Deterministic Rounding of Dynamic Fractional Matchings

Sayan Bhattacharya
Department of Computer Science, University of Warwick, Coventry, UK

Peter Kiss
Department of Computer Science, University of Warwick, Coventry, UK

Abstract

We present a framework for *deterministically* rounding a dynamic fractional matching. Applying our framework in a black-box manner on top of existing fractional matching algorithms, we derive the following new results: (1) The first deterministic algorithm for maintaining a $(2-\delta)$-approximate maximum matching in a fully dynamic bipartite graph, in arbitrarily small polynomial update time. (2) The first deterministic algorithm for maintaining a $(1+\delta)$-approximate maximum matching in a decremental bipartite graph, in polylogarithmic update time. (3) The first deterministic algorithm for maintaining a $(2+\delta)$-approximate maximum matching in a fully dynamic general graph, in *small* polylogarithmic (specifically, $O(\log^4 n)$) update time. These results are respectively obtained by applying our framework on top of the fractional matching algorithms of Bhattacharya et al. [STOC'16], Bernstein et al. [FOCS'20], and Bhattacharya and Kulkarni [SODA'19].

Previously, there were two known general-purpose rounding schemes for dynamic fractional matchings. Both these schemes, by Arar et al. [ICALP'18] and Wajc [STOC'20], were randomized.

Our rounding scheme works by maintaining a good *matching-sparsifier* with bounded arboricity, and then applying the algorithm of Peleg and Solomon [SODA'16] to maintain a near-optimal matching in this low arboricity graph. To the best of our knowledge, this is the first dynamic matching algorithm that works on general graphs by using an algorithm for low-arboricity graphs as a black-box subroutine. This feature of our rounding scheme might be of independent interest.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Matching, Dynamic Algorithms, Data Structures

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.27

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2105.01615`

Funding *Sayan Bhattacharya*: Supported by EPSRC Grant EP/S03353X/1.

1 Introduction

The central question in the area of *dynamic algorithms* is to understand how can we efficiently maintain a good solution to a computational problem, when the underlying input changes over time [26, 28]. In the past decade, an extensive body of work in this area has been devoted to the study of *dynamic matching* [1, 5, 6, 7, 8, 10, 11, 17, 22, 24, 34, 35, 38, 39].

A matching $M \subseteq E$ in G is a set of edges that do not share any common endpoint. In the dynamic matching problem, the input is a graph $G = (V, E)$ that keeps getting updated via edge insertions/deletions, and the goal is to maintain an approximately *maximum matching* in G with small (preferably polylogarithmic) update time, where the phrase "update time" refers to the time it takes to handle an "update" (edge insertion/deletion) in G.[1] From the current landscape of dynamic matching, we can identify a common template that underpins a number of existing algorithms for this problem. This template consists of three steps.

[1] An algorithm has an "amortized" update time of $O(\tau)$ iff starting from an empty graph, it can handle any sequence of κ edges insertions/deletions in $O(\tau \cdot \kappa)$ total time.

Step (I). Design an efficient dynamic algorithm that maintains an approximately maximum *fractional matching*[2] $w : E \to [0, 1]$ in the input graph $G = (V, E)$. All the known algorithms for this first step are deterministic [9, 12, 13, 15, 14, 16, 23].

Step (II). Maintain a *sparse* (bounded-degree) subgraph $S = (V, E_S)$ of the input graph, with $E_S \subseteq E$, that approximately preserves the size of maximum matching [3, 40]. In a bit more details, the subgraph S should have the property that $\mu(S)$ is very close to $size(w)$, where $\mu(S)$ denotes the size of maximum (integral) matching in S, and $size(w) = \sum_{e \in E} w(e)$ denotes the size of the fractional matching w from the previous step. Such a subgraph S is often referred to as a *matching-sparsifier* of G [4]. There are two known algorithms for this second step and both of them are randomized, in sharp contrast to Step (I). Specifically, Arar et al. [3] designed a randomized rounding scheme for sparsifying a dynamic fractional matching. Their algorithm works only in the *oblivious adversary* setting, where the future updates cannot depend on the past actions taken by the algorithm. This result was very recently improved upon by Wajc [40], who presented an elegant dynamic rounding scheme for Step (II) that, although randomized, works in a much more general *adaptive adversary* setting, where the future updates to the algorithm can depend on all its past random bits.

Step (III). Maintain a near-optimal matching in the (bounded-degree) sparsifier S from the previous step, using a known algorithm by Gupta et al. [25], which has $O(\Delta)$ update time on dynamic graphs with maximum degree $\leq \Delta$. Since S has bounded degree, the third step incurs only a small overhead in the update time. The algorithm in [25] is also deterministic.

A natural question arises from the preceding discussion. Can we design an efficient *deterministic* dynamic algorithm for Step (II)? Since Step (I) and Step (III) are already deterministic, an efficient deterministic algorithm for Step (II) will allow us to derandomize multiple existing results in the literature on dynamic matching. We resolve this question in the affirmative. Specifically, our main result is summarized in the theorem below.

▶ **Theorem 1.** *Fix any small constant $\delta > 0$. Consider a dynamic graph $G = (V, E)$ on n nodes and a (dynamic) fractional matching w in G. In this setting, an update either inserts/deletes an edge in $G = (V, E)$ or changes the weight $w(e)$ of an existing edge $e \in E$. We can deterministically maintain a subgraph $S = (V, E_S)$ of G, with $E_S \subseteq E$, such that:*

1. *There exists a fractional matching $h' : E_S \to [0, 1]$ in S with $size(w) \leq (1 + \delta) \cdot size(h')$.*
2. *If w is a (δ, δ)-approximate maximal matching in G, then $\mu(G) \leq (2 + \delta) \cdot \mu(S)$.*
3. *The arboricity of S is $O(\log^2 n)$.*
4. *Every update in G or w, on average, leads to $O(\log^2 n)$ updates in S.*
5. *Our dynamic algorithm for maintaining S has $O(\log^2 n)$ amortized update time.*

Bounded arboricity matching-sparsifiers. We will shortly explain part-(2) of Theorem 1, which uses the notion of a (δ, δ)-*approximate maximal matching* that has not been defined yet. For now, we focus on an intriguing feature of Theorem 1, namely, that it only maintains a subgraph S with bounded *arboricity*.[3] This is in sharp contrast to all previous work on dynamic matching-sparsifiers: they satisfy the strictly stronger requirement of bounded

[2] A fractional matching w in G assigns a weight $w(e) \in [0, 1]$ to every edge $e \in E$, ensuring that the total weight assigned to all the edges incident on any given node is ≤ 1.

[3] Informally, an undirected graph $G' = (V', E')$ has arboricity κ if we can assign a direction to each of its edges $e \in E'$ in such a way that every node $v \in V'$ gets an out-degree of at most $O(\kappa)$. If a graph has maximum degree at most κ, then its arboricity is also $O(\kappa)$, but not vice versa.

maximum degree [3, 40]. Our algorithm exploits this feature in a crucial manner, allowing certain nodes to have large degrees in S while ensuring that the arboricity of S remains at most $O(\log^2 n)$. This does not cause any problem in the overall scheme of things, however, because Peleg and Solomon [37] have shown how to deterministically maintain a $(1+\delta)$-approximate maximum matching in $O(\Delta)$ update time in a dynamic graph with arboricity $\leq \Delta$. Their algorithm allows us to efficiently maintain a near-optimal matching in S.

To summarize, there is an existing line of work on dynamic matching which deal with the special class of low-arboricity graphs [29, 34, 37]. Theorem 1 shows that if we have a good dynamic matching algorithm for low-arboricity graphs, then we can use it in a black-box manner to design better dynamic matching algorithms for *general graphs* as well.

Implications of Theorem 1. We start by focussing on bipartite graphs. If a graph G is bipartite, then the size of a maximum fractional matching in G is equal to $\mu(G)$. Accordingly, part-(1) of Theorem 1 implies that if the input graph G is bipartite, then our dynamic algorithm maintains a sparsifier $S = (V, E_S)$ such that $size(w) \leq (1+\delta) \cdot \mu(S)$. We can now run the dynamic algorithm from [37] on S, which has small arboricity, to efficiently maintain a near-optimal (integral) matching $M \subseteq E_S$ with $size(w) \leq (1+\delta) \cdot |M|$.

Bhattacharya et al. [14] gave a deterministic algorithm for maintaining $(2-\epsilon)$-approximate maximum *fractional* matchings in bipartite graphs with arbitrarily small polynomial update time. Applying our dynamic rounding framework on top of this result from [14], we get the *first deterministic* algorithm for dynamic (integral) matchings in bipartite graphs with the same approximation ratio and similar update time, as summarized in the theorem below.

▶ **Theorem 2.** *For every constant $k \geq 10$, there exists a $\beta_k \in (1,2)$, and a* deterministic *dynamic algorithm that maintains a β_k-approximate maximum matching in an n-node bipartite graph with $O(n^{1/k} \cdot \log^4 n)$ amortized update time.*

Next, very recently Bernstein et al. [9] showed how to maintain a $(1+\delta)$-approximate maximum *fractional* matching in a bipartite graph with $O(\log^3 n)$ amortized update time in the *decremental* setting, where the input graph only undergoes edge-deletions. Applying our dynamic rounding framework on top of their result, we get the *first deterministic* algorithm for maximum (integral) matching in an analogous decremental setting, with the same approximation ratio and similar update time. This is stated in the theorem below.

▶ **Theorem 3.** *We can* deterministically *maintain a $(1+\delta)$-approximate maximum matching in a decremental bipartite graph on n nodes with $O(\log^7 n)$ amortized update time.*

Moving on to general graphs, we note that if a graph $G = (V, E)$ is non-bipartite, then the size of a maximum fractional matching can be as large as $(3/2) \cdot \mu(G)$. Thus, if we are to naively apply our dynamic rounding framework based on the guarantee given to us by part-(1) of Theorem 1, then we will lose out on a factor of $3/2$ in the approximation ratio. This is where part-(2) of Theorem 1 comes in handy. Specifically, as in [3, 40], we invoke the notion of an (α, β)*-approximate maximal matching* (see Definition 6).

To see why this notion is useful for us, consider the result of Bhattacharya and Kulkarni [16], who designed a deterministic dynamic algorithm for $(2+\delta)$-approximate maximum *fractional* matching, for small constant $\delta > 0$, in general graphs with $O(1)$ amortized update time. Furthermore, the fractional matching maintained by [16] is (δ, δ)-approximately maximal. Thus, applying Theorem 1 on top of this result from [16], we can deterministically maintain a sparsifier $S = (V, E_S)$ of the input graph G with $\mu(G) \leq (2+\delta) \cdot \mu(S)$. We can now maintain a near-optimal maximum matching $M \subseteq E_S$ in S, using the algorithm of [37]. Since $\mu(G) \leq (2+\delta) \cdot \mu(S)$, M will be a $(2+\delta)$-approximate maximum (integral) matching in G. Putting everything together, we get the result summarized in the theorem below.

▶ **Theorem 4.** *We can* deterministically *maintain a $(2+\delta)$-approximate maximum matching in an n-node dynamic graph with $O(\log^4 n)$ amortized update time, for small constant $\delta > 0$.*

Prior to our work, the only *deterministic* dynamic algorithm for $(2+\delta)$-approximate maximum matching in general graphs with polylogarithmic update time was due to Bhattacharya et al. [14]. The exact polylogarithmic factor in the update time of [14] was huge (more than $\log^{20} n$), and the algorithm of [14] was significantly more complicated than ours.

Perspective. Existing techniques for proving update-time lower bounds for dynamic problems cannot distinguish between deterministic and randomized algorithms [2, 27, 30, 31, 36]. Thus, understanding the power of randomization in the dynamic setting is an important research agenda, which comprises of two separate strands of work. (1) Studying the power of the *oblivious adversary* assumption while designing a randomized algorithm for a given dynamic problem. (2) Studying the separation between randomized algorithms that work against *adaptive adversaries* on the one hand, and deterministic algorithms on the other. Our work falls under the second category. A recent breakthrough result under this category has been a deterministic algorithm for dynamic minimum spanning forest with worst-case subpolynomial update time [18, 21]. This improves upon earlier work which achieved the same update time guarantee for dynamic minimum spanning forest, but using a randomized algorithm that works against adaptive adversary [32]. There are other well-studied dynamic problems where currently we have polynomial gaps between the update times of the best-known deterministic algorithm and the best-known randomized algorithm against adaptive adversary [19, 20]. Bridging these gaps remain challenging open questions.

Our Techniques. The key ingredient in our rounding scheme is a simple *degree-split* procedure. Given any graph $G' = (V', E')$ as input, this procedure runs in linear time and outputs a subgraph $G'' = (V', E'')$ where the degree of every node $v \in V'$ drops by a factor of $(1/2) \cdot (1 \pm \epsilon)$, provided the initial degree of v in G' was larger than $(1/\epsilon)$. See Algorithm 3.

Using this degree-split procedure, we first design a simple *static* algorithm for sparsifying a *uniform* fractional matching w (which assigns the same weight to every edge) in an input graph $G = (V, E)$. This works in rounds. In each round, we start by repeatedly removing the nodes with degree at most $(1/\epsilon)$, until we are left with a graph $G' = (V', E')$ where every remaining node has degree larger than $(1/\epsilon)$. We now apply the degree-split procedure on $G' = (V', E')$ to obtain a subset of edges $E'' \subseteq E'$, double the weight of every edge $e \in E''$, and discard the edges $e \in E'' \setminus E'$ from the support of the fractional matching. Since the degree-split procedure reduces the degree of every node in V' by (approximately) a factor of $1/2$, it follows that we (approximately) preserve the total weight received by every node while implementing a given round. We can show that if we continue with this process for (roughly) logarithmic many rounds, then we end up with a bounded-arboricity subgraph of the input graph G that approximately preserves the size of the fractional matching w. In the dynamic setting, we try to mimic this static algorithm in a natural lazy manner.

When the input is a dynamic graph G and a (not necessarily uniform) fractional matching w, then, roughly speaking, we first discretize w and then decompose it into $O(\log n)$ many uniform fractional matchings, defined on mutually edge-disjoint subgraphs of G. We run a dynamic algorithm for sparsifying a uniform fractional matching on each of these subgraphs, and we maintain the union of the outputs of all these $O(\log n)$ many dynamic sparsifiers.

2 Notations and Preliminaries

Throughout this paper, we let $G = (V, E)$ denote the input graph, and $n = |V|$ will be the number of nodes in G. For any $v \in V$, $E' \subseteq E$ and $V' \subseteq V$, we let $E'(v, V') = \{(u, v) \in E' : u \in V'\}$ denote the set of edges in E' that are incident on v and have their other endpoints in V'. To ease notations, we define $E'(v) := E'(v, V)$. We also define $\deg_{E'}(v, V') := |E'(v, V')|$ and $\deg_{E'}(v) := |E'(v)|$. Furthermore, given any subset of edges $E' \subseteq E$, we let $V(E') = \bigcup_{(u,v)\in E'} \{u, v\}$ denote the set of endpoints of the edges in E'. Throughout the rest of this paper, we will consider δ to be some small constant, and we will fix two more parameters β and ϵ as stated below.

$$10^{-3} \geq \delta = 20 \cdot \beta = 5000 \cdot \epsilon \cdot \log n > 0. \tag{1}$$

Given any subset of edges $E' \subseteq E$, a *weight-function* $w' : E' \to [0, 1]$ assigns a (possibly fractional) weight $0 \leq w'(e) \leq 1$ to every edge $e \in E'$. We say that E' is the *support* of w' and write $\textsc{Support}(w') := E'$. The *size* of the weight-function w' is defined as $size(w') := \sum_{e\in E'} w'(e)$. For any node $v \in V$, let $w'(v) := \sum_{(u,v)\in E'} w'(u, v)$ denote the total weight received by v from all its incident edges under the weight-function w'. We say that w' is a *fractional matching* in the graph $G' := (V, E')$ iff $w'(v) \leq 1$ for all $v \in V$. Since $E' \subseteq E$, we often abuse notation to say that such a weight-function w' is a fractional matching in $G = (V, E)$ as well. For any $0 \leq \lambda \leq 1$, we say that w' is a λ-*uniform* weight-function (or, fractional matching, if $w'(v) \leq 1$ for all $v \in V$) iff $w'(e) = \lambda$ for all edges $e \in E'$.

Let $\mu(G')$ and $\mu_f(G')$ respectively denote the size of maximum matching and the size of maximum fractional matching in a graph G'. We will use the following well-known theorem.

▶ **Theorem 5.** *Consider any graph G'. If G' is bipartite, then $\mu(G') = \mu_f(G')$. Otherwise, we have $\mu(G) \leq \mu_f(G') \leq (3/2) \cdot \mu(G)$.*

We will use the notion of an *approximately maximal* matching as in Arar et al. [3].

▶ **Definition 6.** *Consider any graph $G' = (V', E')$ and a fractional matching w' in G'. We say that w' is a (α, β)-approximately maximal matching in G' iff the following holds. For every edge $(u, v) \in E'$, either (1) $\{w'(u, v) \geq \beta\}$, or (2) {there is at least one endpoint $x \in \{u, v\}$ such that $w'(x) \geq 1 - \alpha$ and $w'(x, y) < \beta$ for all edges $(x, y) \in E'$ incident on x}.*

An *orientation* of a graph $G' = (V', E')$ assigns a direction to every edge $(u, v) \in E'$. For the rest of this paper, whenever we say that a graph G' has *arboricity* $O(\kappa)$, we mean that G' admits an orientation of its edges where the maximum out-degree of a node is $O(\kappa)$ [33].

3 A Static Algorithm for Sparsifying a Uniform Fractional Matching

In this section, we present a simple static algorithm for sparsifying a uniform fractional matching. This will form the basis of our dynamic algorithm in Section 4 and Section 5.

As input, we receive a graph $G = (V, E)$ and a λ-uniform fractional matching $w : E \to [0, 1]$ in G, for some $\lambda \in [\delta/n^2, \beta)$. Define $L = L(\lambda)$ to be the largest integer k such that $\beta/2 \leq 2^k\lambda < \beta$. Since δ is a constant and $\delta/n^2 \leq \lambda < \beta$, from (1) we infer that $L = O(\log n)$.

The algorithm proceeds in *rounds* $i \in \{0, \ldots, L - 1\}$. Before the start of round $i = 0$, we initialize $E^{(\geq 0)} := E$, $V^{(\geq 0)} := V$, $G^{(\geq 0)} := \left(V^{(\geq 0)}, E^{(\geq 0)}\right)$ and $\gamma^{(0)} := w$. Thus, $\gamma^{(0)}$ is a λ-uniform fractional matching in G. In each round i, we identify a subset of edges $F^{(i)} \subseteq E^{(\geq i)}$ that get *frozen*, in the sense that they are *not* considered in subsequent rounds. Define $\mathcal{F}^{(i)} := E^{(\geq i)} \bigcup_{j=0}^{i-1} F^{(j)}$ and $\mathcal{H}^{(i)} := (V, \mathcal{F}^{(i)})$ for all $i \in [0, L]$. The following invariant will be satisfied in the beginning of each round $i \in [0, L]$. The weight-function $\gamma^{(i)} : \mathcal{F}^{(i)} \to [0, 1]$ ensures that $\gamma^{(i)}(v) \simeq w(v)$ for all $v \in V$. Clearly, this invariant holds for $i = 0$.

Implementing a given round $i \in [0, L-1]$. Initialize $G' = (V', E') := G^{(\geq i)}$. There are two distinct *steps* in this round. During the first step, we keep iteratively removing the nodes with degree $\leq (1/\epsilon)$ from G'. Let $V^{(i)} \subseteq V^{(\geq i)}$ be the collection of nodes that get removed from G' in this manner, and let $F^{(i)} \subseteq E^{(\geq i)}$ denote the set of edges incident on $V^{(i)}$. Define $V^{(\geq i+1)} := V^{(\geq i)} \setminus V^{(i)}$. At the end of this first step, the status of $G' = (V', E')$ is as follows: $V' = V^{(\geq i+1)}$ and $E' = E^{(\geq i)} \setminus F^{(i)}$. Intuitively, we can afford to remove the nodes $V^{(\geq i)}$ from G' because the edges in $F^{(i)}$ admit an orientation with maximum out-degree $(1/\epsilon)$. At the end of this first step every node in G' has degree $\geq (1/\epsilon)$.

In the second step, we call a subroutine DEGREE-SPLIT(E'), which returns a subset of edges $E'' \subseteq E'$ with the following property: $\deg_{E''}(v) \simeq (1/2) \cdot \deg_{E'}(v)$ for all nodes $v \in V'$. We will shortly see how to implement this DEGREE-SPLIT subroutine. For now, we move ahead with the description of round i. We set $E^{(\geq i+1)} := E''$ and $G^{(\geq i+1)} := \left(V^{(\geq i+1)}, E^{(\geq i+1)}\right)$. Next, we discard the edges in $E' \setminus E^{(\geq i+1)}$ from the support of γ and double the weights on the remaining edges in $E^{(\geq i+1)}$. This leads us to a new weight-function $\gamma^{(i+1)} : \mathcal{F}^{(i+1)} \to [0,1]$ in $\mathcal{H}^{(i+1)} = (V, \mathcal{F}^{(i+1)})$ which is defined as follows. For every edge $e \in \mathcal{F}^{(\geq i+1)}$, we have:

$$\gamma^{(i+1)}(e) = \begin{cases} 2 \cdot \gamma^{(i)}(e) & \text{if } e \in E^{(\geq i+1)}; \\ \gamma^{(i)}(e) & \text{else if } e \in \bigcup_{j=0}^{i} F^{(j)}. \end{cases} \tag{2}$$

At this point, if $i < L-1$, then we are ready to proceed to the next round $i+1$. Otherwise, if $i = L-1$, then we terminate the algorithm after setting $F^{(L)} := E^{(\geq L)}$ and $V^{(L)} := V^{(\geq L)}$.

Define $F := \bigcup_{i=0}^{L} F^{(i)}$, $H := (V, F)$, and $h := \gamma^{(L)}$. We will show that $H = (V, F)$ is a good matching-sparsifier for the λ-uniform matching w in the input graph $G = (V, E)$. The relevant pseudocodes are summarized in Algorithm 1, Algorithm 2 and Algorithm 3. For clarity of exposition, in some of these pseudocodes we use one additional notation $h^{(i)}$, which is basically a weight-function $h^{(i)} : F^{(i)} \to [0,1]$ such that $h^{(i)}(e) = 2^i \lambda$ for all $e \in F^{(i)}$.

Implementing the DEGREE-SPLIT(E')subroutine. Consider any graph $G^* = (V^*, E^*)$. A *walk* W in G^* is a set of distinct edges $\{(u_0, v_0), \ldots, (u_k, v_k)\} \subseteq E^*$ such that $v_i = u_{i+1}$ for all $i \in [0, k-1]$. Let $W^{(even)} = \{(u_{2i}, v_{2i}) : i \in [0, \lfloor k/2 \rfloor]\}$ denote the collection of even numbered edges from this walk W. The walk W is said to be *maximal* in G^* iff $\deg_{E^* \setminus W}(u_0) = \deg_{E^* \setminus W}(u_k) = 0$. When we call DEGREE-SPLIT(E'), it first partitions the edge-set E' into a collection of walks $\mathcal{W}$ as specified in Algorithm 3. It then returns the set $E'' \subseteq E'$, which consists of all the even numbered edges from all the walks $W \in \mathcal{W}$.

▷ Claim 7. The subroutine DEGREE-SPLIT(E') runs in $O(|E'|)$ time.

Proof. We can compute a maximal walk W in a given graph $G^* = (V^*, E^*)$ in $O(|W|)$ time. Hence, the total running time of Algorithm 3 is given by $\sum_{W \in \mathcal{W}} O(|W|) = O(|E'|)$. ◁

▷ Claim 8. In Algorithm 3, $\deg_{E''}(v) \in \left[\frac{\deg_{E'}(v)}{2} - 1, \frac{\deg_{E'}(v)}{2} + 1\right]$ for all nodes $v \in V(E')$.

Proof. Consider any graph $G^* = (V^*, E^*)$ and a walk $W = \{(u_0, v_0), \ldots, (u_k, v_k)\}$ in G^*. Let END$(W) = \{u_0, v_k\}$ denote the endpoints of this walk W, and let $V(W) = \bigcup_{i=0}^{k}\{u_i\} \bigcup_{i=0}^{k}\{v_i\}$ denote the set of all nodes touched by W. Note that $\deg_{W^{(even)}}(v) = (1/2) \cdot \deg_W(v)$ for all nodes $v \in V(W) \setminus$ END(W), and $\deg_{W^{(even)}}(v) \in [(1/2) \cdot \deg_W(v) - 1, (1/2) \cdot \deg_W(v) + 1]$ for all nodes $v \in$ END(W). Now, fix any node $v \in V(E')$, and observe that:

1. $\deg_{E''}(v) = \sum_{W \in \mathcal{W}} \deg_{W^{(even)}}(v)$ and $\deg_{E'}(v) = \sum_{W \in \mathcal{W}} \deg_W(v)$.
2. The definition of a maximal walk implies that $v \in$ END(W) for at most one $W \in \mathcal{W}$.

These observations, taken together, imply the claim. ◁

Algorithm 1 STATIC-UNIFORM-SPARSIFY$(G = (V, E), \lambda)$, where $\delta/n^2 \leq \lambda < \beta$.

```
Let $w : E \to [0, 1]$ be a $\lambda$-uniform fractional matching in $G$.
Initialize $V^{(\geq 0)} := V$ and $E^{(\geq 0)} := E$.
Initialize a weight-function $h^{(0)} : E^{(\geq 0)} \to [0, 1]$ so that $h^{(0)}(e) := \lambda$ for all $e \in E^{(\geq 0)}$.
Let $L := L(\lambda)$ be the unique nonnegative integer $k$ such that $\beta/2 \leq 2^k\lambda < \beta$.
Call the subroutine REBUILD$(0, \lambda)$. (see Algorithm 2)
Define $F := \bigcup_{i=0}^{L} F^{(i)}$, and $H := (V, F)$.
Define $h : F \to [0, 1]$ such that for all $i \in [0, L]$ and $e \in F^{(i)}$ we have $h(e) := h^{(i)}(e)$.
```

Algorithm 2 REBUILD(i', λ).

```
for $i = i'$ to $(L - 1)$ do:
    $V^{(i)} \leftarrow \emptyset$.
    while there is some node $v \in V^{(\geq i)} \setminus V^{(i)}$ with $\deg_{E^{(\geq i)}}\left(v, V^{(\geq i)} \setminus V^{(i)}\right) \leq (1/\epsilon)$ do:
        $V^{(i)} \leftarrow V^{(i)} \cup \{v\}$.
    $V^{(\geq i+1)} \leftarrow V^{(\geq i)} \setminus V^{(i)}$.
    $F^{(i)} \leftarrow \{(u, v) \in E^{(\geq i)} :$ either $u \in V^{(i)}$ or $v \in V^{(i)}\}$.
    $E^{(\geq i+1)} \leftarrow$ DEGREE-SPLIT$(E^{(\geq i)} \setminus F^{(i)})$.
    for all edges $e \in E^{(\geq i+1)}$ do:
        $h^{(i+1)}(e) \leftarrow 2 \cdot h^{(i)}(e)$.
$F^{(L)} \leftarrow E^{(\geq L)}$.
$V^{(L)} \leftarrow V^{(\geq L)}$.
```

Algorithm 3 DEGREE-SPLIT(E').

```
Initialize $E^* \leftarrow E'$ and $\mathcal{W} \leftarrow \emptyset$.
while $E^* \neq \emptyset$ do
    Let $G^* := (V(E^*), E^*)$, where $V(E^*)$ is the set of endpoints of the edges in $E^*$.
    Compute a *maximal* walk $W$ in $G^*$.
    Set $\mathcal{W} \leftarrow \mathcal{W} \cup \{W\}$, and $E^* \leftarrow E^* \setminus W$.
Return the set of edges $E'' := \bigcup_{W \in \mathcal{W}} W^{(even)}$.
```

Note that $h^{(i)}$ and $F^{(i)}$ represent global variables in the pseudocodes above.

▶ **Lemma 9.** *Algorithm 1 runs in $O(|E|)$ time.*

Proof. The runtime of Algorithm 1 is dominated by the call to REBUILD$(0, \lambda)$. Accordingly, focus on any given iteration $i \in [0, L - 1]$ of the outer FOR loop in Algorithm 2. During this iteration, using appropriate data structures the inner WHILE loop takes $O(|F^{(i)}|)$ time, the call to DEGREE-SPLIT$(E^{(\geq i)} \setminus F^{(i)})$ takes $O(|E^{(\geq i)} \setminus F^{(i)}|)$ time as per Claim 7, and the inner FOR loop takes $O(|E^{(\geq i+1)}|)$ time. Hence, the total time taken to implement iteration i of the outer FOR loop is at most $O(|E^{(\geq i)}|) + O(|E^{(\geq i)} \setminus F^{(i)}|) + O(|E^{(\geq i+1)}|) = O(|E^{(\geq i)}|)$.

Summing over all $i \in [0, L - 1]$, we derive that:

$$\text{The total runtime of Algorithm 1 is at most } \sum_{i=0}^{L-1} O\left(|E^{(\geq i)}|\right). \tag{3}$$

Next, fix an iteration i of the outer For loop in Algorithm 2, and focus on the call to Degree-Split($E^{(\geq i)} \setminus F^{(i)}$). The inner While loop ensures that $\deg_{E^{(\geq i)} \setminus F^{(i)}}(v) > (1/\epsilon)$ for all $v \in V^{(\geq i+1)}$. By Claim 8, the degree of every concerned node is (roughly) halved by a call to Degree-Split(.). Hence, $|E^{(\geq i+1)}| = O\left((1/2) \cdot |E^{(\geq i)}|\right)$ for all $i \in [0, L-1]$. Plugging this back into (3), the lemma follows since $\sum_{i=0}^{L-1} O\left(|E^{(\geq i)}|\right) = O\left(|E^{(\geq 0)}|\right) = O(|E|)$. ◀

We will next show that the subgraph $H = (V, F)$ returned by our algorithm is a good matching-sparsifier.Towards this end, we first derive the following important claim.

▷ Claim 10 (Informal). For all $v \in V$ and $i \in [0, L-1]$, we have $\gamma^{(i+1)}(v) \simeq (1+\epsilon) \cdot \gamma^{(i)}(v)$.

Proof. Let us track how starting from $\gamma^{(i)}$, the weight-function $\gamma^{(i+1)}$ is constructed during round i. Recall that Support $\left(\gamma^{(i)}\right) := E^{(\geq i)} \bigcup_{j=0}^{i-1} F^{(j)}$. During round i, we first identify the subset $F^{(i)} \subseteq E^{(\geq i)}$, and then identify another subset $E^{(\geq i+1)} \subseteq E^{(\geq i)} \setminus F^{(i)}$. As we switch from $\gamma^{(i)}$ to $\gamma^{(i+1)}$, the following three events occur: (1) The weights of the edges $e \in F^{(i)} \bigcup_{j=0}^{i-1} F^{(j)}$ do not change. (2) The edges $e \in \left(E^{(\geq i)} \setminus F^{(i)}\right) \setminus E^{(\geq i+1)}$ get discarded from the support of $\gamma^{(i+1)}$. (3) The weights of the remaining edges $e \in E^{(\geq i+1)}$ get doubled.

Now, fix any node $v \in V$. The claim follows from our analysis of the two cases below.

Case 1: $v \in V^{(\geq i+1)}$. In this case, the inner While loop in Algorithm 2 ensures that $\deg_{E^{(\geq i)} \setminus F^{(i)}}(v) > (1/\epsilon)$. Hence, applying Claim 8, we get: $\deg_{E^{(\geq i+1)}}(v) \simeq (1 \pm \epsilon) \cdot (1/2) \cdot \deg_{E^{(\geq i)} \setminus F^{(i)}}(v)$. To summarize, (about) half the edges in $E^{(\geq i)} \setminus F^{(i)}$ that are incident on v get discarded from the support of $\gamma^{(i+1)}$, while the remaining edges in $E^{(\geq i)} \setminus F^{(i)}$ double their weights. In contrast, the edges in $F^{(i)} \bigcup_{j=0}^{i-1} F^{(j)}$ that are incident on v do not change their weights at all. This implies that $\gamma^{(i+1)}(v) \simeq (1 \pm \epsilon) \cdot \gamma^{(i)}(v)$.

Case 2: $v \notin V^{(\geq i+1)}$. In this case, every edge $(u, v) \in$ Support $\left(\gamma^{(i)}\right)$ continues to remain in the support of $\gamma^{(i+1)}$ with the same weight. Hence, we get: $\gamma^{(i+1)}(v) = \gamma^{(i)}(v)$. ◁

▶ **Lemma 11** (Informal). *The weight-function $h : F \to [0, 1]$ satisfies three properties:*

1. *$h(e) < \beta$ for all edges $e \in F$.*
2. *$w(v) \simeq (1 \pm \epsilon L) \cdot h(v)$ for all nodes $v \in V$.*
3. *$size(w) \simeq (1 \pm \epsilon L) \cdot size(h)$.*

Proof. Before the start of round 0, we have $\gamma^{(0)}(e) = \lambda$ for all edges $e \in E^{(\geq 0)}$. Subsequently, in each round $i \in [0, L-1]$, the weight of each edge $e \in E^{(\geq i+1)}$ gets doubled. Hence, we have $h(e) = \gamma^{(L)}(e) \leq 2^L \lambda < \beta$ for all $e \in F$. This proves part-(1) of the lemma.

Next, fix any node $v \in V$. Before the start of round 0, we have $\gamma^{(0)} = w$ and hence $\gamma^{(0)}(v) = w(v)$. Subsequently, after each round $i \in [0, L-1]$, Claim 10 guarantees that $\gamma^{(i+1)}(v) \simeq (1 \pm \epsilon) \cdot \gamma^{(i)}(v)$. This gives us: $h(v) = \gamma^{(L)}(v) \simeq (1 \pm \epsilon)^L \cdot \gamma^{(0)}(v) \simeq (1 \pm \epsilon L) \cdot \gamma^{(0)}(v) \simeq (1 \pm \epsilon L) \cdot w(v)$. Finally, summing this (approximate) equality over all nodes $v \in V$, we get: $size(w) \simeq (1 \pm \epsilon L) \cdot size(h)$. This proves part-(2) and part-(3) of the lemma. ◀

Levels of nodes and edges. The *level* of a node $v \in V$ is defined as $\ell(v) := \max\{i \in [0, L] : v \in V^{(\geq i)}\}$. Similarly, the *level* of an edge $e \in E$ is defined as $\ell(e) := \max\{i \in [0, L] : e \in E^{(\geq i)}\}$. Since $V^{(i)} = V^{(\geq i)} \setminus V^{(\geq i+1)}$ for all $i \in [0, L]$, it follows that $\ell(v) = i$ iff $v \in V^{(i)}$.

▶ **Observation 12.** *For all $(u, v) \in F$, we have $\ell(u, v) = \min(\ell(u), \ell(v))$ and $(u, v) \in F^{(\ell(u,v))}$.*

Proof. Consider any edge $(u,v) \in F = \bigcup_{i=0}^{L} F^{(i)}$. W.l.o.g., suppose that $(u,v) \in F^{(j)}$ for some $j \in [0,L]$. Before the start of round 0, we have $(u,v) \in E^{(\geq 0)}$. During each round $i \in [0, j-1]$, the edge (u,v) gets included in the set $E^{(\geq i+1)}$, and both its endpoints u, v get included in the set $V^{(\geq i+1)}$. At round j, one of its endpoints (say, u) gets included in $V^{(j)}$, and the edge (u,v) also gets included in $F^{(j)}$. Since $F^{(j)} \subseteq E^{(\geq j+1)} \setminus E^{(\geq j)}$, we infer that $\ell(u) = j$, $\ell(v) \geq j$, and $\ell(u,v) = \max\left\{i \in [0,L] : (u,v) \in E^{(\geq i)}\right\} = j = \min(\ell(u), \ell(v))$. ◀

▶ **Observation 13.** *For every edge $(u,v) \in F$, we have $h(u,v) = 2^{\ell(u,v)} \cdot \lambda$.*

Proof. Suppose that $\ell(u,v) = i \in [0,L]$, and hence $(u,v) \in F^{(i)}$. Before the start of round 0, we have $(u,v) \in E^{(\geq 0)}$ and $\gamma^{(0)}(u,v) = \lambda$. During each round $j \in [0, i-1]$, the edge (u,v) gets included in $E^{(\geq j+1)}$ and we double its weight, i.e., we set $\gamma^{(j+1)}(u,v) := 2 \cdot \gamma^{(j)}(u,v)$. Thus, at the start of round i, we have $(u,v) \in E^{(\geq i)}$ and $\gamma^{(i)}(u,v) = 2^i \cdot \lambda$. During round i, the edge (u,v) gets included in the set $F^{(i)}$ and its weight is frozen for the subsequent rounds, so that we get: $2^i \cdot \lambda = \gamma^{(i)}(u,v) = \gamma^{(i+1)}(u,v) = \cdots = \gamma^{(L)}(u,v) = h(u,v)$. ◀

▶ **Lemma 14.** *The graph $H = (V,F)$ has arboricity at most $O\left(\epsilon^{-1} + \beta^{-1}\right) = O(\log n)$.*

Proof. For any nodes $u, v \in V$ with $\ell(u) = \ell(v) = i < L$, we say that u was assigned its level *before* v iff we had $u \in V^{(i)}$ just before the iteration of the inner WHILE loop in Algorithm 2 which adds v to $V^{(i)}$. We now define the following orientation of the graph $H = (V,F)$:

- Consider any edge $(u,v) \in F$. W.l.o.g. suppose that $\ell(u) \leq \ell(v)$. If $\ell(u) < \ell(v)$, then the edge is orientated *from u towards v*. Otherwise, if $\ell(u) = \ell(v) = L$, then the edge is oriented in any arbitrary direction. Finally, if $\ell(u) = \ell(v) < L$ and (say) the node u was assigned its level before the node v, then the edge is oriented *from u towards v*.

Fix any node $x \in V$. Define $\text{Out}_F(x) := \{(x,y) \in F :$ the edge (x,y) is oriented away from $x\}$. We will show that $|\text{Out}_F(x)| \leq O\left(\epsilon^{-1} + \beta^{-1}\right)$. The lemma will then follow from (1).

Case 1: $\ell(x) = i < L$. Let $X^- \subseteq V^{(\geq i)}$ be the set of nodes in $V^{(\geq i)}$ that are assigned the level i *before* the node x. In words, the symbol X^- denotes the status of the set $V^{(i)}$ just before x gets added to $V^{(i)}$ in Algorithm 2. For every edge $(x,y) \in \text{Out}_F(x)$, we have $y \in V^{\geq i} \setminus X^-$ and $(x,y) \in E^{(\geq i)}$. Hence, it follows that $|\text{Out}_F(x)| \leq \deg_{E^{(\geq i)}}(x, V^{(\geq i)} \setminus X^-) \leq \epsilon^{-1}$.

Case 2: $\ell(x) = L$. Consider any edge $(x,y) \in \text{Out}_F(x)$. Clearly, this implies that $\ell(y) = L$, and hence $\ell(x,y) = L$ by Observation 12. Thus, by Observation 13 we have $h(x,y) = 2^L \cdot \lambda \geq \beta/2$. In other words, $h(x,y) \geq \beta/2$ for all $(x,y) \in \text{Out}_F(x)$. Now, part-(2) of Lemma 11 implies that: $w(x) = \Omega(h(x)) = \Omega\left(\sum_{(x,y) \in \text{Out}_F(x)} h(x,y)\right) = \Omega\left(|\text{Out}_F(x)| \cdot (\beta/2)\right)$. Accordingly, we get: $w(x) = \Omega\left(|\text{Out}_F(x)| \cdot \beta\right)$, and hence: $|\text{Out}_F(x)| = O\left(\beta^{-1} \cdot w(x)\right) = O(\beta^{-1})$. The last inequality holds since $w(x) \leq 1$. ◀

To summarize, our static algorithm runs in linear time (Lemma 9), returns a subgraph $H = (V,F)$ with bounded arboricity (Lemma 14), and this subgraph H admits a fractional matching that closely approximates the input λ-uniform matching w in G (Lemma 11).

4 Dynamically Sparsifying a Uniform Fractional Matching

In this section, we will present a dynamic algorithm for sparsifying a uniform fractional matching, which will be referred to as DYNAMIC-UNIFORM-SPARSIFY$(G = (V,E), \lambda)$. The input to this algorithm is a dynamic graph $G = (V,E)$ that keeps changing via a sequence of updates (edge insertions/deletions), and a fixed parameter $\delta/n^2 \leq \lambda < \beta$. Throughout this sequence of updates, it is guaranteed that the graph G admits a valid λ-uniform fractional matching w. We will show how to maintain a subgraph $H_{(a)} = (V, F_{(a)})$ of this dynamic graph $G = (V, E)$, with $F_{(a)} \subseteq E$, that is a good matching-sparsifier of G with respect to w.

Our dynamic algorithm will be heavily based on the static algorithm from Section 3. We now introduce a couple of (informal) terms that relate to various aspects of this static algorithm. These terms will be very useful in the ensuing discussion. First, for each $i \in [0, L]$, the term *level-i-structure* will refer to the following sets: $E^{(\geq i)}, V^{(\geq i)}, V^{(i)}$ and $F^{(i)}$. Second, the term *hierarchy* will refer to the union of the level-i-structures over all $i \in [0, L]$.

We will maintain a partition of the edge-set E into two subsets: $E_{(a)}$ and $E_{(p)}$. The edges in $E_{(a)}$ (resp., $E_{(p)}$) will be called *active* (resp., *passive*). We will let $G_{(a)} := (V, E_{(a)})$ and $G_{(p)} := (V, E_{(p)})$ respectively denote the active and passive subgraphs of the input graph $G = (V, E)$. Our dynamic algorithm will make a *lazy attempt* at mimicking the static algorithm from Section 3, when the latter receives the active subgraph $G_{(a)}$ as input.

Preprocessing. At preprocessing, we set $E_{(p)} := \emptyset$ and $E_{(a)} := E$, and then call STATIC-UNIFORM-SPARSIFY$(G_{(a)} = (V, E_{(a)}), \lambda)$, as described in Algorithm 1. It returns the hierarchy, where for each $i \in [0, L]$ the level-i-structure consists of $E^{(\geq i)}, V^{(\geq i)}, F^{(i)}, V^{(i)}$. Finally, for each $i \in [0, L]$, we initialize a set $D^{(\geq i)} := \emptyset$. This concludes the preprocessing step.

Handling an edge-insertion. When an edge e gets inserted into the input graph $G = (V, E)$, we call the subroutine HANDLE-INSERTION(e, λ), as described in Algorithm 4. This classifies the edge e as passive, and sets $E_{(p)} \leftarrow E_{(p)} \cup \{e\}$. If the previous step does not violate Invariant 1, then we are done. Otherwise, if Invariant 1 gets violated, then we throw away the existing hierarchy and all its associated structures (such as the sets $D^{(\geq i)}$), and perform the preprocessing step again on the current input graph G.

▶ **Invariant 1.** $|E_{(p)}| \leq \epsilon \cdot |E_{(a)}|$.

Handling an edge-deletion. When an edge e gets deleted from G, we call the subroutine HANDLE-DELETION(e, λ), as described in Algorithm 6. If e was already passive, then it simply gets removed from the set $E_{(p)}$, and we are done. Henceforth, we assume that e was active, and at level $\ell(e) = k$, just before getting deleted.[4]

First, we remove e from the set $E_{(a)}$, because the edge is no longer present in G. Next, for every $i \in [0, k]$, we insert e into the set $D^{(\geq i)}$. From now on, we will refer to e as a *dead edge*. Intuitively, the edge e, even after getting deleted, continues to be present in the level-i-structure for each $i \in [0, k]$. Thus, up until this point, the hierarchy does not change.

Next, we check if the previous steps lead to a violation of Invariant 2. If Invariant 2 continues to remain satisfied, then we are done. Otherwise, we find the minimum index $j \in [0, L]$ such that $\left|D^{(\geq j)}\right| > \epsilon \cdot |E^{(\geq j)}|$, and then perform the following operations: (1) For every $i \in [j, L]$, we delete the dead edges $D^{(\geq i)}$ from the level-i-structure and reset $D^{(\geq i)} \leftarrow \emptyset$. (2) Finally, we call the subroutine REBUILD(j, λ) as described in Algorithm 2.

▶ **Invariant 2.** $\left|D^{(\geq i)}\right| \leq \epsilon \cdot \left|E^{(\geq i)}\right|$ *for all* $i \in [0, L]$.

[4] See the paragraph just before Observation 12 for the definition of the level of an edge.

Algorithm 4 HANDLE-INSERTION(e, λ).

```
$E_{(p)} \leftarrow E_{(p)} \cup \{e\}$.
if $|E_{(p)}| > \epsilon \cdot |E_{(a)}|$ then
    Call the subroutine CLEAN-UP$(0, \lambda)$.
    $E_{(a)} \leftarrow E_{(a)} \cup E_{(p)}$.
    $E_{(p)} \leftarrow \emptyset$.
    Call the subroutine STATIC-UNIFORM-SPARSIFY$\big(G_{(a)} := (V, E_{(a)}), \lambda\big)$.
```

Algorithm 5 CLEAN-UP(j, λ).

```
for all $i = j$ to $L$ do
    $E^{(\geq i)} \leftarrow E^{(\geq i)} \setminus D^{(\geq i)}$.
    $F^{(i)} \leftarrow F^{(i)} \setminus D^{(\geq i)}$.
    $D^{(\geq i)} \leftarrow \emptyset$.
```

Algorithm 6 HANDLE-DELETION(e, λ).

```
if $e \in E_{(p)}$ then
    $E_{(p)} \leftarrow E_{(p)} \setminus \{e\}$.
else
    $k \leftarrow \ell(e) := \max\{i \in [0, L] : e \in E^{(\geq i)}\}$.
    $E_{(a)} \leftarrow E_{(a)} \setminus \{e\}$.
    for $i = 0$ to $k$ do
        $D^{(\geq i)} \leftarrow D^{(\geq i)} \cup \{e\}$.
    if $|D^{(\geq i)}| > \epsilon \cdot |E^{(\geq i)}|$ for some index $i \in [0, L]$ then
        Let $j$ be the minimum index $i \in [0, L]$ for which $|D^{(\geq i)}| > \epsilon \cdot |E^{(\geq i)}|$.
        Call the subroutine CLEAN-UP$(j, \lambda)$.
        Call the subroutine REBUILD$(j, \lambda)$.
```

Note that $E_{(p)}, E_{(a)}, E^{(\geq i)}, D^{(\geq i)}$, $F^{(i)}$ represent global variables in these pseudocodes.

To summarize, we satisfy Invariant 1 and Invariant 2, and handle the updates to G in a lazy manner. Newly inserted edges are classified as passive, and they are completely ignored in the hierarchy unless their number becomes sufficiently large compared to the total number of active edges, at which point we rebuild everything from scratch. In contrast, when an active edge gets deleted from some level $i \in [0, L]$, it is classified as dead and it continues to be present in the level-j-structure for all $j \in [0, i]$. Finally, if we notice that for some $k \in [0, L]$ the level-k-structure has too many dead edges $D^{(\geq k)}$, then we remove all the dead edges from every level-j-structure with $j \in [k, L]$, and rebuild these structures from scratch.

From Section 3, recall that $F := \bigcup_{i=0}^{L} F^{(i)}$. For any set of edges E', we will use the notation $E'_{(a)} := E' \cap E_{(a)}$ to denote the subset of edges in E' that are active in the current input graph $G = (V, E)$. Accordingly, we define $F_{(a)} := F \cap E_{(a)}$ and $H_{(a)} := (V, F_{(a)})$.

Lemma 15 and Lemma 16 below should respectively be seen as analogues of Lemma 14 and Lemma 11 from Section 3. They show that the subgraph $H_{(a)} = (V, F_{(a)})$ is a good matching sparsifier of the input dynamic graph $G = (V, E)$. Intuitively, Lemma 15 and Lemma 16 hold because Invariant 1 and Invariant 2 ensure that throughout the sequence of updates, the hierarchy maintained by our dynamic algorithm is *very close* to the hierarchy constructed by the algorithm from Section 3 when it receives the current graph $G = (V, E)$ as input. Due to space constraints, the proofs of these two lemmas are deferred to the full version.

▶ **Lemma 15.** *The graph $H_{(a)} = (V, F_{(a)})$ has arboricity at most $O(\epsilon^{-1} + \beta^{-1}) = O(\log n)$.*

▶ **Lemma 16.** *The graph $H_{(a)}$ admits a fractional matching $h' : F_{(a)} \to [0, 1]$ such that:*
1. *For every edge $e \in F_{(a)}$, we have $h'(e) < \beta$.*
2. *For every node $v \in V$, we have $h'(v) \leq w(v)$.*
3. *We have $size(w) \leq (1 + 60\epsilon \cdot \log(\beta/\lambda)) \cdot size(h')$.*

▶ **Lemma 17.** *The dynamic algorithm* Dynamic-Uniform-Sparsify(G, λ) *has an amortized update time of $O\left(\epsilon^{-1} \cdot \log(\beta/\lambda)\right) = O(\log^2 n)$.*

Proof. Define a potential function $\Phi := |E_{(p)}| + \sum_{i=0}^{L} \left|D^{(\geq i)}\right|$. Insertion of an edge increases the potential Φ by at most one unit, as the newly inserted edge gets classified as passive. On the other hand, deletion of an edge e increases the potential Φ by at most $L+1$ units, since e gets added to each of the sets $D^{(\geq 0)}, \ldots, D^{(\geq \ell(e))}$, and $\ell(e) \leq L$. To summarize, each update in G creates at most $O(L)$ units of new potential. We will show that whenever our dynamic algorithm spends T units of time, the potential Φ drops by at least $\Omega(\epsilon \cdot T)$. Since Φ is always ≥ 0, this implies the desired amortized update time of $O(\epsilon^{-1} \cdot L) = O\left(\epsilon^{-1} \cdot \log(\beta/\lambda)\right)$.

Consider the insertion of an edge e into $G = (V, E)$, and suppose that we call Static-Uniform-Sparsify$(G_{(a)}, \lambda)$ while handling this insertion. Let $m_{(a)} := |E_{(a)}|$, $m_{(p)} := |E_{(p)}|$, $m_{(d)} := |D^{(\geq 0)}|$ and $m := |E|$, just before e gets inserted. Invariant 1 and Invariant 2 respectively ensure that $m_{(p)} = \epsilon \cdot m$ and $m_{(d)} \leq \epsilon \cdot m$. By Lemma 9, the call to Static-Uniform-Sparsify$(G_{(a)}, \lambda)$ takes $O(m)$ time. Thus, the total time to handle this edge insertion is given by $T := O(m + m_d) = O(m)$. On the other hand, when our algorithm finishes handling this edge insertion, we have $E_{(p)} = \emptyset$, and hence the potential Φ decreases by at least $m_{(p)} = \epsilon \cdot m$ units. In other words, the drop in the potential Φ is at least $\Omega(\epsilon \cdot T)$.

Next, consider the deletion of an edge e from $G = (V, E)$, and suppose that while handling this deletion we call the subroutine Rebuild(k, λ) for some $k \in [0, L]$. Just before e gets deleted, let $m_{(d)}^{(\geq k)} := \left|D^{(\geq k)}\right|$ and $m^{(\geq k)} := E^{(\geq k)}$. Invariant 2 ensures that $m_{(d)}^{(\geq k)} = \epsilon \cdot m^{(\geq k)}$. By Lemma 9, the call to Rebuild(k, λ) takes $O\left(m^{(\geq k)}\right)$ time. Hence, excluding the time it takes to identify the level k, which is $O(L) = O(\log(\beta/\lambda)) = O(\log n)$ in the worst-case, our dynamic algorithm spends $T = O\left(m^{(\geq k)}\right)$ time to handle this edge deletion. On the other hand, this decreases the potential Φ by at least $m_{(d)}^{(\geq k)} = \epsilon \cdot m^{(\geq k)}$, since once we are done processing this edge deletion, we have $D^{(\geq k)} = \emptyset$. So the potential Φ drops by $\Omega(\epsilon \cdot T)$. ◀

5 Dynamically Sparsifying an Arbitrary Fractional Matching

In this section, we briefly sketch our dynamic algorithm for maintaining a matching-sparsifier as specified by Theorem 1.

The input is a dynamic graph $G = (V, E)$ with n nodes, and a (*not* necessarily uniform) fractional matching $w : E \to [0, 1]$ in G. An "update" either inserts/deletes an edge in G or changes the weight $w(e)$ of an existing edge e in G. Our algorithm works in three steps.

Step I (Discretizing w). For every integer $j \geq 0$, define $\lambda_j := (\beta/n^2) \cdot (1+\beta)^j$. Let K be the largest integer j such that $\lambda_j < \beta$. We now discretize w to get a new fractional matching $\hat{w} : E \to [0, 1]$, which is defined as follows. Consider any edge $e \in E$. If $w(e) < \lambda_0$, then $\hat{w}(e) := 0$. Else if $\lambda_0 \leq w(e) < \beta$, then $\hat{w}(e) := \lambda_i$ where i is the unique integer such that $\lambda_i \leq w(e) < \lambda_{i+1}$. Otherwise, if $w(e) \geq \beta$, then $\hat{w}(e) := w(e)$.

For each $i \in [0, K]$, let E_i denote the subset of edges $e \in E$ with $\hat{w}(e) = \lambda_i$, let $G_i := (V, E_i)$, and let $w_i : E_i \to [0, 1]$ be the restriction of the fractional matching $\hat{w}$ onto the set E_i (i.e., w_i is a λ_i-uniform fractional matching in G_i). Finally, define the subset of edges $E_{\geq\beta} := \{e \in E : \hat{w}(e) = w(e) \geq \beta\}$, and let $G_{\geq\beta} := (V, E_{\geq\beta})$. In the dynamic setting, we can easily maintain the subgraphs $G_0, \ldots, G_K, G_{\geq\beta}$ of the input graph G *on the fly*.

Step II (Sparsifying each G_i). For each $i \in [0, K]$, we maintain a sparsifier of G_i with respect to w_i, with the help of the dynamic algorithm from Section 4. Specifically, let $H_i = (V, F_i)$ denote the sparsifier $H_{(a)} = (V, F_{(a)})$ maintained by the algorithm DYNAMIC-UNIFORM-SPARSIFY(G_i, λ_i) from Section 4 (thus, we have $F_i \subseteq E_i$).

Step III (Putting everything together). Let $E_S := \bigcup_{i=0}^{K} F_i \bigcup E_{\geq\beta}$. In the full version of the paper, we show that the subgraph $S := (V, E_S)$ of G satisfies all the five conditions stated in Theorem 1.

References

1 Amir Abboud, Raghavendra Addanki, Fabrizio Grandoni, Debmalya Panigrahi, and Barna Saha. Dynamic set cover: improved algorithms and lower bounds. In *STOC*, 2019.

2 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *FOCS*, 2014.

3 Moab Arar, Shiri Chechik, Sarel Cohen, Cliff Stein, and David Wajc. Dynamic matching: Reducing integral algorithms to approximately-maximal fractional algorithms. In *ICALP*, 2018.

4 Sepehr Assadi and Aaron Bernstein. Towards a unified theory of sparsification for matching problems. In *SOSA*, 2019.

5 S. Baswana, M. Gupta, and S. Sen. Fully dynamic maximal matching in $O(\log n)$ update time. In *FOCS*, 2011.

6 Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Cliff Stein, and Madhu Sudan. Fully dynamic maximal independent set with polylogarithmic update time. In *FOCS*, 2019.

7 Soheil Behnezhad, Jakub Lacki, and Vahab S. Mirrokni. Fully dynamic matching: Beating 2-approximation in $O(\Delta^\epsilon)$ update time. In *SODA*, 2020.

8 Aaron Bernstein, Sebastian Forster, and Monika Henzinger. A deamortization approach for dynamic spanner and dynamic maximal matching. In *SODA*, 2019.

9 Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental reachability, scc, and shortest paths via directed expanders and congestion balancing. In *FOCS*, 2020.

10 Aaron Bernstein and Cliff Stein. Fully dynamic matching in bipartite graphs. In *ICALP*, 2015.

11 Aaron Bernstein and Cliff Stein. Faster fully dynamic matchings with small approximation ratios. In *SODA*, 2016.

12 Sayan Bhattacharya, Deeparnab Chakrabarty, and Monika Henzinger. Deterministic fully dynamic approximate vertex cover and fractional matching in O(1) amortized update time. In *IPCO*, 2017.

13 Sayan Bhattacharya, Monika Henzinger, and Giuseppe F. Italiano. Deterministic fully dynamic data structures for vertex cover and matching. In *SODA*, 2015.

14 Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. New deterministic approximation algorithms for fully dynamic matching. In *STOC*, 2016.

15 Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. Fully dynamic approximate maximum matching and minimum vertex cover in $O(\log^3 n)$ worst case update time. In *SODA*, 2017.

16 Sayan Bhattacharya and Janardhan Kulkarni. Deterministically maintaining a $(2+\epsilon)$-approximate minimum vertex cover in $O(1/\epsilon^2)$ amortized update time. In *SODA*, 2019.

17 Moses Charikar and Shay Solomon. Fully dynamic almost-maximal matching: Breaking the polynomial barrier for worst-case time bounds. In *ICALP*, 2018.

18 Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. In *FOCS*, 2020.

19 Ran Duan, Haoqing He, and Tianyi Zhang. Dynamic edge coloring with improved approximation. In Timothy M. Chan, editor, *SODA*, 2019.

20 Jacob Evald, Viktor Fredslund-Hansen, Maximilian Probst Gutenberg, and Christian Wulff-Nilsen. Decremental APSP in directed graphs versus an adaptive adversary. *CoRR*, abs/2010.00937, 2020. `arXiv:2010.00937`.

21 Gramoz Goranci, Harald Räcke, Thatchaphol Saranurak, and Zihan Tan. The expander hierarchy and its applications to dynamic graph algorithms. In *SODA*, 2021.

22 Fabrizio Grandoni, Stefano Leonardi, Piotr Sankowski, Chris Schwiegelshohn, and Shay Solomon. $(1+\epsilon)$-approximate incremental matching in constant deterministic amortized time. In *SODA*, 2019.

23 Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Debmalya Panigrahi. Online and dynamic algorithms for set cover. In *STOC*, 2017.

24 Manoj Gupta. Maintaining approximate maximum matching in an incremental bipartite graph in polylogarithmic update time. In *FSTTCS*, 2014.

25 Manoj Gupta and Richard Peng. Fully dynamic $(1+\epsilon)$-approximate matchings. In *FOCS*, 2013.

26 Monika Henzinger and Valerie King. Randomized dynamic graph algorithms with polylogarithmic time per operation. In *STOC*, 1995.

27 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *STOC*, 2015.

28 Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, 2001.

29 Tsvi Kopelowitz, Robert Krauthgamer, Ely Porat, and Shay Solomon. Orienting fully dynamic graphs with worst-case time bounds. In *ICALP*, 2014.

30 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3sum conjecture. In *SODA*, 2016.

31 Kasper Green Larsen. The cell probe complexity of dynamic range counting. In Howard J. Karloff and Toniann Pitassi, editors, *STOC*, 2012.

32 Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. Dynamic minimum spanning forest with subpolynomial worst-case update time. In *FOCS*, 2017.

33 C St JA Nash-Williams. Edge-disjoint spanning trees of finite graphs. *Journal of the London Mathematical Society*, 1(1):445–450, 1961.

34 Ofer Neiman and Shay Solomon. Simple deterministic algorithms for fully dynamic maximal matching. In *STOC*, 2013.

35 Krzysztof Onak and Ronitt Rubinfeld. Maintaining a large matching and a small vertex cover. In *STOC*, 2010.

36 Mihai Patrascu. Towards polynomial lower bounds for dynamic problems. In *STOC*, 2010.

37 David Peleg and Shay Solomon. Dynamic $(1+\epsilon)$-approximate matchings: A density-sensitive approach. In *SODA*, 2016.

38 Piotr Sankowski. Faster dynamic matchings and vertex connectivity. In *SODA*, 2007.

39 Shay Solomon. Fully dynamic maximal matching in constant update time. In *FOCS*, 2016.

40 David Wajc. Rounding dynamic matchings against an adaptive adversary. In *STOC*, 2020.