A graph $G = (V, E)$ has *arboricity* $\alpha = \alpha(G)$ if $\alpha = \max_{S \subseteq V} \left\lceil \frac{m_S}{n_S - 1} \right\rceil$, where m_S and n_S denote the number of edges and vertices in the subgraph induced by S, respectively. By the Nash-Williams Theorem [22], α is the minimum number of edge-disjoint forests into which the edges of G can be partitioned. Up to constants, arboricity equals the maximum average degree over all subgraphs and other basic measures, such as *degeneracy* and *thickness*. While $\alpha(G) \leq \Delta(G)$ holds for every graph, the gap between the two parameters can be large, e.g., for the n-star, $\alpha = 1$ but $\Delta = n - 1$. Significant gaps between Δ and α are common in real-world networks, including social networks, the world wide web graph, and transaction graphs, as well as in stochastic settings such as the preferential attachment model. The family of bounded-arboricity graphs is broad: even for $\alpha = O(1)$, it includes all graphs excluding a fixed minor, and hence all bounded-treewidth and bounded-genus (including planar) graphs.

In the classic sequential setting, an $O(\alpha)$-approximation can be computed in near-linear time [3, 21, 29, 13]; in particular, Sun [29] gave a fast PD $(\alpha + 1)$-approximation algorithm for weighted graphs of arboricity α. This bound is almost tight: one cannot achieve $(\alpha - \epsilon)$-approximation in polynomial time under the unique games conjecture [3].

In the CONGEST model of distributed computing, improving over [20, 21], Dory et al. [13] presented a deterministic $(2\alpha + 1)(1 + \epsilon)$-approximation algorithm running in $O(\epsilon^{-1} \log \Delta)$ rounds and a randomized $\alpha(1 + o(1))$-approximation algorithm running in $O(\alpha \log \Delta)$ rounds.

The dynamic setting

We study the MDS problem in the *standard dynamic setting*, where the graph undergoes a sequence of edge insertions and deletions on a fixed vertex set, starting from an empty graph. The goal is to maintain an approximate MDS with low update time; we focus on optimizing the *amortized* rather than *worst-case* update time; for brevity, we will mostly ignore the distinction between these two update time measures.

There are only a few results on the MDS problem in the dynamic setting, and they are obtained by adapting dynamic MSC algorithms. In contrast, the dynamic MSC problem has been studied extensively over the past decade. Most of the known algorithms are PD-based [5, 14, 1, 2, 6, 7, 9, 28], and they provide $(1+\epsilon)f$-approximate MSC with increasingly better update times,[1] culminating in deterministic and randomized update times of $O(f \log f)$ and $O(f \cdot \log^* f)$, respectively [9]. More relevant to this work are the greedy-based algorithms [14, 27, 28], which provide $(1+\epsilon)\ln n$-approximate MSC with update time $O(f \log n)$.

By adapting the greedy-based dynamic MSC algorithm of [14], Hjuler et al. [17] showed that an $O(\log n)$-approximate MDS can be maintained with update time $O(\Delta \log n)$ in unweighted graphs, where Δ is an upper bound on the maximum degree throughout the update sequence. Solomon and Uzrad [27] (see also [28]) designed refined greedy-based dynamic algorithms for MSC and MDS in weighted instances; for weighted MDS, they achieved $(1+\epsilon)\ln\Delta$-approximation with update time $O(\Delta \log n)$. This yields an update time of $\tilde{O}(1)$ in bounded-degree graphs, where we use the notation $\tilde{O}$ to suppress $\mathbf{poly}(\log n)$ terms. However, the question of whether one can replace the dependence on Δ by α and obtain $\tilde{O}(\alpha)$ update time has remained elusive, **even for forests** and **unweighted instances**.

▶ **Question 1.** Can one maintain an $O(\alpha)$-approximate MDS with $\tilde{O}(\alpha)$ update time?

1.1 Our Contribution

In this work we prove the following theorem, which resolves Question 1 in the affirmative.

▶ **Theorem 1.** *There is a deterministic algorithm that maintains an $O(\alpha)$-approximate MDS with amortized update time $O(\alpha \log(Cn))$ in any dynamic graph of arboricity bounded by α.*

Remarks

- Our result replaces the dependence on Δ in the update time $O(\Delta \log n)$ of the dynamic MDS algorithms [14, 27, 28] by a dependence on α, but it incurs a logarithmic dependence on the aspect ratio C. We remind that $\alpha \leq \Delta$ in any graph.
- Our approach yields an approximation factor that approaches $6\alpha + 2$. While we can reduce the leading constant 6 with an extra effort, our approach cannot reduce it all the way to 1 (as done in the static case [29, 13]); this remains an intriguing open problem.

1.2 Perspective

In contrast to MDS, for several fundamental problems, including maximal matching (MM) and maximal independent set (MIS), and approximate maximum matching, minimum vertex cover and maximum independent set, there exist dynamic algorithms for graphs with arboricity bounded by α with update time $\mathrm{poly}(\alpha, \log n)$. At a high level, these results are typically obtained by reducing bounded-arboricity graphs to bounded-degree graphs, using either *low outdegree orientations* or *bounded-degree sparsifiers*, as we next discuss.

[1] For brevity, we will assume in this section that ϵ is a small *constant*.

MM and MIS

For MM and MIS, the known reductions [23, 16, 19, 10, 24] are based on dynamically maintaining a *low outdegree orientation*, i.e., an orientation of the edges of the graph in which the maximum outdegree is close to the graph's arboricity α. Dynamic algorithms for low outdegree orientations have been well studied [8, 16, 19, 4, 10], and it is long known that an outdegree of $O(\alpha)$ can be maintained with $O(\log n)$ update time [8]. Once each vertex has outdegree $O(\alpha)$, one can simulate the corresponding "bounded-degree algorithms" (i.e., the dynamic algorithms that perform efficiently on bounded-degree graphs) in bounded-arboricity graphs, by cleverly restricting attention to out-neighborhoods rather than full neighborhoods, in order to reduce the Δ-dependencies in the update time bounds to α-dependencies.

Approximate problems

More relevant to MDS than (exact) MM and MIS are the problems of *approximate* maximum matching, minimum vertex cover, and maximum independent set. In these problems, the key tool (used sometimes implicitly) for the reductions is that of *bounded-degree sparsifiers* [25, 26], combined with the basic *periodic rebuilding* approach [15]. The bounded-degree sparsifiers [25, 26] rely on simple local rules and can be maintained efficiently in dynamic graphs of bounded arboricity. In the matching sparsifier of [26], each vertex marks $\Delta = O(\alpha/\varepsilon)$ arbitrary neighbors, and only edges marked by both endpoints are included in the sparsifier; the resulting subgraph has a maximum degree at most Δ and it preserves the maximum matching size to within a factor of $1 + \epsilon$. Similarly, for $(1 + \epsilon)$-approximate vertex cover sparsification, all vertices of degree at least $\Delta = O(\alpha/\varepsilon)$ are included in the solution, and the problem is reduced to the induced subgraph on the vertices of degree $< \Delta$; independent set sparsification is symmetric to vertex cover sparsification.

MDS

The MDS problem appears to be significantly more challenging to dynamize.

For the problems above, the dynamic bounded-degree algorithms are simple. To maintain an MM dynamically, it suffices to scan the neighborhoods of the at most two updated vertices after each edge update, yielding a trivial $O(\Delta)$ update time algorithm. Using a dynamic low out-degree orientation as a black-box, extending this algorithm to bounded-arboricity graphs is straightforward. For dynamic MIS, the bounded-degree algorithm is also simple and local, though the extension to bounded-arboricity graphs requires a nontrivial effort. For the approximate problems mentioned above, the dynamic bounded-degree algorithms are obtained by combining the periodic rebuilding approach with simple structural reductions, and extending them to bounded-arboricity graphs can be done in a straightforward manner using the bounded-degree sparsifiers of [25, 26].

For MDS, however, the dynamic bounded-degree algorithms are much more complex than the algorithms for the other problems above. We focus on the dynamic PD algorithms [5, 6, 7, 28, 9], since even in static graphs, all known $O(\alpha)$-approximation algorithms [3, 21, 29, 13] are based on linear programming (and especially PD) techniques. (We note that the algorithm of [21] can also be cast as a PD algorithm, although it is not presented as such.) These dynamic PD algorithms provide $(1 + \epsilon)f$-approximate MSC with update time $O(f \log n)$, and are directly translated to provide $(1 + \epsilon)(\Delta + 1)$-approximate MDS with update time $O(\Delta \log n)$. Importantly, our goal is to replace the Δ-dependence in *both* the approximation factor and the update time by an α-dependence.

These algorithms are organized around maintaining, or periodically restoring, the underlying *complementary slackness conditions*. That is, some of the algorithms [5] maintain these conditions locally at all times, while others [6, 7, 28, 9] allow the constraints to be violated as long as the overall approximation is in check, and they fix the violated constraints periodically, as part of *global rebuild* procedures. Both the local fixing rules and the global rebuild procedures rely heavily on scanning entire "neighborhoods" of vertices or elements (in the set cover problem), in order to maintain or restore the complementary slackness conditions. Alas, such scans incur at least a linear dependence on Δ in the update time.

To translate the dependence on Δ in the update time to an α-dependence, scanning entire neighborhoods is no longer feasible. Instead, to achieve an update time proportional to α rather than Δ, one must work with only an $O(\alpha)$-sized subset of each neighborhood. This creates an immediate basic obstacle: in the MDS problem, even a single neighbor can be crucial (e.g., a vertex that uniquely dominates a large portion of the graph), and there is no known simple local rule, analogous to those used for matching, vertex cover or independent set, that allows one to safely ignore most neighbors.

This basic obstacle creates several highly nontrivial algorithmic and analytical challenges. These challenges become even more significant, since our goal is also to reduce the approximation from $O(\Delta)$ to $O(\alpha)$ while maintaining fast update time. We discuss these challenges and our approach in more detail in Section 2 and Section 3.

1.3 Organization

In Section 2 we review the primal-dual framework for MDS, focusing on prior dynamic algorithms and static algorithms for bounded-arboricity graphs.

In Section 3 we give a technical overview, highlighting the main challenges in replacing the Δ-dependence by an α-dependence and outlining our key ideas, including lazy maintenance of packing values, sparsification of the rebuild procedure, and the forgetting mechanism.

The complete description of our dynamic algorithm and its amortized runtime analysis, as well as a glossary of notation and conclusions, are deferred to the full version of the paper.

2 Preliminaries and Previous approaches

2.1 The Primal-Dual framework for MDS

We denote by $N[v]$ the *closed neighborhood* of v. For a set $S \subseteq V$, we write $N[S] = \bigcup_{v \in S} N[v]$ for the set of vertices dominated by S. Let $L = \lceil \log_{1+\epsilon}(Cn) \rceil + 1$. Let τ_v be the cost of a cheapest neighbor of v, i.e. $\tau_v = \min_{u \in N[v]} c_u$. Let $\epsilon \in (0, 1/3)$.

Our algorithm is a direct adaptation of the primal-dual framework of [5, 6, 7] from MSC to MDS, and we present the definitions only for MDS for brevity.

For the MDS problem, each vertex v is assigned a *packing value* $x_v \geq 0$, and the weight of a vertex v is $\omega(v) = \sum_{u \in N[v]} x_u$. A packing is feasible if $\omega(v) \leq c_v$ for every vertex v. Let OPT denote the minimum cost of any dominating set. For any $S \subseteq V$, let $c(S) = \sum_{v \in S} c_v$.

The following lemma is the MDS analogue of the corresponding lemma for MSC, used in all previous dynamic primal-dual algorithms [5, 6, 7, 9].

▶ **Lemma 2.** *Consider any feasible fractional packing $\{x_v\}_{v \in V}$ for MDS. Then $\sum_{v \in V} x_v \leq OPT$. In addition, if there is a dominating set $D \subseteq V$ and $\rho \geq 1$ such that $\omega(v) \geq c_v/\rho$ for all $v \in D$, then $c(D) \leq \rho(\Delta + 1)OPT$.*

Proof. Let D^* be an optimal dominating set. Then

$$\sum_{v\in V} x_v \le \sum_{u\in D^*}\sum_{v\in N[u]} x_v = \sum_{u\in D^*}\omega(u) \le \sum_{u\in D^*} c_u = OPT.$$

For the second claim,

$$c(D) \le \rho\sum_{v\in D}\omega(v) \le \rho(\Delta+1)\sum_{u\in V} x_u \le \rho(\Delta+1)OPT,$$

since each x_u appears in at most $\Delta+1$ closed neighborhoods. ◀

A fractional packing and a dominating set satisfying the conditions of Lemma 2 can be obtained by the classic *water-filling* primal-dual algorithm. For efficiency, this algorithm is *discretized* by introducing a *hierarchical partition* of vertices into levels. Specifically, each vertex v is assigned a *dominating level* $\mathrm{dom}(v) \in [L]$, and the *dominated level* of v is defined as $\mathrm{Dom}(v) = \max_{u\in N[v]} \mathrm{dom}(u)$; note that $\mathrm{Dom}(v) \ge \mathrm{dom}(v)$. The packing value of v is then given by $x_v = (1+\epsilon)^{-\mathrm{Dom}(v)}$. The water-filling algorithm determines the levels of vertices as follows. It starts by assigning $\mathrm{dom}(v) = L$ for all vertices v, and it gradually decreases the dominating levels, stopping for a vertex v once $\omega(v) > c_v/(1+\epsilon)$ or $\mathrm{dom}(v) = \mathsf{base}(v)$, where $\mathsf{base}(v) = \left\lfloor \log_{1+\epsilon} \frac{1}{c_v} \right\rfloor$ is the *base level* of v. Define by $\mathsf{bot}(v) = \left\lfloor \log_{1+\epsilon} \frac{1}{\tau_v} \right\rfloor$ the *bottom level* of v; note that $\mathsf{bot}(v) = \max_{u\in N[v]} \mathsf{base}(u)$.

Let $T \subseteq V$ be the collection of vertices v satisfying $\omega(v) > c_v/(1+\epsilon)$. We call vertices satisfying this condition *tight*; other vertices are called *slack*. We note that every vertex v with $\mathrm{Dom}(v) = \mathsf{bot}(v)$ has a neighbor from T, since $x_v \ge \tau_v$. Thus, T is a dominating set that satisfies the conditions of Lemma 2, and so T is a $(1+\epsilon)(\Delta+1)$-approximate MDS.

2.2 Previous Dynamic Primal-Dual Algorithms

Previous dynamic primal-dual algorithms for the MDS problem are obtained by dynamizing the WaterFilling subroutine described above. In particular, the dynamic algorithms of [6, 7, 9] all use, as a basic rebuilding primitive, the following MDS adaptation of the water-filling algorithm from [6].

▶ **Lemma 3** (From [6], adapted for MDS). *There exists a subroutine* WaterFilling *with the following properties.*[2] *The subroutine* WaterFilling(k, D', V') *takes as input two collections of vertices D' and V' such that* $\mathrm{dom}(v) = k$ *for every $v \in D'$ and* $\mathrm{Dom}(u) = k$ *for every $u \in V'$. Moreover, each vertex $v \in D'$ satisfies $\omega(v) \le c_v$. The subroutine updates the dominating levels of vertices in D' and the dominated levels of vertices in V' so that: (1) $\omega(v) \le c_v$ for every $v \in D'$, and (2) if* $\mathrm{dom}(v) > \mathsf{base}(v)$*, then $\omega(v) > c_v/(1+\epsilon)$ for every $v \in D'$. The subroutine runs in time $O\big(k + |D'| + \sum_{u\in V'} |N[u]|\big)$.*

The algorithm of [6] is based on a global rebuild strategy: rather than maintaining a solution that always satisfies the conditions of Lemma 2, it keeps the current solution as long as its cost remains sufficiently close to $(\Delta+1)\sum_{v\in V} x_v$. When the algorithm determines that the solution might deviate significantly from this target, it chooses an appropriate level k, lets $D_{\le k}$ denote the set of vertices v with $\mathrm{dom}(v) \le k$ and $V_{\le k}$ denote the set of vertices v with $\mathrm{Dom}(v) \le k$, raises every vertex in $D_{\le k}$ to dominating level k, raises every vertex in $V_{\le k}$ to level k, and then invokes the subroutine from Lemma 3. The algorithms of [7, 9] follow

[2] In [6], this subroutine is called FixLevel.

essentially the same global rebuild paradigm, still using Lemma 3 as the core rebuilding step, while additionally applying local fixing rules during updates to improve the update-time guarantees.

This rebuild paradigm is exactly what causes the linear dependence on Δ in the update time of previous dynamic MDS algorithms. Indeed, when Lemma 3 is applied as described above, its running time is proportional to $k+|D_{\leq k}|+\sum_{u\in V_{\leq k}}(\deg(u)+1)$, since the subroutine must inspect the entire neighborhood of each processed vertex. Thus, rebuilding a prefix of the level hierarchy incurs a cost proportional to the sum of the degrees of the involved vertices. In bounded-degree graphs, this yields efficient update times, but in general graphs it necessarily introduces a linear dependence on Δ.

The same issue arises in the local fixing rules. More generally, all previous dynamic MDS algorithms rely on scanning the full neighborhoods of vertices, both in rebuild subroutines and in local updates. Thus, their update-time bounds inherently depend linearly on Δ.

2.3 Static Primal-Dual Algorithms in Bounded Arboricity Graphs

Static $O(\alpha)$-approximation algorithms for MDS on graphs of arboricity at most α were obtained in a sequence of works [3, 21, 29, 13]. Our algorithm builds on the distributed algorithm and analysis of Dory et al. [13], which exploit low-outdegree orientations of bounded-arboricity graphs.

The following lemma and claim are direct consequences of the work of Dory et al. [13].

▶ **Lemma 4.** *There exists a static algorithm that computes a feasible packing $\{x_v\}_{v\in V}$ such that, with H defined as the set of vertices not dominated by T and $\lambda = \frac{1}{(1+\epsilon)^2(2\alpha+1)}$, the following hold: (1) $c(T) \leq (1+\epsilon)(2\alpha+1)\sum_{v\in V\setminus H} x_v$; (2) $x_v \geq \lambda\tau_v$ for every $v \in H$; (3) $x_v \leq (1+\epsilon)\lambda\tau_v$ for every $v \in V$.*

The proof of this lemma uses the following observation.

▶ **Observation 5.** *The edges of any graph G of arboricity at most α can be oriented so that every vertex has out-degree at most α.*

Proof. Since G has arboricity at most α, its edges can be partitioned into at most α edge-disjoint forests. The edges of a forest can be oriented so that each vertex has at most one outgoing edge. This can be done by fixing a root vertex in each tree of the forest and orienting every edge toward this root. Thus, applying such an orientation to each forest in the partition yields the required orientation. ◀

Proof of Lemma 4. We run the water-filling algorithm with the following modification. We *scale down the packing values by a factor of* λ, redefining the packing value of v to be $x_v = \lambda(1+\epsilon)^{-\mathrm{Dom}(v)}$. Therefore, the resulting packing satisfies $x_v \leq (1+\epsilon)\lambda\tau_v$. Every vertex with $\mathrm{dom}(v) > \mathsf{base}(v)$ in the resulting packing still has $\omega(v) > c_v/(1+\epsilon)$, and so v is tight. Thus, every vertex v with $\mathrm{Dom}(v) > \mathsf{bot}(v)$ is dominated by T. However, we cannot claim that vertices with $\mathrm{Dom}(v) = \mathsf{bot}(v)$ are dominated by T, since we scaled down the packing values. But for these vertices we have $x_v \geq \lambda\tau_v$.

Consider the orientation from Observation 5. Then

$$\omega(v) = \sum_{u\in N^{in}[v]} x_u + \sum_{u\in N^{out}(v)} x_u \leq \sum_{u\in N^{in}[v]} x_u + (1+\epsilon)\alpha\lambda c_v. \tag{1}$$

Summing over $v \in T$ and swapping the order of summation gives

$$\sum_{v\in T}\sum_{u\in N^{in}[v]} x_u = \sum_{u\in V\setminus H} x_u \cdot |\{v \in T : v \in N^{out}[u]\}| \leq (\alpha+1)\sum_{v\in V\setminus H} x_v. \tag{2}$$

For a tight vertex $v \in T$, we have $c_v < (1+\epsilon)\omega(v)$. Summing over T and using Equations (1) and (2) yields

$$\begin{aligned} c(T) &\leq \sum_{v\in T}(1+\epsilon)\omega(v) \\ &\leq (1+\epsilon)\sum_{v\in T}\Big(\sum_{u\in N^{in}[v]} x_u + (1+\epsilon)\alpha\lambda c_v\Big) \\ &\leq (1+\epsilon)(\alpha+1)\sum_{v\in V\setminus H} x_v + \frac{\alpha}{2\alpha+1}\sum_{v\in T} c_v, \end{aligned}$$

hence $c(T) \leq (1+\epsilon)(2\alpha+1)\sum_{v\in V\setminus H} x_v$. ◀

Let H' be the set of cheapest neighbors of the vertices in H from Lemma 4 (for each vertex, we take a single neighbor). Then we make the following claim.

▷ Claim 6. The set $T \cup H'$ from Lemma 4 forms a $(1+3\epsilon)(2\alpha+1)$-approximate MDS.

Proof. Since $x_v \geq \lambda\tau_v = \frac{\tau_v}{(1+\epsilon)^2(2\alpha+1)}$ for $v \in H$, we have

$$c(H') = \sum_{v\in H}\tau_v \leq (1+\epsilon)^2(2\alpha+1)\sum_{v\in H} x_v.$$

Therefore, $c(T\cup H') \leq (1+3\epsilon)(2\alpha+1)\sum_{v\in V} x_v \leq (1+3\epsilon)(2\alpha+1)OPT$. ◁

The algorithm from [6] waits until the cost of the solution deviates significantly from $(\Delta+1)\sum_{v\in V} x_v$. However, if we aim for an $O(\alpha)$ approximation ratio, we need it to be close to $\alpha\sum_{v\in V} x_v$. Thus, we might have to rebuild the solution much more frequently, and this requires us to reduce the runtime cost of the rebuild subroutine significantly, from $O(\Delta)$ per vertex to $O(\alpha)$.

Another issue in applying the algorithm from [6] is keeping track of the vertices that are *undominated* by T. However, since our goal is an $O(\alpha)$ approximation, we can simply define H to consist of *all* vertices v such that $x_v \geq \lambda\tau_v$ (which is a superset of the original H from Lemma 4). This may cause H' and T to intersect, but the approximation ratio would increase by at most a factor of 2.

3 Technical Overview

3.1 Overview of the Dynamic Algorithm

At a high level, our algorithm combines the static distributed algorithm for arboricity-α graphs from [13] with the dynamic primal-dual algorithms of [6]. The main obstacle is that combining these approaches directly leads to an update time that depends at least linearly on Δ. The static distributed bounded-arboricity analysis yields the desired $O(\alpha)$ approximation, but the known dynamic primal-dual algorithms rely crucially on scanning full neighborhoods, which costs $\Omega(\Delta)$ time. Our task is therefore to preserve the primal-dual structure while replacing all essential Δ-dependent operations by α-dependent ones. To achieve $O(\alpha)$ approximation as in the distributed algorithm, additional constraints must be enforced, beyond the usual constraints of the primal-dual dynamic algorithms. Maintaining these additional constraints is essential for achieving an $O(\alpha)$-approximation, but it makes it significantly more difficult to obtain fast update time. In the following subsections, we summarize the main obstacles and our approach to overcoming them. We begin by explaining how to avoid scanning entire neighborhoods during updates, and then describe how to sparsify the rebuild subroutine to obtain an α-dependent update time.

We use the framework of [6], which distinguishes between *active* and *passive* vertices. Each vertex is assigned a *level* $\mathsf{lev}(v) \geq \mathrm{Dom}(v)$, and the packing value is $x_v = \lambda(1+\epsilon)^{-\mathsf{lev}(v)}$. We use $\lambda = \frac{1}{(1+\epsilon)^2(3\alpha+1)}$, in contrast to the definition in Lemma 4. This modified choice of λ enables faster update time at the cost of increasing the approximation ratio by a constant factor, via the mechanism of "forgetting" vertices, explained later. For an active vertex, we maintain its dominated level $\mathrm{Dom}(v)$ exactly in $\mathsf{lev}(v)$. For a passive vertex v, we have $\mathsf{lev}(v) > \mathrm{Dom}(v)$.

Passive vertices introduce an additional logarithmic dependency in the update time. This is because each passive vertex may need to be processed multiple times before becoming active. The key point is that whenever a passive vertex is processed by the rebuild subroutine, the gap $\mathsf{lev}(v) - \mathsf{zlev}(v)$ decreases (we discuss the lazy level $\mathsf{zlev}(v)$ later; intuitively it acts like a lazy version of $\mathrm{Dom}(v)$). Consequently, unless an incident edge update occurs first, any passive vertex can participate in at most $O(\log_{1+\epsilon}(Cn))$ rebuilds before it becomes active.

To simplify this overview, for the rest of this section, we assume that there are no passive vertices (unless stated otherwise). This allows us to convey the main ideas more effectively. However, dealing with passive vertices is a highly nontrivial challenge and we defer this to the full version. Thus, throughout this overview we assume $\mathsf{lev}(v) = \mathrm{Dom}(v)$.

Following the static algorithm from Lemma 4, we maintain the following invariant.

▶ **Invariant 1** (the tightness invariant). *Every vertex v with $\mathrm{dom}(v) > \mathsf{base}(v)$ is tight.*

The notion of tightness is defined later in Definition 9. Intuitively, a vertex v is tight if its weight is close to c_v. Let T be the set of all tight vertices, and H be all vertices with $\mathsf{lev}(v) = \mathsf{bot}(v)$; the definition of H' remains the same as after Lemma 4. The dominating set our algorithm maintains is $T \cup H'$ (we defer the proof that this is indeed a dominating set to the full version). We show by Lemma 7 this is an $O(\alpha)$-approximate MDS.

Due to scaling down the packing values by a factor of λ and Invariant 1 we have the following bound on the packing values $x_v \leq (1+\epsilon)\lambda\tau_v$. This is different from the previous primal-dual dynamic algorithms, where the bound was $x_v \leq (1+\epsilon)\tau_v$ (or simply $x_v \leq 1$). This bound on the packing values plays a crucial role in achieving the required bounds for both the approximation factor and the update time; we explain it in the following sections.

3.1.1 Handling Deletions without Full Neighborhood Scans

In the standard framework, when an edge disappears, the dominated level $\mathrm{Dom}(v)$ of a vertex v may decrease, and restoring the exact value of $\mathrm{Dom}(v)$ may again require scanning the entire neighborhood $N[v]$.

We handle that issue by keeping a *lazy level* $\mathsf{zlev}(v) \leq \mathrm{Dom}(v)$ and not tracking $\mathrm{Dom}(v)$ exactly for passive vertices, in the spirit of [9]. Thus, whenever an edge is deleted, instead of scanning the neighborhood to compute $\mathrm{Dom}(v)$, we reset $\mathsf{zlev}(v)$ to 0, which enables us to handle deletion in constant time. Later, when v is processed in a rebuild subroutine, either its lazy level is increased (and so gets closer to $\mathrm{Dom}(v)$), or v becomes active. Thus, intuitively, we defer scanning the neighborhood for v to global rebuilds.

We note that in our case, unlike [9], $\mathsf{zlev}(v)$ cannot be any level $\leq \mathrm{Dom}(v)$, since not every vertex with $\mathrm{dom}(v) > 0$ is taken to the solution. This issue is handled by the lazy level invariant, which guarantees that at least one neighbor is tight and thus is taken to the solution.

3.1.2 Handling Insertions without Full Neighborhood Scans

In previous dynamic primal-dual algorithms, when $\mathrm{Dom}(v)$ increases due to an edge insertion, one has to explicitly update the weights of all vertices in $N[v]$. This is already problematic after a single edge insertion. Indeed, if inserting an edge causes $\mathrm{Dom}(v)$ to increase, then the packing value x_v decreases, and updating the weights $\omega(u)$ of all vertices $u \in N[v]$ may require scanning the entire neighborhood of v, leading to $\Omega(\Delta)$ update time.

To avoid this, we maintain weights approximately using a lazy update scheme. One source of error is the *deviation*, denoted by δ, which captures the discrepancy between the true weights and the maintained lazy weights due to outdated information about neighbors' levels. Another is *dead weight* $\phi(v)$, which we add after deletions to temporarily compensate for lost weight; let $\phi = \sum_{v \in V} \phi(v)$.

Throughout the algorithm we bound the total deviation by Invariant 4 provided later. As long as this bound holds, the approximation degrades by only a factor of $1 + O(\epsilon)$. This is formally captured by the following Lemma 7, the proof of which we defer to the full version.

First we explain the concept of viewed levels and how they allow us to avoid scanning the neighborhoods on edge insertions, and then show how this relates to δ and the resulting approximation guarantee.

In the algorithm, we do not maintain the true weight $\omega(v) = \sum_{u \in N[v]} x_u$ explicitly. Instead, for every vertex v and every neighbor $u \in N[v] \setminus F(v)$, we maintain a *viewed level* $\widehat{\mathsf{lev}}_v(u)$, which represents the level at which v currently views u. The set $F(v)$ is the set of forgotten neighbors, which we define later in Section 3.1.3. The corresponding *viewed value* is then defined by $\hat{x}_u(v) = \lambda(1+\epsilon)^{-\widehat{\mathsf{lev}}_v(u)}$. Thus each vertex v stores a *viewed weight* $\hat{\omega}(v) = \sum_{u \in N[v] \setminus F(v)} \hat{x}_u(v)$, where always $\hat{x}_u(v) \geq x_u$. This inequality is enforced by maintaining the following invariant.

▶ **Invariant 2** (the view invariant). *For every vertex v and every $u \in N[v] \setminus I(v)$, we have* $\mathrm{dom}(u) \leq \widehat{\mathsf{lev}}_u(v) \leq \mathsf{lev}(v)$.

We do not maintain this bound for a set of ignoring vertices $I(v)$, since for them it holds trivially; we explain this later when we introduce forgotten vertices in Section 3.1.3. In particular, this implies $\omega(v) \leq \hat{\omega}(v) + |F(v)| \cdot (1+\epsilon)\lambda c_v$, and hence if $\hat{\omega}(v) \leq c_v - |F(v)| \cdot (1+\epsilon)\lambda c_v$ for every vertex v, then the packing is feasible. The last part is enforced by the following invariant.

▶ **Invariant 3** (the bounded weight invariant). *For every vertex v we have $\hat{\omega}(v) \leq c_v - |F(v)| \cdot (1+\epsilon)\lambda c_v$.*

Next, let us explain how we maintain the bound $\widehat{\mathsf{lev}}_u(v) \leq \mathsf{lev}(v)$. The key point is that during insertions, we may only increase $\mathsf{lev}(v)$, while deletions leave $\mathsf{lev}(v)$ unchanged; $\mathsf{lev}(v)$ may decrease only during rebuilds, but whenever we decrease $\mathsf{lev}(v)$ during rebuild, this is when we update all $\widehat{\mathsf{lev}}_u(v)$ to the actual value $\mathsf{lev}(v)$. Therefore, upon insertions and deletions we can safely let all neighbors $u \in N[v] \setminus I(v)$ keep their old $\widehat{\mathsf{lev}}_u(v)$.

To show how this relates to δ, define the *deviation for a vertex v* as

$$\delta(v) = \max_{u \in N[v] \setminus I(v)} (\hat{x}_v(u) - x_v).$$

Thus, whenever x_v drops due to an insertion, $\delta(v)$ increases. Then $\delta = \sum_{v \in V} \delta(v)$. The actual $\delta(v)$ we use in the algorithm is defined a bit differently to simplify the amortized analysis, but these definitions differ up to a constant factor. Whenever $\mathsf{lev}(v)$ increases due to an insertion, the deviation $\delta(v)$ increases. Conversely, rebuild subroutines reduce deviation by restoring consistency between viewed levels and true levels.

As deviation and dead weight accumulate, the approximation guarantee deteriorates. To control this, we maintain a global invariant that bounds their total contribution. Recall that T is the set of tight vertices.

▶ **Invariant 4** (the bounded δ, ϕ invariant). $(\alpha+1)\delta+\phi \le \epsilon\Big(\xi c(T)+(\alpha+1)\sum_{v\in V} x_v\Big)$, *where* $\xi = \frac{\alpha+1}{(1+\epsilon)(3\alpha+1)}$.

In Lemma 7 we show that when the invariants are maintained the approximation factor stays close to $6\alpha+2$. Conversely, whenever Invariant 4 is violated, we claim that we accumulated enough credits to afford a rebuild subroutine, which reduces δ and ϕ (the full analysis is provided in the full version).

▶ **Lemma 7.** *If all the invariants hold, then* $c(T)+c(H') \le (1+5\epsilon)\cdot 2(3\alpha+1)\cdot OPT$.

We defer the proof of this lemma to the full version. The proof is similar to Lemma 4, but also accounts for deviation and dead weight, and forgotten neighbors, which we define later.

3.1.3 Improving the Runtime of Rebuild Via Sparsification

The main bottleneck in previous dynamic primal-dual algorithms is the rebuild step. The algorithm chooses a level k and rebuilds all vertices at level k or below, using the WaterFilling subroutine from Lemma 3 as the main ingredient. In bounded-degree graphs this is acceptable, because scanning all relevant neighborhoods costs only $O(\Delta)$ per processed vertex. In our setting, however, we cannot afford to inspect every neighbor of every processed vertex.

Main idea: WaterFilling sparsification

Instead of applying the WaterFilling subroutine on the entire subgraph induced by $V_{\le k}$, which may have size $\Theta(|V_{\le k}|\Delta)$, we construct a sparsified subgraph of size roughly $O(|V_{\le k}|\alpha)$ and run WaterFilling only on this subgraph. We show that this sparsification preserves the approximation guarantee up to a constant factor.

To explain how we handle this, we will provide a simplified rebuild subroutine that works under the assumption that all vertices are active (recall that we assumed this in previous sections as well), since passive vertices introduce additional challenges. An additional simplifying assumption is that $F(v) \subseteq V_{\le k}$. In the actual algorithm, this might not be the case. We handle that by a more intricate potential analysis that allows us to charge previous rebuilds part of the costs. Then we provide the amortized update time analysis of this simplified subroutine. The full subroutine and its analysis are provided in the full version.

Reducing the degree in Rebuild

Consider a rebuild at level k. A natural obstacle is that we can afford to scan or process only $O(\alpha)$ (ignoring the dependency on ϵ) relevant neighbors (in $D_{\le k} \setminus V_{\le k}$) per vertex in $V_{\le k}$, but the number of relevant neighbors per vertex in $V_{\le k}$ could be Δ rather than $O(\alpha)$. The key structural point is that there cannot be too many vertices in $D_{\le k} \setminus V_{\le k}$ that have many neighbors inside $V_{\le k}$.

▶ **Lemma 8.** *Let B denote the set of vertices $u \in D_{\le k} \setminus V_{\le k}$ such that $|N[v] \cap V_{\le k}| > (1+\epsilon)\alpha$. Then $|B| \le \frac{1}{\epsilon}|V_{\le k}|$.*

Proof. Every vertex of B contributes $> (1+\epsilon)\alpha$ edges to the bipartite graph induced by B and $V_{\le k}$, thus this graph contains $> (1+\epsilon)\alpha|B|$ edges. On the other hand, since the graph has arboricity $\le \alpha$, every induced subgraph on $|B|+|V_{\le k}|$ vertices has at most $\alpha(|B|+|V_{\le k}|)$ edges. Therefore, $(1+\epsilon)\alpha|B| < \alpha(|B|+|V_{\le k}|)$, which implies $\epsilon|B| < |V_{\le k}|$, as claimed. ◀

Thus, all vertices in $D_{\leq k} \setminus V_{\leq k}$ that have more than $(1+\epsilon)\alpha$ neighbors in $V_{\leq k}$ form only a small exceptional set of size $O(|V_{\leq k}|/\epsilon)$. Further, we will show that it suffices to focus only on edges between $V_{\leq k}$ and this exceptional set. Hence, the total number of such edges would be $O\big(\frac{\alpha}{\epsilon}|V_{\leq k}|\big)$; equivalently, the average number of such relevant neighbors per vertex of $V_{\leq k}$ is $O(\alpha/\epsilon)$. In that case, the rebuilding step, and in particular the call to the WaterFilling subroutine from Lemma 3 (which deliberately ignores neighbors of vertices in $V_{\leq k}$ that lie outside of $V_{\leq k} \cup B$), would run in time $O\big(k + \frac{\alpha}{\epsilon} \cdot |V_{\leq k}|\big)$, instead of $O(k + \Delta \cdot |V_{\leq k}|)$. We will later reason that these neighbors can be ignored, and this will have only a small effect on the approximation factor; but it does increase the leading constant in the approx factor.

Forgetting vertices

Now we explain the mechanism that allows us to exclude vertices with only a few neighbors in $V_{\leq k}$ from the rebuilding process by "forgetting" them. We refer to this process as *forgetting* neighbors. The key point is that such vertices contribute only a small amount to the weights, and therefore their effect can be safely approximated.

A key property (by Lemma 4) is that the packing values satisfy $x_v \leq (1+\epsilon)\lambda\tau_v$. Thus the total contribution of any $(1+\epsilon)\alpha$ neighbors to the weight of v is at most $(1+\epsilon)^2\alpha\lambda\tau_v \leq \frac{\alpha}{3\alpha+1}c_v$.

Consider a vertex v with $\leq (1+\epsilon)\alpha$ neighbors in $V_{\leq k}$, and let $\omega'(v)$ be the weight of v without these neighbors. If $\omega'(v) \leq \frac{2\alpha+1}{3\alpha+1}c_v$, then even after adding back the contribution of these neighbors, the total weight of v remains at most c_v. Therefore, from the perspective of the water-filling process, these neighbors can be ignored without violating feasibility. To make use of this, we relax the notion of tightness. With this relaxed definition, the approximation factor increases only by a constant factor from the original factor of $(1+\epsilon)(2\alpha+1)$ of [13].

▶ **Definition 9.** *A vertex is called* tight *if* $\hat{\omega}(v) + \phi(v) > \frac{2\alpha+1}{(1+\epsilon)(3\alpha+1)}c_v$*; otherwise, it is called* slack.

The gap between $\frac{2\alpha+1}{(1+\epsilon)(3\alpha+1)}c_v$ and c_v is what allows a slack vertex to forget a bounded number of neighbors without violating feasibility.

To implement this idea, each vertex v maintains a set $F(v)$ of *forgotten* neighbors. If $u \in F(v)$, then u stops contributing to $\hat{\omega}(v)$. Forgotten neighbors do not create outdated copies, and therefore they do not contribute to the deviation. We bound the number of forgotten neighbors with the following invariant.

▶ **Invariant 5** (the forgotten neighbors invariant)**.** *For every vertex* v*, we have* $|F(v)| \leq (1+\epsilon)\alpha$*.*

Symmetrically, from the perspective of u, if $u \in F(v)$, then v is called an *ignoring* neighbor of u. We denote by $I(u)$ the set of neighbors that currently ignore u.

Thus, with our relaxed definition of tightness, a slack vertex can safely forget up to $(1+\epsilon)\alpha$ neighbors without violating Invariant 3. This is the key step that allows us to eliminate most low-impact edges from the subgraph on which the rebuild subroutine operates.

A simplified rebuild subroutine

The goal of the rebuild subroutine is to restore Invariant 4 by cleaning up all deviation and dead weight up to level k, and then to restore Invariant 1 by running the WaterFilling subroutine. In our simplified subroutine, we omit the details of how this cleanup is performed, and simply assume that it can be done in constant time. In the actual algorithm, it is done via forgetting some vertices outside $V_{\leq k}$, and thus requires a more careful analysis. All this is explained in the full version in the appendix.

The pseudocode of our simplified rebuild subroutine is provided in Algorithm 1. At a high level, the rebuild subroutine operates as follows. Let T' be the vertices in $D_{\leq k}$ that were tight at the beginning of the rebuild subroutine, and let $V' = V_{\leq k}$. First, the subroutine cleans up deviation and dead weight up to level k. Then, for every vertex $v \in V_{\leq k}$, it attempts to forget v for all $u \in N[v] \setminus I(v)$. This is implemented using auxiliary sets of "need-to-be-forgotten" vertices $F'(v)$, which track which vertices request to be forgotten. More precisely, for each vertex u, the set $F'(u)$ will store all vertices v that request u to forget them. As a result, some vertices u might accumulate more than $(1+\epsilon)\alpha - |F(v)|$ requests in $F'(u)$. Let S be the set of such vertices. The vertices in S correspond to the high-degree vertices from Lemma 8, namely those that cannot safely forget all their incident requests. For these vertices, and the vertices from T', we reset their forgotten neighbor sets by setting $F(v) = \emptyset$. After that, we invoke the WaterFilling subroutine on the vertices in V', considering only neighbors from $N[v] \setminus I(v)$ for each $v \in V'$. Due to the forgetting step, all such relevant neighbors belong to $S \cup T'$, and thus the subroutine operates only on this sparsified structure. The complete description of our actual rebuild subroutine is provided in the full version.

Thus, the rebuild subroutine reduces the instance to one in which every relevant edge is incident to a vertex in $S \cup T'$, and the total number of such edges is $O\left(\frac{\alpha}{\epsilon}|V_{\leq k}|\right)$.

Algorithm 1 A simplified version of our rebuild subroutine.

1 Let $T' = T_{\leq k}$ and $V' = V_{\leq k}$;
2 Clean up dead weight and deviation up to level k;
3 **foreach** $v \in V'$ *and* $u \in N[v] \setminus I(v)$ **do** Add v to $F'(u)$;
4 Let S be the set of vertices with $|F(v)| + |F'(v)| > (1+\epsilon)\alpha$;
5 **foreach** $v \in S \cup T'$ **do** $F(v) \leftarrow \emptyset$;
6 **foreach** $v \notin S$ *such that* $F'(v) \neq \emptyset$ **do** $F(v) \leftarrow F(v) \cup F'(v)$;
7 Set $\mathrm{dom}(v) \leftarrow k$ for all $v \in S \cup T'$ and $\mathrm{Dom}(v) \leftarrow k$ for all $v \in V'$;
8 Call WaterFilling$(k, S \cup T', V')$ considering only $N[v] \setminus I(v)$ for each $v \in V_{\leq k}$;

Let $f_v = |F(v)|$ at the beginning of the rebuild subroutine, and let $f'_v = |F'(v)|$ at the end of it. Next, we prove the following claim about the runtime of the rebuild subroutine.

▷ Claim 10. The runtime of the rebuild subroutine, ignoring the cost of cleaning up deviation up to level k, is $O\left(k + \frac{\alpha}{\epsilon}|V_{\leq k}| + \alpha|T_{\leq k}| + \sum_{v \notin S \cup T_{\leq k}} f'_v\right)$.

Proof. At the moment of the call to WaterFilling, every vertex in V' is incident only on vertices from $S \cup T'$, so $\sum_{v \in V'} |N[v] \setminus I(v)| \leq \sum_{v \in S \cup T'} f_v + f'_v$. Note that we can compute the set S in time $O\left(\sum_{v \in V} f'_v\right)$, since we only need to consider vertices with nonempty $F'(v)$. Observe that the runtime of everything up to the call to WaterFilling can be expressed in terms of f_v and f'_v as $O\left(k + |V_{\leq k}| + \sum_{v \notin S \cup T_{\leq k}} f'_v + \sum_{v \in S \cup T_{\leq k}} f_v + f'_v\right)$.

According to Lemma 8, the size of S can be bounded by $|S| \leq \frac{1+\epsilon}{\epsilon}|V_{\leq k}|$. By Invariant 5, $f_v \leq (1+\epsilon)\alpha$. Thus, $\sum_{v \in S \cup T_{\leq k}} f_v = O(\alpha|S| + \alpha|T_{\leq k}|)$.

Since the graph has arboricity at most α, we have $\sum_{v \in S} f'_v \leq \alpha(|S| + |V_{\leq k}|) = O\left(\frac{\alpha}{\epsilon}|V_{\leq k}|\right)$. Finally, note that for $v \in T_{\leq k} \setminus S$, we have $f'_v \leq (1+\epsilon)\alpha$, and so $\sum_{v \in T_{\leq k} \setminus S} f'_v = O(\alpha|T_{\leq k}|)$. Thus, we obtain the claimed runtime bound. ◁

We note that in the actual algorithm, $F'(v)$ can contain vertices outside $V_{\leq k}$. Because of this, we also bound the runtime in terms of passive vertices and outdated copies, which we do not explain here; these terms are handled in the full analysis in the full version.

We proceed to bound the amortized update time of the rebuild subroutine, and in particular the term $O(\sum_{v \notin S \cup T_{\leq k}} f'_v)$.

Amortized analysis of the simplified rebuild

We analyze the amortized update time using a potential function defined formally in the full version. Here we provide an analysis of our simplified rebuild subroutine. The actual analysis follows the same idea, but has to account for more details, since the actual rebuild subroutine is more complicated. In particular, our analysis here ignores the presence of passive vertices.

The goal is to prove the following lemma.

▶ **Lemma 11.** *The decrease in potential due to rebuild is* $\geq \sum_{v \notin S \cup T_{\leq k}} f'_v + \frac{\alpha}{\epsilon}|V_{\leq k}| + \alpha|T_{\leq k}|$.

This implies that the amortized runtime of our rebuild subroutine is $O(k)$. This $O(k)$ runtime cost can also be charged to the decrease in potential. This is done using the level potential defined in the full version.

First, we lower bound the decrease in potential due to cleaning up deviation up to level k. This decrease can be summarized by the following lemma, whose proof is deferred to the full version. The actual statement there is different and includes some additional terms.

▶ **Lemma 12.** *The decrease in potential due to cleaning up dead weight and deviation up to level* k *is at least* $\frac{5(\alpha+1)}{\epsilon}|V_{\leq k}| + \frac{5(\alpha+1)}{\epsilon}|T_{\leq k}|$.

The proof of this lemma is similar to [7]. However, as the runtime bound from the previous section, unlike [7], this lemma alone is not enough to bound the amortized runtime cost.

We introduce the following potential function, which intuitively allows us to cover the costs associated with forgetting neighbors of v. For each vertex v, we define the *forget potential* as $\Phi_{\text{forget}}(v) = -|F(v)|$. The *total forget potential* is defined as $\Phi_{\text{forget}} = \sum_{v \in V} \Phi_{\text{forget}}(v)$.

Proof of Lemma 11. Consider a vertex $v \notin S \cup T_{\leq k}$. By the algorithm, we set $F(v) \leftarrow F(v) \cup F'(v)$, so $\Phi_{\text{forget}}(v)$ decreases by f'_v. Consider a vertex $v \in S \cup T_{\leq k}$. For v, we set $F(v) \leftarrow \emptyset$, and so $\Phi_{\text{forget}}(v)$ increases by f_v. According to Invariant 5, $f_v \leq (1+\epsilon)\alpha$, which is at most 2α. Finally, from Lemma 8, we have $|S| \leq \frac{1+\epsilon}{\epsilon}|V_{\leq k}| \leq \frac{2}{\epsilon}|V_{\leq k}|$. Thus the increase in the total forget potential is at most $\frac{4\alpha}{\epsilon}|V_{\leq k}| + 2\alpha|T_{\leq k}| - \sum_{v \notin S \cup T_{\leq k}} f'_v$. Summing this with the bound from Lemma 12 yields the claimed bound. ◀

References

1 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. doi:10.1145/3313276.3316376.

2 Sepehr Assadi and Shay Solomon. Fully Dynamic Set Cover via Hypergraph Maximal Matching: An Optimal Approximation Through a Local Approach. In *29th Annual European Symposium on Algorithms (ESA 2021)*, volume 204 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ESA.2021.8.

3 Nikhil Bansal and Seeun William Umboh. Tight approximation bounds for dominating set on graphs of bounded arboricity. *Information Processing Letters*, 122:21–24, 2017. doi:10.1016/J.IPL.2017.01.011.

4 Edvin Berglin and Gerth Stølting Brodal. A simple greedy algorithm for dynamic graph orientation. *Algorithmica*, 82(2):245–259, 2020. doi:10.1007/S00453-018-0528-0.

5 Sayan Bhattacharya, Monika Henzinger, and Giuseppe F Italiano. Design of dynamic algorithms via primal-dual method. In *International Colloquium on Automata, Languages, and Programming*, pages 206–218. Springer, 2015. doi:10.1007/978-3-662-47672-7_17.

6 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. doi:10.1109/FOCS.2019.00033.

7 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. doi:10.1137/1.9781611976465.150.

8 Gerth Stølting Brodal and Rolf Fagerberg. Dynamic representations of sparse graphs. In *Workshop on Algorithms and Data Structures*, pages 342–351. Springer, 1999. doi:10.1007/3-540-48447-7_34.

9 Anton Bukov, Shay Solomon, and Tianyi Zhang. Nearly optimal dynamic set cover: Breaking the quadratic-in-f time barrier. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 824–863. SIAM, 2025. doi:10.1137/1.9781611978322.24.

10 Chandra Chekuri, Aleksander Bjørn Christiansen, Jacob Holm, Ivor van der Hoog, Kent Quanrud, Eva Rotenberg, and Chris Schwiegelshohn. Adaptive out-orientations with applications. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3062–3088. SIAM, 2024.

11 Miroslav Chlebík and Janka Chlebíková. Approximation hardness of dominating set problems in bounded degree graphs. *Information and Computation*, 206(11):1264–1275, 2008. doi:10.1016/J.IC.2008.07.003.

12 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. doi:10.1145/2591796.2591884.

13 Michal Dory, Mohsen Ghaffari, and Saeed Ilchi. Near-optimal distributed dominating set in bounded arboricity graphs. In Alessia Milani and Philipp Woelfel, editors, *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25 - 29, 2022*, pages 292–300. ACM, 2022. doi:10.1145/3519270.3538437.

14 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. doi:10.1145/3055399.3055493.

15 Manoj Gupta and Richard Peng. Fully dynamic (1+ e)-approximate matchings. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 548–557. IEEE, 2013. doi:10.1109/FOCS.2013.65.

16 Meng He, Ganggui Tang, and Norbert Zeh. Orienting dynamic graphs, with applications to maximal matchings and adjacency queries. In *International Symposium on Algorithms and Computation*, pages 128–140. Springer, 2014. doi:10.1007/978-3-319-13075-0_11.

17 Niklas Hjuler, Giuseppe F. Italiano, Nikos Parotsidis, and David Saulpic. Dominating sets and connected dominating sets in dynamic graphs. In Rolf Niedermeier and Christophe Paul, editors, *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, Berlin, Germany, March 13-16, 2019*, LIPIcs, pages 35:1–35:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.STACS.2019.35.

18 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.

19 Tsvi Kopelowitz, Robert Krauthgamer, Ely Porat, and Shay Solomon. Orienting fully dynamic graphs with worst-case time bounds. In *International Colloquium on Automata, Languages, and Programming*, pages 532–543. Springer, 2014. doi:10.1007/978-3-662-43951-7_45.

20 Christoph Lenzen and Roger Wattenhofer. Minimum dominating set approximation in graphs of bounded arboricity. In Nancy A. Lynch and Alexander A. Shvartsman, editors, *Distributed Computing, 24th International Symposium, DISC 2010, Cambridge, MA, USA, September 13-15, 2010. Proceedings*, volume 6343 of *Lecture Notes in Computer Science*, pages 510–524. Springer, 2010. doi:10.1007/978-3-642-15763-9_48.

21 Adir Morgan, Shay Solomon, and Nicole Wein. Algorithms for the minimum dominating set problem in bounded arboricity graphs: Simpler, faster, and combinatorial. In Seth Gilbert, editor, *35th International Symposium on Distributed Computing, DISC 2021, Freiburg, Germany (Virtual Conference), October 4-8, 2021*, LIPIcs, pages 33:1–33:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.DISC.2021.33.

22 C St JA Nash-Williams. Decomposition of finite graphs into forests. *Journal of the London Mathematical Society*, 1(1):12–12, 1964.

23 Ofer Neiman and Shay Solomon. Simple deterministic algorithms for fully dynamic maximal matching. *ACM Transactions on Algorithms (TALG)*, 12(1):1–15, 2015. doi:10.1145/2700206.

24 Krzysztof Onak, Baruch Schieber, Shay Solomon, and Nicole Wein. Fully dynamic mis in uniformly sparse graphs. *ACM Transactions on Algorithms (TALG)*, 16(2):1–19, 2020. doi:10.1145/3378025.

25 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*, pages 712–729. SIAM, 2016.

26 Shay Solomon. Local algorithms for bounded degree sparsifiers in sparse graphs. *arXiv preprint arXiv:2105.02084*, 2021. arXiv:2105.02084.

27 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. doi:10.1145/3564246.3585211.

28 Shay Solomon, Amitai Uzrad, and Tianyi Zhang. A lossless deamortization for dynamic greedy set cover. In *2024 IEEE 65th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 264–290. IEEE, 2024. doi:10.1109/FOCS61266.2024.00025.

29 Hao Sun. An improved approximation bound for minimum weight dominating set on graphs of bounded arboricity. In *International Workshop on Approximation and Online Algorithms*, pages 39–47. Springer, 2021. doi:10.1007/978-3-030-92702-8_3.

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