Dynamic $(1+\epsilon)$-Approximate Matchings: A Density-Sensitive Approach

David Peleg* Shay Solomon†

Abstract

Approximate matchings in fully dynamic graphs have been intensively studied in recent years. Gupta and Peng [FOCS'13] presented a deterministic algorithm for maintaining fully dynamic $(1+\epsilon)$-approximate maximum cardinality matching (MCM) in general graphs with worst-case update time $O(\sqrt{m}\cdot\epsilon^{-2})$, for any $\epsilon > 0$, where m denotes the current number of edges in the graph. Despite significant research efforts, this $\sqrt{m}$ update time barrier remains the state-of-the-art even if amortized time bounds and randomization are allowed or the approximation factor is allowed to increase from $1+\epsilon$ to $2-\epsilon$, and even in basic graph families such as planar graphs.

This paper presents a simple deterministic algorithm whose performance depends on the *density* of the graph. Specifically, we maintain fully dynamic $(1+\epsilon)$-approximate MCM with worst-case update time $O(\alpha\cdot\epsilon^{-2})$ for graphs with *arboricity*[1] bounded by α. The update time bound holds even if the arboricity bound α changes dynamically. Since the arboricity ranges between 1 and $\sqrt{m}$, our density-sensitive bound $O(\alpha\cdot\epsilon^{-2})$ naturally generalizes the $O(\sqrt{m}\cdot\epsilon^{-2})$ bound of Gupta and Peng.

For the family of bounded arboricity graphs (which includes forests, planar graphs, and graphs excluding a fixed minor), in the regime $\epsilon = O(1)$ our update time reduces to a constant. This should be contrasted with the previous best 2-approximation results for bounded arboricity graphs, which achieve either an $O(\log n)$ worst-case bound (Kopelowitz et al., ICALP'14) or an $O(\sqrt{\log n})$ amortized bound (He et al., ISAAC'14), where n stands for the number of vertices in the graph.

En route to this result, we provide *local* algorithms of independent interest for maintaining fully dynamic approximate matching and vertex cover.

1 Introduction

1.1 Dynamic Matching. Graph matching constitutes one of the most well-studied problems in combinatorial optimization, with applications that extend far beyond the boundaries of computer science. Of particular importance is the problem of computing *maximum cardinality matchings* (MCMs), which has been subject to extensive research for over 50 years. On static graphs $G=(V,E)$, the state-of-the-art MCM algorithms, on bipartite graphs by Hopcroft and Karp [20] and on general graphs by Micali and Vazirani [30, 38], run in $O(m\sqrt{n})$ time (where $n=|V|, m=|E|$). Despite many attempts over the years, this $O(m\sqrt{n})$-time barrier has resisted improvement. The MCM algorithms of [20, 30, 38] actually provide $(1+\epsilon)$-approximate MCMs (or $(1+\epsilon)$-MCMs for short) in time $O(m/\epsilon)$. There is also a simple greedy $O(m)$-time algorithm for computing an *(inclusion-wise) maximal matching*, which provides a 2-approximate MCM (or 2-MCM).

In this paper we consider a dynamic setting, where the input graph G is being updated by a sequence of edge insertions and deletions over a fixed vertex set V; throughout the paper let n denote the number of vertices in V, and m the (current) number of edges. The basic question in this context is whether one can do better than simply applying the static algorithm from scratch following each edge update. For dynamic MCMs, the only known result is by Sankowski [36], who gave a randomized algorithm with a worst-case update time of $O(n^{1.495})$. For dynamic maximal matchings (and thus 2-MCMs), there is a naive deterministic algorithm with a worst-case update time of $O(n)$. Ivković and Lloyd [21] devised another maximal matching algorithm with an *amortized* update time of $O((n+m)^{\frac{\sqrt{2}}{2}})$.

There has been a growing interest in dynamic algorithms for *approximate matchings* since the pioneering paper by Onak and Rubinfeld [32], which gives a randomized algorithm for maintaining c-MCMs with an amortized update time of $O(\log^2 n)$, for a large unspecified constant c. Baswana et al. [6] devised a maximal matching algorithm with an expected amortized update time of $O(\log n)$. Neiman and Solomon [31] designed a deterministic algorithm for maintaining 3/2-MCMs with a worst-case update time of $O(\sqrt{m})$. A similar result was obtained independently by Anand [4]. Gupta and Peng [16] showed that a similar update time $O(\sqrt{m}\cdot\epsilon^{-2})$ suffices for maintaining $(1+\epsilon)$-MCMs, for any $\epsilon > 0$; in graphs of degree bounded by some parameter D, the update time of their algorithm reduces to

*The Weizmann Institute of Science, Rehovot, Israel. Email: david.peleg@weizmann.ac.il. Supported in part by the Israel Science Foundation (grant 1549/13), the I-CORE program of the Israel PBC and ISF (grant 4/11), and the United States-Israel Binational Science Foundation (grant 2008348).

†School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel. E-mail: solo.shay@gmail.com. Part of this work was done while the author was affiliated with the Weizmann Institute of Science, under the support of the Koshland Center for basic Research.

[1]The arboricity of a graph is the minimum number of edge-disjoint forests into which it can be partitioned, and it is close to the density of its densest subgraph.

$O(D \cdot \epsilon^{-2})$. We remark that this update time of [16] is deterministic and holds in the worst-case. Surprisingly, no better results are known (1) even if amortized time bounds and randomization are allowed, and (2) even if the approximation factor is allowed to increase from $1+\epsilon$ to $2-\epsilon$, for any $\epsilon > 0$. In fact, this result remains the state-of-the-art even in basic graph families such as planar graphs. The only exception is a result by Gupta and Sharma [17], showing that an MCM can be maintained in dynamic forests with a worst-case update time of $O(\log n)$. However, in contrast to all other results, the matching maintained in [17] is not represented *explicitly*: In order to determine if an edge belongs to the matching, one should run an $O(\log n)$-time query. Thus, strictly speaking, this $\sqrt{m}$ update time barrier holds even in dynamic forests.

The only deterministic algorithms with update time $o(\sqrt{m})$ have approximation factor greater than 3: Bhattacharya et al. [8] devised a dynamic algorithm for maintaining $(3+\epsilon)$-MCMs (respectively, $(4+\epsilon)$-MCMs) with an amortized (resp., worst-case) update time $O(\min\{\sqrt{n}/\epsilon, m^{1/3} \cdot \epsilon^{-2}\})$ (resp., $O(m^{1/3} \cdot \epsilon^{-2})$).

Arboricity vs. density. This paper presents a simple deterministic algorithm whose performance depends on the *density* of the graph. A graph $G = (V, E)$ has *arboricity* α if $\alpha = \max_{U \subseteq V} \left\lceil \frac{|E(U)|}{|U|-1} \right\rceil$, where $E(U) = \{(u, v) \in E \mid u, v \in U\}$. Thus the arboricity is close to the maximum *density* $|E(U)|/|U|$ over all induced subgraphs of G. While a graph of bounded arboricity is *uniformly sparse*, a graph of bounded density (i.e., a sparse graph) may contain a dense subgraph (e.g., on $\sqrt{m}$ of the vertices), and may thus have large arboricity. For this reason, the arboricity reveals additional information about the graph's inherent density. The family of bounded arboricity graphs contains planar and bounded genus graphs, bounded tree-width graphs, and in general all graphs excluding fixed minors.

Neiman and Solomon [31] also presented a dynamic maximal matching algorithm for graphs with arboricity bounded by α, based on a reduction to a certain *dynamic edge orientation* problem from [10]. The algorithm of [31] is deterministic, and its *amortized* update time for bounded arboricity graphs is $O(\log n / \log\log n)$. He et al. [19] established new results for the dynamic edge orientation problem, and consequently improved the amortized update time of [31] to $O(\sqrt{\log n})$; for general arboricity α, the update time of [19] is $O(\alpha + \sqrt{\alpha \log n})$. Kopelowitz et al. [25] obtained worst-case bounds for the dynamic edge orientation problem, and thus maintained maximal matchings with a worst-case update time of $O(\inf_{\beta>1}\{\alpha^2\beta^2 + \alpha\beta\lceil\log_\beta n\rceil\})$, which reduces to $O(\log n)$ for bounded arboricity graphs.

Our contribution. In this paper we devise a simple deterministic algorithm for dynamic $(1+\epsilon)$-MCMs with a worst-case update time $O(\alpha \cdot \epsilon^{-2})$, for graphs with arboricity bounded by α. Moreover, this time bound $O(\alpha \cdot \epsilon^{-2})$ continues to hold even if the arboricity bound α is dynamically changing. Since the arboricity of an m-edge graph ranges between 1 and $\sqrt{m}$, our density-sensitive bound $O(\alpha \cdot \epsilon^{-2})$ naturally generalizes the previous bound $O(\sqrt{m} \cdot \epsilon^{-2})$ due to Gupta and Peng. (See Table 1.)

For constant arboricity graphs, in the regime $\epsilon = O(1)$ our update time reduces to a constant. This result, of a $(1+\epsilon)$-MCM in constant worst-case update time, should be contrasted with the previous best 2-MCM results for constant arboricity graphs, which achieve either an $O(\log n)$ worst-case bound [25] or an $O(\sqrt{\log n})$ amortized bound [19].[2]

1.2 Dynamic Vertex Cover. While an MCM can be computed in time $O(m\sqrt{n})$, finding a minimum (cardinality) vertex cover (MCVC) is NP-hard. Nevertheless, these two problems remain closely related as their LP-relaxations are duals of each other.[3] Moreover, the set of matched vertices in a maximal matching is a 2-MCVC, thus the MCM and MCVC are of the same size up to a factor of 2. This means that all the aforementioned dynamic approximate MCM algorithms also provide dynamic approximate MCVC algorithms with a similar approximation factor. In addition, Bhattacharya et al. [8] presented a deterministic dynamic algorithm for $(2+\epsilon)$-MCVCs with an *amortized* update time $O(\log n \cdot \epsilon^{-2})$.

Our contribution. In this paper we devise a deterministic algorithm for dynamic $(2+\epsilon)$-MCVCs with a *worst-case* update time $O(\alpha/\epsilon)$, for graphs with arboricity bounded by α. For constant arboricity graphs, in the regime $\epsilon = O(1)$ our update time reduces to a constant. The previous best algorithms for dynamic $O(1)$-MCVCs in bounded arboricity graphs achieve either an $O(\log n)$ worst-case bound [25] or an $O(\sqrt{\log n})$ amortized bound [19]. (The algorithms of [25, 19] actually achieve a bet-

[2] Independently of us, Bernstein and Stein [7] devised an algorithm for maintaining $(3/2+\epsilon)$-MCMs in *bipartite graphs* with update time $O(m^{1/4} \cdot \epsilon^{-2.5})$. If the bipartite graph has arboricity bounded by α, a variant of their algorithm provides $(1+\epsilon)$-MCMs with update time $O(\alpha(\alpha + \log n) + \epsilon^{-4}(\alpha + \log n) + \epsilon^{-6})$, which reduces to $O(\log n)$ for constant arboricity and ϵ; our result is stronger and more general (applying to general bounded arboricity graphs rather than bipartite ones).

[3] Under the unique games conjecture, the MCVC cannot be efficiently approximated within any factor better than 2 [23]. So while the natural goal for MCM is 1 or $(1+\epsilon)$-approximation, the analogous goal for MCVC is 2 or $(2+\epsilon)$-approximation.

Reference	Approximation	Update time	Deterministic?	Worst-case?	Local?
Neiman and Solomon (STOC'13) [31]	3/2	$O(\sqrt{m})$	✓	✓	✓
Gupta and Peng (FOCS'13) [16]	$1+\epsilon$	$O(\sqrt{m}\cdot\epsilon^{-2})$	✓	✓	✗
Baswana et al. (FOCS'11) [6]	2	expected $O(\log n)$	✗	✗	✗
"	"	w.h.p. $O(\log^2 n)$	"	"	"
Bhattacharya et al. (SODA'15) [8]	$3+\epsilon$	$O(\min\{\sqrt{n}/\epsilon, m^{1/3}\cdot\epsilon^{-2}\})$	✓	✗	✗
"	$4+\epsilon$	$O(m^{1/3}\cdot\epsilon^{-2})$	✓	✓	✗
Kopelowitz et al. (ICALP'14) [25] for $\alpha=O(1)$	2	$O(\inf_{\beta>1}\{\alpha^2\beta^2+\alpha\beta\lceil\log_\beta n\rceil\})$	✓	✓	✗
"	"	$O(\log n)$	"	"	"
He et al. (ISAAC'14) [25] for $\alpha=O(1)$	2	$O(\alpha+\sqrt{\alpha\log n})$	✓	✗	✗
"	"	$O(\sqrt{\log n})$	"	"	"
This paper for $\alpha=O(1),\epsilon=O(1)$	$\mathbf{1+\epsilon}$	$\boldsymbol{(\alpha/\epsilon)^{O(1/\epsilon)}}$	✓	✓	✓
"	"	$\boldsymbol{O(1)}$	"	"	"
This paper for $\alpha=O(1),\epsilon=O(1)$	$\mathbf{3/2+\epsilon}$	$\boldsymbol{O(\alpha/\epsilon)}$	✓	✓	✓
"	"	$\boldsymbol{O(1)}$	"	"	"
This paper for $\alpha=O(1),\epsilon=O(1)$	$\mathbf{1+\epsilon}$	$\boldsymbol{O(\alpha\cdot\epsilon^{-2})}$	✓	✓	✗
"	"	$\boldsymbol{O(1)}$	"	"	"
Bernstein and Stein (ICALP'15) [7] (for bipartite graphs)	$3/2+\epsilon$	$O(m^{1/4}\cdot\epsilon^{-2.5})$	✓	✓	✗
Bernstein and Stein (ICALP'15) [7] (for bipartite graphs) for $\alpha=O(1),\epsilon=O(1)$	$1+\epsilon$	$O(\alpha(\alpha+\log n)+\epsilon^{-4}(\alpha+\log n)+\epsilon^{-6})$	✓	✓	✗
	"	$O(\log n)$	"	"	"

Table 1: A comparison of previous and our results for dynamic matchings, with α designating the arboricity bound. Our work is independent of the work by Bernstein and Stein [7], which is given in the last two rows of the table for completeness.

ter approximation factor of 2, but no faster algorithm is known *for any (non-trivial) approximation factor*. See Table 2.)

1.3 Local dynamic algorithms. When dealing with large-scale networks, it is important to devise algorithms that are intrinsically *local*. There is a vast body of literature on *local algorithms*, from various perspectives; see [27, 3, 37, 35, 29, 12, 13] and the references therein. A local algorithm in a dynamic network is one restricting its update operations following a local update to the vertices affected by the change and their immediate surroundings. Indeed, in many real-life applications, it is unrealistic to allow a global update procedure over a huge-sized network due to a single local update; even if such a procedure triggers only a few small changes in a far-away part of the network, implementing it is often prohibitively expensive, and sometimes even impossible (due to physical constraints). In such cases the update procedure is allowed to perform changes only within the radius-ℓ balls around the affected vertices, for a small parameter ℓ. In a distributed setting, a stronger requirement often arises: Following a dynamic update, only the affected vertices wake up, and only they are allowed to initiate the update procedure.

Among all previous dynamic algorithms for approximate MCM (and MCVC), the only local algorithms are the naive maximal matching algorithm and the 3/2-MCM algorithm of [31] (both providing 2-MCVC).

While our algorithm for dynamic $(1+\epsilon)$-MCMs with update time $O(\alpha\cdot\epsilon^{-2})$ is not local, our algorithm for dynamic $(2+\epsilon)$-MCVCs with update time $O(\alpha/\epsilon)$ is local. We also devise a local algorithm for dynamic $(1+\epsilon)$-MCMs with update time $(\alpha/\epsilon)^{O(1/\epsilon)}$; even though the former (non-local) algorithm is much faster, the locality of the latter may compensate for that in contexts as discussed above. A variant of our local algorithm provides dynamic $(3/2+\epsilon)$-MCMs with update time $O(\alpha/\epsilon)$. (See Table 1.) A natural application of our local algorithm for dynamic $(1+\epsilon)$-MCMs is to (dynamic) distributed networks. Specifically, the *message complexity* of our update procedure in distributed networks of arboricity bounded by α is $(\alpha/\epsilon)^{O(1/\epsilon)}$; the *round complexity* (or update time) is much better, namely, $O(1/\epsilon)$, even in general distributed networks. We remark that in static distributed networks, the state-of-the-art algorithms for $(1+\epsilon)$-MCMs require either $O(\log n\cdot\epsilon^{-3})$ time [27] or $\Delta^{O(1/\epsilon)}+O(\epsilon^{-2}\cdot\log^* n)$ time [12], where Δ is the maximum degree in the graph; the message complexity of these algorithms [27, 12] was not analyzed.

1.4 Preliminaries. Throughout the paper we consider graph sequences $\mathcal{G}=(G_0,G_1,\ldots,G_k)$ on a fixed n-vertex set V, where the initial graph G_0 is empty, and each graph G_i is obtained from the previous graph G_{i-1} in the sequence by either adding or deleting a

Reference	Approximation	Update time	Deterministic?	Worst-case?	Local?
Neiman and Solomon (STOC'13) [31]	2	$O(\sqrt{m})$	✓	✓	✓
Baswana et al. (FOCS'11) [6]	2	expected $O(\log n)$	✗	✗	✗
"	"	w.h.p. $O(\log^2 n)$	"	"	"
Bhattacharya et al. (SODA'15) [8]	$2+\epsilon$	$O(\log n \cdot \epsilon^{-2}\})$	✓	✗	✗
Kopelowitz et al. (ICALP'14) [25]	2	$O(\inf_{\beta>1}\{\alpha^2\beta^2 + \alpha\beta\lceil\log_\beta n\rceil\})$	✓	✓	✗
for $\alpha = O(1)$	"	$O(\log n)$	"	"	"
He et al. (ISAAC'14) [25]	2	$O(\alpha + \sqrt{\alpha \log n})$	✓	✗	✗
for $\alpha = O(1)$	"	$O(\sqrt{\log n})$	"	"	"
This paper	$\boldsymbol{2+\epsilon}$	$\boldsymbol{O(\alpha/\epsilon)}$	✓	✓	✓
for $\boldsymbol{\alpha = O(1), \epsilon = O(1)}$	"	$\boldsymbol{O(1)}$	"	"	"

Table 2: A comparison of previous and our results for dynamic vertex covers, with α designating the arboricity bound.

single edge. We say that such a graph sequence $\mathcal{G}$ has arboricity (at most) α if all graphs G_i in it have arboricity at most α.[4]

A vertex is called *matched* if it is incident on some edge of $\mathcal{M}$. Otherwise it is *free*. An *augmenting path* is an *alternating path* (with edges alternating between $\mathcal{M}$ and $E \backslash \mathcal{M}$) that starts and ends at different free vertices. A *length-ℓ* (augmenting) path is an (augmenting) path of length at most ℓ.

For a vertex v in G, let $\Gamma_G(v)$ denote the set of neighbors (or *neighborhood*) of v in G.

FACT 1.1. **[20, 30, 38]** *A $(1+\epsilon)$-MCM can be computed (statically) in $O(m/\epsilon)$ time for any m-edge graph.*

1.5 Overview. We now review the main previous approaches and the new ideas used in our algorithms.

Almost-maximal matchings. At the core of most dynamic matching algorithms lies the following naive dynamic maximal matching algorithm. Following an edge update $e = (u, v)$, if e is inserted to the graph and its two endpoints u and v are free, then the algorithm adds e to the maximal matching $\mathcal{M}$. Analogously, if $e \in \mathcal{M}$ and it is deleted from the graph, then the algorithm removes it from $\mathcal{M}$. Next, let $w \in \{u, v\}$. Assuming w is free, the algorithm scans *all* neighbors of w looking for a free vertex. If a free neighbor z of w is found, then the edge (w, z) is added to $\mathcal{M}$. Clearly, the update time is $O(\deg(u)+\deg(v))$, which may be as high as $O(n)$, even in forests. On the bright side, this bound holds in the worst-case and it is deterministic. Moreover, this update algorithm is inherently local, scanning only u, v and their immediate neighbors.

[4]We assume for simplicity that the arboricity of the graph sequences $\mathcal{G}$ considered in this paper is upper bounded by an arbitrary *static* parameter α. In Sect. 5 we show how to adapt our algorithms to cope with dynamic arboricity.

Is a "complete scan" from scratch (of *all* neighbors of the free updated vertices) necessary? Allowing both randomization and amortization, the answer is no: An expected $O(\log n)$ amortized update time is known [6]. However, if one insists on either deterministic or worst-case bounds, then no algorithm avoiding a complete scan is known. The state-of-the-art update time of $O(\sqrt{m})$ was achieved by guaranteeing that all free vertices have degree $O(\sqrt{m})$ [31], thus ensuring that the complete scan is not too costly.

What about basic graph families, such as forests? Since most vertices in a forest have constant degree, it seems reasonable to expect an $O(1)$-time update algorithm, as we have for bounded degree graphs. Alas, it is possible that most of the "action" (insertions and deletions) will concentrate on the few high degree vertices, thus pushing up the update costs of the algorithm. Indeed, the problem of $O(1)$-time maintenance of maximal matchings in dynamic forests has been open for many years.

A natural strategy for bypassing this obstacle is to restrict attention to the subgraph $\hat{G}$ of G induced by the low degree vertices, e.g., the vertices with degree $O(1/\epsilon)$ for suitable ϵ, and maintain a maximal matching dynamically w.r.t. this subgraph. Since only an ϵ-fraction of the vertices have degree $\omega(1/\epsilon)$, the subgraph $\hat{G}$ must contain almost all vertices of the entire graph G. Thus intuitively, a maximal matching w.r.t. $\hat{G}$ should provide a "good" matching w.r.t. G. This intuition, however, is flawed. Consider a perfect k-ary tree $T = T_k$, for some parameter $k = \omega(1/\epsilon)$; while the low degree subgraph $\hat{T}$ of T is an *empty* graph over the leaves of T, the size of an MCM for T is $\Theta(n/k)$.

Our first insight is that a partial (rather than complete) scan suffices for obtaining an *almost-maximal matching (AMM)*. We say that a matching for G is *almost-maximal* (w.r.t. some slack parameter ϵ) if it is maximal w.r.t. any graph obtained from G after

removing $\epsilon \cdot |\mathcal{M}^*|$ arbitrary vertices, where $\mathcal{M}^*$ is an MCM for G. Just as a maximal matching is also a 2-MCM, an AMM is a $(2+O(\epsilon))$-MCM. More specifically, we consider graphs whose arboricity is bounded by some parameter α, and set the degree threshold as $D = \Theta(\alpha/\epsilon)$, separating the vertices into low degree vertices, namely, ones with degree less than D, and high degree vertices, whose degree is at least D. A partial scan simply goes over (up to) D *arbitrary* neighbors of the (at most two) free updated vertices. This will clearly bound the update time by $O(\alpha/\epsilon)$. We next briefly explain why a partial scan should suffice for obtaining an AMM. Denote by F and M the sets of free and matched vertices w.r.t. the updated matching $\mathcal{M}$ for the (dynamically changing) graph G, and let F^{low} and F^{high} denote the sets of low and high degree free vertices, respectively. First note that edges connecting two vertices of F^{low} *must* be detected by the partial scan, and can thus be prevented. This already guarantees that the matching $\mathcal{M}$ will be maximal w.r.t. the low degree subgraph $\hat{G}$. To obtain an AMM, it suffices to guarantee that $|F^{high}| = O(\epsilon) \cdot |M|$. To this end, one could try maintaining the invariant that each vertex v of F^{high} has at least D matched neighbors. However, this approach is doomed to fail: When a high degree matched neighbor w of v becomes free, we cannot afford to scan all its neighbors for a free vertex as part of the partial scan. Hence w may become free, even though it may have free neighbors like v. Instead, we maintain the weaker invariant that each vertex of F^{high} has at least D neighbors outside F^{low}; in Sect. 2 we demonstrate that this weaker invariant is *automatiaclly* maintained as part of the partial scan (in time $O(\alpha/\epsilon)$), and moreover, we show that it suffices for guaranteeing that $|F^{high}| = O(\epsilon) \cdot |M|$.

We remark that the vertex set $M \cup F^{high}$ (maintained by our dynamic AMM algorithm) is a $(2+\epsilon)$-MCVC. In this way we get a dynamic $(2+\epsilon)$-MCVC algorithm with a worst-case update time of $O(\alpha/\epsilon)$.

From almost-maximal to almost-maximum matchings. As mentioned, AMMs (and thus $(2+\epsilon)$-MCMs) can be dynamically maintained via a *local* update algorithm whose runtime is linear in the graph's arboricity and $1/\epsilon$. Our second insight is that dynamic AMMs provide a useful tool for obtaining dynamic $(1+\epsilon)$-MCMs. In fact, we present two inherently different approaches for maintaining dynamic $(1+\epsilon)$-MCMs using dynamic AMMs.

The first approach is local, and is based on dynamically excluding *short* augmenting paths, i.e., of length bounded by $\ell = \Theta(1/\epsilon)$. Following a single edge update, length-ℓ augmenting paths may be created. In Sect. 3 we show that it suffices to augment the matching along two *carefully chosen* length-ℓ augmenting paths (so that afterwards no length-ℓ augmenting paths may exist), and devise a *local* procedure `Augment` for computing these paths. In particular, for graphs with degree bounded by D, the runtime of this procedure is $D^{O(\ell)}$. While our dynamic graph G has an arboricity bound of α, its maximum degree may be $\Omega(n)$. The crux of the problem is to dynamically maintain a bounded degree *matching sparsifier*, i.e., a subgraph of G for which the MCM is of roughly the same size (up to a factor of $1+\epsilon$). Let M^{low} (respectively, M^{high}) be the set of matched vertices with fewer than (resp., at least) D neighbors outside F^{low}. We first show that the subgraph $G^{low} = G[F^{low} \cup M^{low}]$ of G induced by $F^{low} \cup M^{low}$ is a matching sparsifier for G. Next, although this sparsifier is not a bounded degree graph, we observe that it is close to being one. In particular, it can be shown that running Procedure `Augment` on top of this sparsifier takes $D^{O(\ell)}$ time. More specifically, it is unclear how to dynamically maintain this sparsifier *explicitly* within the same time, as it contains high degree vertices. The key observation is that there is no need to represent the sparsifier explicitly – it suffices to exclude length-ℓ augmenting paths restricted to it, and this can be done "on the fly": When traversing any vertex v along such a path, we simply scan its neighbors outside F^{low}. If there are at least D such neighbors, then we backtrack, as v does not belong to G^{low}. Otherwise v belongs to G^{low}, and we can either continue the path via the (fewer than D) edges to v's matched neighbors, or complete the path via a single edge (out of possibly many) to a free neighbor. In this way we can implement Procedure `Augment` within $D^{O(\ell)} = (\alpha/\epsilon)^{O(1/\epsilon)}$ time.

The second approach is global, and is based on the "lazy" scheme of [16]. Since the size of an MCM changes by at most 1 following each update, one can compute a $(1+\epsilon)$-MCM $\mathcal{M}$ at a certain update, and then use it as a $(1+O(\epsilon))$-MCM throughout the subsequent $\epsilon \cdot |\mathcal{M}|$ updates (gradually removing from $\mathcal{M}$ its edges that get deleted from the graph). In this way the static time complexity $O(m/\epsilon)$ of computing $(1+\epsilon)$-MCM is amortized over $\epsilon \cdot |\mathcal{M}|$ updates. This yields amortized time $O(m/(|\mathcal{M}|\epsilon^2))$, which is $\omega(\sqrt{m} \cdot \epsilon^{-2})$ when $|\mathcal{M}| = o(\sqrt{m})$. The key insight behind the scheme of [16] is that one can compute in time $O(|\mathcal{M}|^2)$ a matching sparsifier for G of size $O(|\mathcal{M}|^2)$. With such a sparsifier G', a $(1+\epsilon)$-MCM for G can be computed in $O(\min\{m, |\mathcal{M}|^2\}/\epsilon)$ time, and the amortized time is reduced to $O(\min\{m, |\mathcal{M}|^2\}/\epsilon)/(\epsilon \cdot |\mathcal{M}|) = O(\sqrt{m} \cdot \epsilon^{-2})$.[5] The matching sparsifier G' is derived from an

[5] As shown in [16], this amortized bound can be easily translated into the same (up to a constant factor) worst-case bound. Nevertheless, in this intuitive discussion concerning the lazy

$O(1)$-MCVC VC for G. Specifically, $G' = (V', E')$ contains (1) all edges in the subgraph $G[VC]$ induced by VC, and (2) for each vertex $v \in VC$, (up to) $|VC|+1$ edges connecting it with arbitrary neighbors outside VC. As shown in [16], G' is indeed a sparsifier and $|E'| = O(|VC|^2) = O(|\mathcal{M}|^2)$. For this scheme to work, one needs to dynamically maintain $O(1)$-MCVC with update time $O(\sqrt{m} \cdot \epsilon^{-2})$; such a result is known [31].

To dynamically maintain a $(1+\epsilon)$-MCM with update time $O(\alpha \cdot \epsilon^{-2})$, we make two critical adjustments to the above scheme. The first adjustment is to plug our own density-sensitive $(2+\epsilon)$-MCVC $\widetilde{VC}$ instead of the ordinary 2-MCVC VC of [31]. As a result, the cost of maintaining an $O(1)$-MCVC in the above scheme is reduced to $O(\alpha/\epsilon)$. The second adjustment is in computing a density-sensitive matching sparsifier $\tilde{G}$. Specifically, our sparsifier $\tilde{G} = (\tilde{V}, \tilde{E})$ contains (1) all edges in the subgraph $G[\widetilde{VC}]$ induced by $\widetilde{VC}$, and (2) for each vertex $v \in \widetilde{VC}$, (up to) $D = O(\alpha/\epsilon)$ (rather than $|VC|+1$ as before) edges connecting it with arbitrary neighbors outside $\widetilde{VC}$. (See Figure 1.) It is easy to see

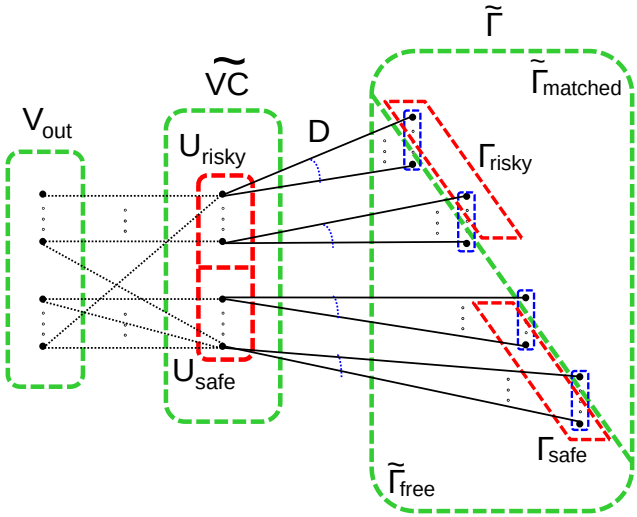

Figure 1: Illustration of the matching sparsifier. $\tilde{\Gamma} = \tilde{V} \setminus \widetilde{VC}$ is the set of vertices of $\tilde{G}$ outside $\widetilde{VC}$, and V_{out} denotes the set of remaining vertices of G outside $\tilde{G}$. Edges touching V_{out} (dotted) do not belong to $\tilde{G}$.

that $|\tilde{E}| = O(|\mathcal{M}| \cdot \alpha/\epsilon)$. Plugging this bound instead of the bound $O(|\mathcal{M}|^2)$ of [16] reduces the amortized update time from $O(\sqrt{m} \cdot \epsilon^{-2})$ to $O(\alpha \cdot \epsilon^{-3})$. Shaving a factor of ϵ^{-1} from this bound requires some extra effort (see Sect. 4.3). Since the arboricity is always upper bounded by the size of an MCVC (see Appendix A), our density-sensitive matching sparsifier $\tilde{G}$ is always sparser (up to a factor of $O(\epsilon^{-1})$) than that of [16]. The challenge is then to show that our (possibly much sparser) subgraph $\tilde{G}$ is indeed a matching sparsifier.

1.6 Related Work. Exact MCM algorithms in incremental bipartite graphs were studied in [9, 15]. There is a large body of literature on approximate MCM streaming algorithms in bipartite graphs; see [2, 24, 14, 22], and the references therein. Static and dynamic algorithms for exact and approximate maximum weighted matchings have been extensively studied, both in centralized and distributed networks (see, e.g., [27, 34, 5, 16, 12, 11, 15]). Conditional lower bounds for exact and approximate MCMs were given in [1, 26]. In particular, Abboud and Vassilevska Williams [1] showed that dynamic algorithms for approximate MCMs that are based on excluding short augmenting paths are inherently limited, in the sense that they can be converted into static algorithms for several fundamental problems (namely, 3SUM, triangle detection and Boolean matrix multiplication). Consequently, an update time of $O(m^\epsilon)$, for a small enough constant $\epsilon > 0$, would imply a breakthrough on these problems. For graphs with arboricity bounded by α, though, the approach of [1] can only lead to a lower bound of $\Omega(\alpha^2)$ on the update time.

2 Dynamic $(2+\epsilon)$-MCMs and $(2+\epsilon)$-MCVCs

In this section we present an algorithm for dynamically maintaining $(2+\epsilon)$-MCMs. Our algorithm is deterministic and its worst-case update time is $O(\alpha/\epsilon)$, for any graph sequence with arboricity α.

Let $\mathcal{G} = (G_0, G_1, \ldots, G_k)$ be an arbitrary graph sequence on V with arboricity $\alpha \geq 1$. Suppose that $0 < \epsilon \leq 1/2$, and set $D = 8\alpha/\epsilon$. Denote by $\mathcal{M}$ the matching maintained by our algorithm, and let $\mathcal{M}_i$ be the value of $\mathcal{M}$ for the ith graph G_i. Let $M = V(\mathcal{M})$ (respectively, $M_i = V(\mathcal{M}_i)$) denote the set of vertices incident to the edges of the matching $\mathcal{M}$ (resp., $\mathcal{M}_i$). A vertex is *free* in G_i if it does not belong to M_i. At the outset of the execution, the graph G_0 and the matching $\mathcal{M}_0$ are empty, and all vertices are free. The algorithm proceeds in rounds. In each round $i = 1, 2, \ldots$, a single edge e_i is either added to the graph G_{i-1} or deleted from it, to form G_i, and the algorithm updates the matching $\mathcal{M} = \mathcal{M}_{i-1}$ to $\mathcal{M}_i$ as follows. If the edge e_i is inserted to the graph and its two endpoints u and v are free, then the algorithm adds the edge (u, v) to the matching $\mathcal{M}$. Symmetrically, if e_i is in the matching $\mathcal{M}_{i-1}$ and is deleted from the graph, then the algorithm removes

scheme, we will concentrate on amortized bounds.

it from $\mathcal{M}$. Next, let $w \in \{u, v\}$. Assuming w is free, the algorithm scans the neighbors of w looking for a free vertex, and stops once D neighbors are scanned. If a free neighbor z of w was found, then the edge (w, z) is added to the matching $\mathcal{M}$. This concludes the description of the ith round of the algorithm. Clearly the runtime per round is $O(D) = O(\alpha/\epsilon)$.

We argue that the following invariants are preserved at the end of each round i. Denote by F_i^{low} (respectively, F_i^{high}) the free vertices in G_i whose degree is less than (resp., at least) D. (We sometimes omit the subscript i when it is clear from the context.)

INVARIANT 1. *The vertices of F_i^{low} form an independent set in the graph G_i.*

INVARIANT 2. *Every vertex in F_i^{high} has at least D neighbors outside F_i^{low} (i.e., in $M_i \cup F_i^{high}$).*

Invariants 1 and 2 clearly hold for G_0 and $\mathcal{M}_0$, before the first round starts and edge e_1 is handled. Indeed, at this stage (1) the graph is empty, and (2) all vertex degrees are zero (and so $F_i^{high} = \emptyset$).

LEMMA 2.1. *Assuming Invariants 1 and 2 hold at the beginning of the ith round, for $i = 1, 2, \ldots$, they will continue to hold at the end of the round.*

Proof. We start with making the following observation.

OBSERVATION 1. *The only vertices that may turn from matched to free during the ith round are u and v (due to the deletion of edge (u, v)). Moreover, only u and v may change their degree during this round.*

We turn to proving the validity of Invariant 1. Suppose for contradiction that this invariant holds at the beginning of the ith round, but is violated at the end of the round. Thus, there are two vertices x and y in F_i^{low} that are incident in G_i at the end of the round. How could this have happened?

Case 1: the edge (x, y) has been added on the ith round. Since $e_i = (u, v)$ is the only edge that has been added at the ith round, we have $(x, y) = (u, v)$. However, in this case the algorithm would add edge (x, y) to the matching, hence both x and y would be matched at the end of the round, a contradiction.

Case 2: at least one among x or y, without loss of generality x, was outside F^{low} at the beginning of the ith round, but moved to F^{low} by the end of the round. This can be either because x was previously matched and became free, or because its degree decreased from at least D to some $D' < D$. By Observation 1, x is either u or v. However, the algorithm would scan all D' neighbors of x, and would necessarily find a free neighbor (as y is one) and match x with it, leading again to a contradiction.

Next, we prove the validity of Invariant 2. Suppose for contradiction that this invariant holds at the beginning of the ith round, but is violated at the end of that round. Thus, there is a vertex x in F^{high} with at most $D - 1$ neighbors outside F^{low} at the end of that round. How could this have happened?

Case 1: x was in F^{high} at the beginning of round i, but with at least D neighbors outside F^{low}. We know that x has less than D neighbors outside F^{low} at the end of the round. There are two subcases.

Case 1.a: At least one neighbor y of x moved to F^{low} during the ith round. This can be either because y was previously matched and became free, or because its degree decreased from at least D to some $D' < D$. By Observation 1, y is either u or v. However, the algorithm would scan all D' neighbors of y, and would necessarily find a free neighbor (as x is one) and match y with it, a contradiction.

Case 1.b: An edge between x and some vertex y outside F^{low} has been deleted during the ith round. Since edge $e_i = (u, v)$ is the only edge that has been deleted at the ith round, we have $(x, y) = (u, v)$. However, in this case the algorithm would scan up to D neighbors of x for a free neighbor. If a free neighbor were found, then x would be matched with it, a contradiction. This means that x must have at least D matched neighbors, all of which are by definition outside F^{low}, leading again to a contradiction.

Case 2: x was matched at the beginning of round i, and became free during the round. By Observation 1, x is either u or v. However, similarly to above, the algorithm would scan up to D neighbors of x for a free neighbor. If a free neighbor were found, then x would be matched with it, a contradiction. Thus x has at least D matched neighbors, which are by definition outside F^{low}, leading again to a contradiction.

Case 3: x was in F^{low} at the beginning of round i, and moved to F^{high} during the round. In this case the degree of x increased from below D to at least D during the round. By Observation 1, x is either u or v. As above, the algorithm would scan up to D neighbors of x, and we will reach the same contradiction.

This completes the proof of Lemma 2.1. ∎

We next show that Invariants 1 and 2 imply that the maintained matching $\mathcal{M}_i$ is a $(2+\epsilon)$-MCM in graph G_i. Formally, letting $\mathcal{M}_i^*$ be an MCM for the graph G_i, we have:

LEMMA 2.2. *If Invariants 1 and 2 hold at all times, then $|\mathcal{M}_i^*| \leq (2 + \epsilon) \cdot |\mathcal{M}_i|$ for every i.*

Proof. In what follows we omit the subscript i. Consider the graph $G' = G[V']$ induced by $V' = V \setminus F^{low}$. Let $M = M^{low} \cup M^{high}$, where M^{low} (resp., M^{high}) is the set of matched vertices whose degree in G' is less than (resp., at least) D.

The following claim is used to prove Lemma 2.2. We remark that the proof of this claim relies on the fact that the graph's arboricity is always bounded by α.

CLAIM 1. $|F^{high}| + |M^{high}| \leq \frac{\epsilon}{2} \cdot |M^{low}|$.

Remark. For proving Lemma 2.2 it suffices to obtain a weaker upper bound, namely $|F^{high}| \leq \frac{\epsilon}{2} \cdot |M|$. However, we use this stronger bound for obtaining a $(1+\epsilon)$-MCM later (see the proof of Lemma 3.2).

Proof. By Invariant 2, the degree of each vertex of F^{high} in G' is at least D. Also, the degree of each vertex of M^{high} in G' is at least D by definition. Thus the number of edges that are incident on vertices from F^{high} and M^{high} in G' is at least $(|F^{high}| + |M^{high}|) \cdot D/2$. On the other hand, we know that the arboricity of G' is bounded by the arboricity bound α for the entire graph G, implying that the total number $|E(G')|$ of edges in G' is at most $\alpha \cdot |V'|$. It follows that

$$\begin{aligned}(|F^{high}| + |M^{high}|) \cdot D/2 \;&\leq\; |E(G')| \;\leq\; \alpha \cdot |V'| \\ &=\; \alpha \cdot ((|F^{high}| + |M^{high}|) + |M^{low}|),\end{aligned}$$

hence

$$(1 - \frac{\epsilon}{4}) \cdot (|F^{high}| + |M^{high}|) \;\leq\; \frac{\epsilon}{4} \cdot |M^{low}|.$$

As $\epsilon < 1$, the claim follows. ∎

We proceed with the proof of the lemma. Claim 1 implies that

$$\begin{aligned}|F^{high}| \;&\leq\; |F^{high}| + |M^{high}| \\ &\leq\; \frac{\epsilon}{2} \cdot |M^{low}| \;\leq\; \frac{\epsilon}{2} \cdot |M|. \qquad (2.1)\end{aligned}$$

Let $\mathcal{M}^*$ be an arbitrary MCM in the graph. By Invariant 1, each edge of $\mathcal{M}^*$ is incident on at least one vertex outside F^{low}. Hence

$$|\mathcal{M}^*| \;\leq\; |V'| \;=\; |F^{high}| + |M| \;\leq\; (1 + \frac{\epsilon}{2}) \cdot |M|.$$

(The last inequality follows from Equation (2.1).) Since $|\mathcal{M}| = |M|/2$, we have $|\mathcal{M}^*| \leq (2+\epsilon) \cdot |\mathcal{M}|$. Lemma 2.2 follows. ∎

Almost-maximal matchings, and their induced vertex cover. Invariant 1 implies that $\mathcal{M}$ is a maximal matching for the graph $G[V \setminus F^{high}]$ induced by the vertex set $V \setminus F^{high}$. By Equation (2.1),

$$F^{high} \;\leq\; \epsilon/2 \cdot |M^{low}| \;\leq\; \epsilon \cdot |\mathcal{M}^*|.$$

Hence $\mathcal{M}$ is a maximal matching for *almost* the entire graph, i.e., it is maximal w.r.t. a subgraph obtained from the original graph after removing $\epsilon \cdot |\mathcal{M}^*|$ arbitrary vertices; we say that such a matching is *almost-maximal*. While a maximal matching is a 2-MCM, an almost-maximal matching (AMM) is a $(2+\epsilon)$-MCM. Moreover, our AMM directly translates into a $(2+\epsilon)$-MCVC.

LEMMA 2.3. *The vertex set $VC = M \cup F^{high}$ is a $(2+\epsilon)$-MCVC for G.*

Proof. Since $\mathcal{M}$ is a maximal matching for the graph $G[V \setminus F^{high}]$, any edge of G with two endpoints outside F^{high} must be incident on at least one vertex of M. Consequently, VC is a vertex cover for G. Let VC^* be an optimal vertex cover for G. For each edge $e \in \mathcal{M}$, at least one of its endpoints must belong to VC^*, implying that $|VC^*| \geq |\mathcal{M}| = |M|/2$. Thus, by Equation (2.1),

$$\begin{aligned}|VC| \;&=\; |M| + |F^{high}| \;\leq\; \left(1 + \frac{\epsilon}{2}\right) \cdot |M| \\ &\leq\; (2+\epsilon) \cdot |VC^*|. \qquad ∎\end{aligned}$$

Our results for density-sensitive $(2+\epsilon)$-MCMs and $(2+\epsilon)$-MCVCs are summarized in the following statement.

COROLLARY 2.1. *For graphs with arboricity bounded by α, $(2+\epsilon)$-MCMs and $(2+\epsilon)$-MCVCs can be maintained dynamically with worst-case update time $O(\alpha/\epsilon)$.*

3 Dynamic $(1+\epsilon)$-MCMs: The Local Approach

In this section we present a local algorithm for dynamically maintaining $(1+\epsilon)$-MCMs with worst-case update time $(\alpha/\epsilon)^{O(1/\epsilon)}$. We start (Sect. 3.1) by providing a local procedure for dynamically excluding short augmenting paths (in general graphs), thus yielding $(1+\epsilon)$-MCMs. We then show an efficient implementation of this procedure in bounded degree graphs as well as in general distributed networks.

We proceed (Sect. 3.2) by showing a simple reduction from bounded arboricity graphs to bounded degree graphs. This reduction carefully builds on top of the algorithm from Sect. 2 and its analysis.

3.1 Local dynamic procedure for excluding short augmenting paths. Our goal is to dynamically maintain matchings $\mathcal{M}$ which exclude length-ℓ augmenting paths, for a parameter $\ell \leq n-1$ that determines the approximation guarantee of the matching.

Due to the following well-known fact, this would imply that our maintained matching is a $(1+\epsilon)$-MCM, for $\epsilon = 2/(\ell+1)$.

FACT 3.1. **[20]** *If the shortest augmenting path w.r.t. $\mathcal{M}$ has length $2k+1$, then $|\mathcal{M}^*| \leq (1+1/k)\cdot|\mathcal{M}|$.*

Consider an arbitrary edge update (u,v). It is guaranteed that there are no length-ℓ augmenting paths in the graph just before the update w.r.t. the maintained matching $\mathcal{M}$. However, there may be multiple length-ℓ augmenting paths w.r.t. $\mathcal{M}$ in the updated graph (i.e., after the update).

Next, we devise a simple procedure, Procedure `Augment`, for computing (at most) two length-ℓ augmenting paths Q_1 and Q_2, such that after augmenting the matching $\mathcal{M}$ along these paths, no length-ℓ augmenting path may exist. This procedure is local, examining just the ℓ-radius balls centered around u and v. We remark that a similar procedure was given in [28] for a related setting, which concerns vertex updates rather than edge updates.

Procedure `Augment` starts by computing a shortest length-ℓ augmenting path P_u w.r.t. $\mathcal{M}$ traversing u but not v. If such a path P_u is found, i.e., $P_u \neq \perp$, then the procedure computes a shortest length-ℓ augmenting path P'_v w.r.t. $\mathcal{M}'_u = \mathcal{M} \oplus P_u$ traversing v. In this case, Procedure `Augment` terminates with $Q_1 = P_u, Q_2 = P'_v$. (It is possible that $Q_2 = P'_v = \perp$.)

Otherwise $P_u = \perp$, and the procedure computes a shortest length-ℓ augmenting path P_v w.r.t. $\mathcal{M}$ traversing v. In this case, the procedure terminate with $Q_1 = \perp, Q_2 = P_v$. (It is possible that $Q_2 = P_v = \perp$.)

To prove the correctness of Procedure `Augment`, we use the following well-known fact.

FACT 3.2. **[20]** *Let $\mathcal{M}$ and $\mathcal{M}'$ be matchings in any graph $G=(V,E)$, with $|\mathcal{M}'| > |\mathcal{M}|$. Then the subgraph $H = (V, \mathcal{M} \oplus \mathcal{M}')$ contains at least $|\mathcal{M}'| - |\mathcal{M}|$ vertex-disjoint augmenting paths w.r.t. $\mathcal{M}$.*

We start with the following simple observation.

OBSERVATION 2. *Since the graph before the update excludes length-ℓ augmenting paths w.r.t. $\mathcal{M}$, any length-ℓ augmenting path w.r.t. $\mathcal{M}$ in the updated graph must traverse either u or v, or both of them.*

LEMMA 3.1. *After augmenting $\mathcal{M}$ along the paths Q_1 and Q_2, no length-ℓ augmenting path exists.*

Proof. The proof splits into two cases, according to the execution of the procedure.

Case 1: $P_u \neq \perp$. In this case P_u is a shortest (non-empty) length-ℓ augmenting path w.r.t. $\mathcal{M}$ traversing u but not v, and the procedure computes a shortest length-ℓ augmenting path P'_v w.r.t. $\mathcal{M}'_u$ traversing v, terminating with $Q_1 = P_u, Q_2 = P'_v$.

We start with making the following claim.

CLAIM 2. *Any length-ℓ augmenting path w.r.t. $\mathcal{M}'_u = \mathcal{M} \oplus P_u$ must traverse v.*

Proof. Suppose for contradiction the existence of a length-ℓ augmenting path P' w.r.t. $\mathcal{M}'_u$ that does not traverse v, and define $\mathcal{M}' = \mathcal{M} \oplus P_u \oplus P'$. By Fact 3.2, the edges in $\mathcal{M} \oplus \mathcal{M}' = P_u \oplus P'$ contain at least two vertex-disjoint augmenting paths P_1 and P_2 w.r.t. $\mathcal{M}$. Since none of these paths traverse v, at least one of them traverses neither u nor v. Suppose without loss of generality that $|P_1| \leq |P_2|$. Since

$$|P_1| + |P_2| \;\leq\; |P_u \oplus P'| \;\leq\; |P_u| + |P'| \;\leq\; 2\ell,$$

it follows that $|P_1| \leq \ell$. Thus P_1 is a length-ℓ augmenting path w.r.t. $\mathcal{M}$ that does not traverse v. Observation 2 implies that P_1 must traverse u. The first consequence of this fact is that P_2 traverses neither u nor v. Also, by definition $|P_u| \leq |P_1|$, hence

$$|P_u| + |P_2| \;\leq\; |P_1| + |P_2| \;\leq\; |P_u| + |P'|.$$

It follows that $|P_2| \leq |P'| \leq \ell$. Summarizing, we have shown that P_2 is a length-ℓ augmenting path w.r.t. $\mathcal{M}$ that traverses neither u nor v, contradicting Observation 2. ∎

Having proved Claim 2, we next argue that there is no length-ℓ augmenting path w.r.t. $\mathcal{M}''_u = \mathcal{M}'_u \oplus P'_v = \mathcal{M} \oplus P_u \oplus P'_v$, proving the lemma in this case. (We assume that $P'_v \neq \perp$, otherwise this assertion follows immediately from Claim 2.) Suppose for contradiction the existence of a length-ℓ augmenting path P' w.r.t. $\mathcal{M}''_u$, and define $\mathcal{M}' = \mathcal{M}''_u \oplus P' = \mathcal{M}'_u \oplus P'_v \oplus P'$. By Fact 3.2, the edges in $\mathcal{M}'_u \oplus \mathcal{M}' = P'_v \oplus P'$ contain at least two vertex-disjoint augmenting paths P_1 and P_2 w.r.t. $\mathcal{M}'_u$. By Claim 2, P'_v is a shortest augmenting path w.r.t. $\mathcal{M}'_u$, implying that the length of both paths P_1 and P_2 is no smaller than that of P'_v. Thus, as in the proof of Claim 2, it can be shown that both these paths have length at most ℓ. However, at least one of these paths does not traverse v, contradicting Claim 2.

Case 2: $P_u = \perp$. In this case there is no length-ℓ augmenting path w.r.t. $\mathcal{M}$ traversing u but not v, and the procedure computes a shortest length-ℓ augmenting path P_v w.r.t. $\mathcal{M}$ traversing v, terminating with $Q_1 = \perp, Q_2 = P_v$. We next argue that there is no length-ℓ augmenting path w.r.t. $\mathcal{M}'_v = \mathcal{M} \oplus P_v$, proving the lemma in this case. (We assume that $P_v \neq \perp$, otherwise

this assertion follows immediately from Observation 2.) Suppose for contradiction the existence of a length-ℓ augmenting path P' w.r.t. $\mathcal{M}'_v$, and define $\mathcal{M}' = \mathcal{M}'_v \oplus P' = \mathcal{M} \oplus P_v \oplus P'$. By Fact 3.2, the edges in $\mathcal{M} \oplus \mathcal{M}' = P_v \oplus P'$ contain at least two vertex-disjoint augmenting paths P_1 and P_2 w.r.t. $\mathcal{M}$. By Observation 2, P_v is a shortest augmenting path w.r.t. $\mathcal{M}$, implying that the length of both paths P_1 and P_2 is no smaller than that of P_v. Thus, as in the proof of Claim 2, it can be shown that both these paths have length at most ℓ. By Observation 2, each of these paths must traverse at least one vertex among u and v, which means that exactly one of these paths, without loss of generality P_1, traverses u, and the other path, P_2, traverses v. However, P_1 is a length-ℓ augmenting path w.r.t. $\mathcal{M}$ traversing u but not v, which contradicts the fact that $P_u = \perp$.

Lemma 3.1 follows. ∎

To summarize, we showed that it suffices to augment the matching $\mathcal{M}$ along two augmenting paths Q_1 and Q_2, and presented Procedure `Augment` for computing these paths. While this procedure examines possibly many length-ℓ augmenting paths before terminating with Q_1 and Q_2. all such paths are contained in the ℓ-radius balls centered around u and v. Thus, Procedure `Augment` is inherently local, and can be implemented efficiently in bounded degree graphs and in general distributed networks.

Bounded degree graphs. Assume that the maximum degree in the graph is bounded by Δ. In this case the number of length-ℓ augmenting paths contained in the ℓ-radius balls centered around u and v is bounded by $\Delta^{O(\ell)}$. Hence a brute force process will find the required paths Q_1 and Q_2 in $\Delta^{O(\ell)}$ time. (We do not try to optimize the constant in the exponent.)

Our result for local $(1 + 1/k)$-MCMs in bounded degree graphs is summarized in the following theorem.

THEOREM 3.1. *Assume the maximum degree in the graph is always bounded by Δ. For any $k \geq 1$, length-$(2k-1)$ augmenting paths can be excluded dynamically (yielding $(1 + 1/k)$-MCMs) within a worst-case update time of $\Delta^{O(k)}$.*

Length-3 augmenting paths in bounded degree graphs. We remark that dealing with augmenting paths of length at most 3 is easier, and was known prior to this work. Specifically, length-1 augmenting paths can be easily excluded (yielding maximal matchings) within a worst-case update time of $O(\Delta)$. While length-3 augmenting paths can be easily excluded (yielding 3/2-MCMs) within a worst-case update time of $O(\Delta^2)$, a more careful implementation (see [8]) leads to an improved update time of $O(\Delta)$.

THEOREM 3.2. *[8] Length-3 augmenting paths can be excluded dynamically (yielding 3/2-MCMs) within a worst-case update time of $O(\Delta)$.*

Distributed networks. Since all augmenting paths examined by Procedure `Augment` are contained in the ℓ-radius balls centered around u and v, a simple distributed implementation of this procedure will require $O(\ell)$ communication rounds. We consider the standard $\mathcal{LOCAL}$ model of communication (cf. [33]), which is a standard distributed computing model capturing the essence of spatial locality. In a static setting all processors wake up simultaneously, and computation proceeds in fault-free synchronous rounds during which every processor exchanges messages of unbounded size. In a dynamic setting, however, upon the insertion or deletion of an edge $e = (u, v)$, only the affected vertices (u and v) are woken up. In this setting, a *worst-case update time* bounds the maximum number of communication rounds (in the $\mathcal{LOCAL}$ model) needed for repairing the solution per update operation, over a worst-case sequence of update operations. We assume that the topological changes occur serially and are sufficiently spaced so that the protocol has enough time to complete its operation before the occurrence of the next change; since the worst-case update time of our algorithm is low (depending, of course, on the required approximation guarantee), this assumption should be acceptable in many practical scenarios.

Our result for $(1 + 1/k)$-MCMs in general distributed networks is summarized in the following theorem.

THEOREM 3.3. *For any distributed network and any $k \geq 1$, length-$(2k-1)$ augmenting paths can be excluded dynamically (yielding $(1+1/k)$-MCMs) within a worst-case update time of $O(k)$.*

3.2 From Bounded Degree to Bounded Arboricity Graphs.

Recall the algorithm of Sect. 2 and the corresponding terminology. Denote by V^{low} the vertex set $F^{low} \cup M^{low}$. Let $\mathcal{M}^{low*}$ be an MCM for the graph $G^{low} = G[V^{low}]$ induced by V^{low}, and recall that $\mathcal{M}^*$ is an MCM for the entire graph G.

LEMMA 3.2. *Assuming $\epsilon < 1/2$, we have $|\mathcal{M}^*| \leq (1 + 2\epsilon) \cdot |\mathcal{M}^{low*}|$.*

Proof. By Claim 1, $|M^{low}| \geq \frac{2}{\epsilon} \cdot (|F^{high}| + |M^{high}|)$, so

$$\begin{aligned} |\mathcal{M}^*| &\geq |\mathcal{M}| = |M|/2 \geq |M^{low}|/2 \\ &\geq (|F^{high}| + |M^{high}|)/\epsilon. \end{aligned}$$

Let $\mathcal{M}^{low}$ be the matching in G^{low} obtained from $\mathcal{M}^*$ after removing all edges incident on $F^{high} \cup M^{high}$. Note that

$$\begin{aligned} |\mathcal{M}^{low*}| &\geq |\mathcal{M}^{low}| \geq |\mathcal{M}^*| - (|F^{high}| + |M^{high}|) \\ &\geq (1-\epsilon) \cdot |\mathcal{M}^*|. \end{aligned}$$

It follows that

$$|\mathcal{M}^*| \leq (\frac{1}{1-\epsilon}) \cdot |\mathcal{M}^{low*}| \leq (1+2\epsilon) \cdot |\mathcal{M}^{low*}|,$$

where the last inequality holds for $\epsilon \leq 1/2$. ∎

By Lemma 3.2, a $(1+\epsilon)$-MCM for G^{low} gives a $((1+\epsilon)\cdot(1+2\epsilon))$-MCM for the entire graph G. (By appropriate scaling, we get a $(1+\epsilon)$-MCM for G.) To maintain dynamic $(1+\epsilon)$-MCM in G^{low}, we dynamically exclude length-ℓ augmenting paths in G^{low}, for any ℓ, just as in bounded degree graphs.

Following a single edge update (u, v), length-ℓ augmenting paths may be created. (We assume that prior to this update our matching $\mathcal{M}$ excludes length-ℓ augmenting paths.) We start by running the $O(\alpha/\epsilon)$-time update algorithm of Sect. 2 for maintaining Invariants 1 and 2 following the edge update.

As mentioned, Procedure `Augment` locally computes (in general graphs) at most two paths, such that after augmenting the matching $\mathcal{M}$ along the edges of these paths, no length-ℓ augmenting path may exist. In bounded degree graphs, the running time of this procedure is $D^{O(\ell)}$; see Theorem 3.1. Even though G^{low} is not exactly a bounded degree graph, it is rather close to being one.

Observation 3. *Each vertex of F^{low} has degree at most D. While the degree of vertices of M^{low} may be arbitrarily large, each vertex of M^{low} has at most D matched neighbors.*

By Observation 3, the maximum degree of the subgraph $G[M^{low}]$ induced by M^{low} is at most D. Besides its two free endpoints, any augmenting path in G^{low} is contained entirely in $G[M^{low}]$. Moreover, each vertex of $G[M^{low}]$ can maintain all the required information regarding its (possibly many) free neighbors in G^{low}. In particular, each such vertex can dynamically maintain a partition of its neighborhood $\Gamma(v)$ into two subsets $\Gamma'(v)$ and $\Gamma''(v)$, where $\Gamma'(v)$ contains all neighbors of v that belong to F^{low} and $\Gamma''(v) = \Gamma(v) \setminus \Gamma'(v)$. (Indeed, the degree of each vertex of F^{low} in the entire graph G is bounded by D. Thus when such a vertex either enters F^{low} or leaves it, it can notify all its neighbors about that within $O(D)$ time. Each of these neighbors can, in turn, update its own neighborhood partition in constant time.) Consequently, in terms of computing augmenting paths, the graph G^{low} can be viewed as having maximum degree D. In particular, running Procedure `Augment` on the graph G^{low} will take $D^{O(\ell)}$ time.

However, maintaining the graph G^{low} *explicitly* is problematic: Following a single edge update, a vertex of F^{high} with many neighbors in F^{low} might move to M^{low} (alternatively, a vertex of M^{low} might move to F^{high}), hence all the many edges connecting this vertex with vertices in F^{low} should be added to (or removed from) G^{low}. Note, however, that there is no need to maintain the graph G^{low} explicitly. All we need to do is to efficiently compute augmenting paths restricted to this graph, and this can be done "on the fly": When traversing any vertex v along such a path, we simply scan its neighbors in $\Gamma''(v)$. If there are at least D such neighbors, then we backtrack, as v does not belong to G^{low}. Otherwise v belongs to G^{low}, and we can either continue the path via the (less than D) edges to v's matched neighbors, or complete the path via a single edge (out of possibly many) to a free neighbor.

Recall that $D = 8\alpha/\epsilon$, where α is the arboricity bound and ϵ is some small parameter (see Lemma 3.2). To optimize the approximation factor, we set $\ell = \Theta(1/\epsilon)$; also, we can apply Theorem 3.2 instead of Theorem 3.1 (when $\ell = 3$), thus reducing the update time to $O(\alpha/\epsilon)$ at the expense of increasing the approximation guarantee to $(3/2)(1+2\epsilon) = 3/2 + O(\epsilon)$. This completes the reduction from bounded arboricity graphs (with arboricity bound α) to bounded degree graphs (with degree bound $D = O(\alpha/\epsilon)$).

Our results for local $(1+\epsilon)$-MCMs and $(3/2+\epsilon)$-MCMs in bounded arboricity graphs are summarized in the following theorem.

Theorem 3.4. *For any graph with arboricity bounded by α and any $\epsilon > 0$, there is a local update procedure for dynamically maintaining $(1+\epsilon)$-MCMs (respectively, $(3/2+\epsilon)$-MCMs) within a worst-case update time of $(\alpha/\epsilon)^{O(1/\epsilon)}$ (resp., $O(\alpha/\epsilon)$).*

In distributed networks, one can measure the maximum number of messages sent following a single edge update, over a worst-case sequence of update operations, hereafter *update message complexity*. Theorems 3.3 and 3.4 imply the following corollary, which summarizes our results for $(1+\epsilon)$-MCMs and $(3/2+\epsilon)$-MCMs in distributed networks of bounded arboricity.

Corollary 3.1. *For any distributed network of arboricity bounded by α and any $\epsilon > 0$, one can dynamically maintain $(1+\epsilon)$-MCMs (resp. $(3/2+\epsilon)$-MCMs) with worst-case update time $O(1/\epsilon)$ and update message complexity $(\alpha/\epsilon)^{O(1/\epsilon)}$ (resp., $O(\alpha/\epsilon)$).*

4 Dynamic $(1+\epsilon)$-MCMs: The Lazy (and Global) Approach

Unsatisfied with the bound of the previous section, here we obtain an improved update time of $O(\alpha \cdot \epsilon^{-2})$.

4.1 The Lazy Approach.

Our algorithm builds on the scheme of Gupta and Peng [16], which provides a dynamic $(1+\epsilon)$-MCM with update time $O(\sqrt{m}\cdot\epsilon^{-2})$ for general m-edge graphs. The scheme of [16] exploits the following *stability* property of matchings: The size of the MCM $\mathcal{M}^* = \mathcal{M}_i^*$ changes by at most 1 following each update. This means that if we have a *large* matching of size close to the maximum, it will remain close to it throughout a long sequence of updates. Formally we have the following lemma.

LEMMA 4.1. (**Lemma 3.1 in [16]**) *Let $\epsilon, \epsilon' \leq 1/2$. Suppose that $\mathcal{M}_i$ is a $(1+\epsilon)$-MCM for G_i. For $j = i, i+1, \ldots, i + \lfloor \epsilon' \cdot |\mathcal{M}_i| \rfloor$, let $\mathcal{M}_i^{(j)}$ denote the matching $\mathcal{M}_i$ after removing from it all edges that got deleted during the updates $i+1, \ldots, j$. Then $\mathcal{M}_i^{(j)}$ is a $(1+2\epsilon+2\epsilon')$-MCM for the graph G_j.*

This motivates applying the following lazy approach to dynamic matching. Compute a $(1+\epsilon/4)$-MCM $\mathcal{M}_i$ at a certain update i (initially $i=1$), and use the same matching $\mathcal{M}_i^{(j)}$ throughout all updates $j = i, i+1, \ldots, i' = i + \lfloor \epsilon/4 \cdot |\mathcal{M}_i| \rfloor$. (By Lemma 4.1, $\mathcal{M}_i^{(j)}$ is a $(1+\epsilon)$-MCM for all graphs G_j.) Next compute a fresh $(1+\epsilon/4)$-MCM $\mathcal{M}_{i'}$ following update i' and use it throughout all updates $i', i'+1, \ldots, i' + \lfloor \epsilon/4 \cdot |\mathcal{M}_{i'}| \rfloor$, and repeat. In this way the static time complexity of computing a $(1+\epsilon)$-MCM $\mathcal{M}$, which by Fact 1.1 is $O(m/\epsilon)$ for m-edge graphs, is amortized over $1 + \lfloor \epsilon/4 \cdot |\mathcal{M}| \rfloor = \Omega(\epsilon \cdot |\mathcal{M}|)$ updates. Consequently, the amortized time is $O(m/(|\mathcal{M}|\epsilon^2))$. There are now two cases, depending on the size of the matching $\mathcal{M}$.
Case 1: $|\mathcal{M}| = \Omega(\sqrt{m})$. In this case the amortized time is $O(\sqrt{m}\cdot\epsilon^{-2})$, as required.
Case 2: $|\mathcal{M}| = o(\sqrt{m})$. In this case the amortized time is $\omega(\sqrt{m}\cdot\epsilon^{-2})$, which is too costly. The key insight behind the scheme of [16] is that one can compute in time $O(|\mathcal{M}|^2)$ a *matching sparsifier* of G, i.e., a sparse (of size $O(|\mathcal{M}|^2)$) subgraph G' of G for which the MCM is of the same size. With such a sparsifier G' at hand, a $(1+\epsilon)$-MCM for the entire graph G can be computed in $O(|\mathcal{M}|^2/\epsilon)$ time by Fact 1.1, so the amortized time is reduced to

$$O((|\mathcal{M}|^2/\epsilon)/(\epsilon\cdot|\mathcal{M}|)) \;=\; O(|\mathcal{M}|\cdot\epsilon^{-2}) \;=\; o(\sqrt{m}\cdot\epsilon^{-2}).$$

Let VC be an $O(1)$-MCVC for G. The matching sparsifier $G' = (V', E')$ of [16] is a subgraph of G containing (1) all edges in the subgraph $G[VC]$ induced by VC, and (2) for each vertex $v \in VC$, (up to) $|VC|+1$ edges connecting it with arbitrary neighbors outside VC. (See Figure 1 for an illustration.) Observe that

$$|E'| \;\leq\; \binom{|VC|}{2} + |VC|\cdot(|VC|+1) \;=\; O(|VC|^2).$$

Recall that MCVC and MCM of any graph are of the same size (up to a factor of 2). Since VC is an $O(1)$-MCVC for G and $\mathcal{M}$ is an $O(1)$-MCM for it, it follows that $|E'| = O(|VC|^2) = O(|\mathcal{M}|^2)$. It is left to show that G' is indeed a matching sparsifier for G.

LEMMA 4.2. (**Lemma 3.3 in [16]**) *An MCM for G' is also an MCM for G.*

For this scheme to work, one needs to dynamically maintain an $O(1)$-MCVC with update time $O(\sqrt{m}\cdot\epsilon^{-2})$. As shown in [31], a worst-case update time of $O(\sqrt{m})$ suffices for maintaining a 2-MCVC.

Extreme cases. If $|\mathcal{M}|\cdot\epsilon/4 < 1$, then the matching $\mathcal{M}$ is used only for a single update. However, in this case the matching sparsifier is of size at most $O(\min\{m, |\mathcal{M}|^2\}) = O(\min\{m, \epsilon^{-2}\})$, and computing a $(1+\epsilon)$-MCM over the sparsifier from scratch requires only $O(\min\{m,\epsilon^{-2}\}/\epsilon) = O(\sqrt{m}\cdot\epsilon^{-2})$ time.

The number of updates following the last $(1+\epsilon)$-MCM computation may be smaller than $|\mathcal{M}|\cdot\epsilon/4$. However, the cost of this last computation can be amortized over all updates in the entire sequence.

Worst-case bounds. This scheme computes a $(1+\epsilon/4)$-MCM $\mathcal{M}$ *from scratch*, and then re-uses it throughout $\lfloor \epsilon/4 \cdot |\mathcal{M}| \rfloor$ additional updates. Even though these computations do not occur often, the worst-case update time may be as high as $O(\min\{m, |\mathcal{M}|^2\}/\epsilon)$. To improve the worst-case guarantee, a standard trick used in [16] is to simulate a static approximate-MCM computation within a "sliding window" of $1 + \lfloor \epsilon/4\cdot|\mathcal{M}| \rfloor$ consecutive updates, so that following each update the algorithm simulates only $O(\min\{m, |\mathcal{M}|^2\}/\epsilon)/(1+\lfloor\epsilon/4\cdot|\mathcal{M}|\rfloor) = O(\sqrt{m}\cdot\epsilon^{-2})$ steps of the static computation. During this sliding window the gradually-computed matching is useless, so the previously-computed matching is re-used. This means that each matching is re-used throughout a time window of twice as many updates, hence the approximation guarantee increases from $1+\epsilon$ to $1+2\epsilon$. (Of course, the gradually-computed matching does not include edges that are added to or deleted from the graph during the sliding window.)

The extra worst-case cost $O(\sqrt{m})$ of dynamically maintaining an $O(1)$-MCVC is negligible.

4.2 Density-Sensitive Adjustments.

To dynamically maintain $(1+\epsilon)$-MCMs with a worst-case update

time of $O(\alpha \cdot \epsilon^{-2})$, where α is an upper bound on the graph's arboricity, we make two critical adjustments to the above scheme. The first adjustment is to plug our own density-sensitive (dynamic) $(2+\epsilon)$-MCVC (given by Corollary 2.1) instead of the ordinary 2-MCVC of [31]. As a result, the cost of dynamically maintaining an $O(1)$-MCVC in the above scheme is reduced from $O(\sqrt{m})$ to $O(\alpha/\epsilon)$.

Hereafter, let $\widetilde{VC} = \widetilde{VC}_i$ denote our density-sensitive $O(1)$-MCVC for $G = G_i$. Also, let $\tilde{M} = \tilde{M}_i$ denote the approximate matching for G. The second adjustment is in computing a density-sensitive matching sparsifier $\tilde{G}$. Specifically, our sparsifier $\tilde{G} = (\tilde{V}, \tilde{E})$ is a subgraph of G containing (1) all edges in the subgraph $G[\widetilde{VC}]$ induced by $\widetilde{VC}$, and (2) for each vertex $v \in \widetilde{VC}$, (up to) $D = 8\alpha/\epsilon$ (rather than $|\widetilde{VC}|+1$) edges connecting it with arbitrary neighbors outside $\widetilde{VC}$. Let $\tilde{\Gamma} = \tilde{V} \setminus \widetilde{VC}$ be the set of vertices of $\tilde{G}$ outside $\widetilde{VC}$, and let V_{out} denote the set of remaining vertices of G outside $\tilde{G}$. (See Figure 1 for an illustration.)

We first claim that our matching sparsifier $\tilde{G}$ is of small size.

LEMMA 4.3. $|\tilde{E}| = O(|\tilde{\mathcal{M}}| \cdot \alpha/\epsilon)$.

Proof. Since the arboricity of the subgraph $G[\widetilde{VC}]$ of G induced by $\widetilde{VC}$ is bounded by α, this subgraph contains at most $|\widetilde{VC}| \cdot \alpha$ edges. All these edges belong to $\tilde{E}$. In addition to these edges, each vertex of $\widetilde{VC}$ is incident in $\tilde{G}$ to at most D vertices outside $\widetilde{VC}$. Thus

$$|\tilde{E}| \;\le\; |\widetilde{VC}| \cdot \alpha + |\widetilde{VC}| \cdot D \;=\; O(|\widetilde{VC}| \cdot \alpha/\epsilon).$$

Recall that the MCVC and the MCM are of the same size (up to a factor of 2). Since $\widetilde{VC}$ is an $O(1)$-MCVC for G and $\tilde{\mathcal{M}}$ is an $O(1)$-MCM for it, it follows that $|\tilde{E}| = O(|\widetilde{VC}| \cdot \alpha/\epsilon) = O(|\tilde{\mathcal{M}}| \cdot \alpha/\epsilon)$. ∎

The time needed to build the matching sparsifier $\tilde{G}$ is proportional to its size. By Fact 1.1, a $(1+\epsilon)$-MCM for $\tilde{G}$ can be computed in time $O(|\tilde{\mathcal{M}}| \cdot \alpha \cdot \epsilon^{-2})$. Plugging this instead of the bound $O(\min\{m, |\tilde{\mathcal{M}}|^2\}/\epsilon)$ of [16] reduces the update time from $O(\sqrt{m} \cdot \epsilon^{-2})$ to $O(\alpha \cdot \epsilon^{-3})$. In Sect. 4.3 we shave a factor of $1/\epsilon$ from this bound, which results in the required update time of $O(\alpha \cdot \epsilon^{-2})$.

While there exist (infinitely many) n-vertex graphs for which the arboricity is smaller than the size of an MCVC by a factor of $\Omega(n)$, the arboricity is always upper bounded by the MCVC's size. (Refer to Appendix A for more details.) This means that our density-sensitive matching sparsifier $\tilde{G}$ is always sparser (up to a factor of $8/\epsilon$) than that of [16]. The challenge is then to show that our (possibly much sparser) subgraph $\tilde{G}$ is indeed a matching sparsifier. Even though we can no longer claim that MCMs for $\tilde{G}$ and for G are of precisely the same size (as in Lemma 4.2), we next show that they are of almost the same size, namely, we may lose at most a negligible factor of $1+\epsilon$.

LEMMA 4.4. *Let $\mathcal{M}^*$ and $\tilde{\mathcal{M}}^*$ be MCMs for the graphs G and $\tilde{G}$, respectively. Then $|\mathcal{M}^*| \le (1+\epsilon) \cdot |\tilde{\mathcal{M}}^*|$.*

Proof. To prove the lemma, we construct a matching $\tilde{\mathcal{M}}$ for $\tilde{G}$ satisfying $|\mathcal{M}^*| \le (1+\epsilon) \cdot |\tilde{\mathcal{M}}|$.

We start by partitioning the edges of $\mathcal{M}^*$ into two subsets $\mathcal{M}_1^*$ and $\mathcal{M}_2^*$, where $\mathcal{M}_1^*$ contains all the edges of $\mathcal{M}^*$ that belong to the graph $\tilde{G}$, and $\mathcal{M}_2^* = \mathcal{M}^* \setminus \mathcal{M}_1^*$.

We proceed by initiating $\tilde{\mathcal{M}}$ to $\mathcal{M}_1^*$. Next, we show that at least $|\mathcal{M}_2^*| - \epsilon/2 \cdot |\mathcal{M}_1^*|$ additional edges of $\tilde{G}$ can be added to $\tilde{\mathcal{M}}$ while keeping it a valid matching. Clearly, all these additional edges must be vertex-disjoint from all the edges of $\mathcal{M}_1^*$ (which already belong to $\tilde{\mathcal{M}}$).

Consider an arbitrary edge $e = (u, v) \in \mathcal{M}_2^*$. Since $\widetilde{VC}$ is a vertex cover of G, at least one endpoint of e, without loss of generality u, must be in $\widetilde{VC}$. By definition, e does not belong to the graph $\tilde{G}$; since all edges internal to $\widetilde{VC}$ belong to $\tilde{G}$, the other endpoint v of e must be outside $\widetilde{VC}$. Moreover, the vertex $u \in \widetilde{VC}$ must have more than D neighbors outside $\widetilde{VC}$ in G, otherwise the edge e would belong to $\tilde{G}$.

Let $U = \{u \mid (u, v) \in \mathcal{M}_2^*, u \in \widetilde{VC}, v \notin \widetilde{VC}\}$ be the set of endpoints in $\widetilde{VC}$ of all edges $e \in \mathcal{M}_2^*$. Notice that $|U| = |\mathcal{M}_2^*|$. Since $\mathcal{M}^* = \mathcal{M}_1^* \cup \mathcal{M}_2^*$ is a valid matching, no vertex of U is incident on any edge of $\mathcal{M}_1^*$. Recall that each vertex of U has more than D neighbors outside $\widetilde{VC}$ in the original graph G, and note that precisely D of them remain its neighbors in the sparsifier $\tilde{G}$. Let us partition the vertices of $\tilde{\Gamma}$ into two sets, the set $\tilde{\Gamma}_{matched}$ of vertices that are matched by edges of $\mathcal{M}_1^*$, and the set $\tilde{\Gamma}_{free}$ of vertices that are free w.r.t. $\mathcal{M}_1^*$. A vertex $u \in U$ is called *risky* (respectively, *safe*) if at least $D/2$ of its neighbors (w.r.t. $\tilde{G}$) in $\tilde{\Gamma}$ are in $\tilde{\Gamma}_{matched}$ (resp., $\tilde{\Gamma}_{free}$). We correspondingly partition U into two subsets U_{risky} and U_{safe}, where U_{risky} (respectively, U_{safe}) contains all risky (resp., safe) vertices of U. (See Figure 1 for an illustration.)

CLAIM 3. $|U_{risky}| \le \epsilon/2 \cdot |\mathcal{M}_1^*|$.

Proof. For each vertex $u \in U_{risky}$, let $\Gamma_{matched}(u)$ be the set of its (at least $D/2$) neighbors in $\tilde{\Gamma}_{matched}$. Let Γ_{risky} denote the union of the sets $\Gamma_{matched}(u)$ over all vertices $u \in U_{risky}$, i.e., $\Gamma_{risky} = \bigcup_{u \in U_{risky}} \Gamma_{matched}(u)$. Since $\widetilde{VC}$ is a vertex cover and $\widetilde{VC} \cap \Gamma_{risky} = \emptyset$, any edge of $\mathcal{M}_1^*$ is incident on at most one vertex of

Γ_{risky}, hence $|\Gamma_{risky}| \leq |\mathcal{M}_1^*|$. Consider the subgraph $G_{risky} = (V_{risky}, E_{risky})$ of $\tilde{G}$ induced by the vertex set $V_{risky} = U_{risky} \cup \Gamma_{risky}$. Since each vertex in U_{risky} has at least $D/2 = 4\alpha/\epsilon$ neighbors in Γ_{risky}, it follows that $|E_{risky}| \geq |U_{risky}| \cdot 4\alpha/\epsilon$. As the arboricity of G_{risky} is bounded by α, we have

$$\begin{aligned}\alpha &\geq \left\lceil \frac{|E_{risky}|}{|V_{risky}|-1} \right\rceil \geq \frac{|E_{risky}|}{|V_{risky}|-1} \\ &\geq \frac{|U_{risky}| \cdot 4\alpha/\epsilon}{|U_{risky}| + |\Gamma_{risky}| - 1}.\end{aligned}$$

Hence

$$\begin{aligned}|U_{risky}| \cdot 4/\epsilon &\leq |U_{risky}| + |\Gamma_{risky}| - 1 \\ &\leq |U_{risky}| + |\mathcal{M}_1^*|,\end{aligned}$$

and so

$$|U_{risky}| \leq |\mathcal{M}_1^*|/(4/\epsilon - 1) \leq \epsilon/2 \cdot |\mathcal{M}_1^*|.$$

(The last inequality holds for $\epsilon \leq 2$.) This completes the proof of Claim 3. ∎

Consider an edge in $\tilde{G}$ between an arbitrary vertex in U_{safe} and a vertex in $\tilde{\Gamma}_{free}$. Obviously, such an edge is vertex-disjoint to all edges of $\mathcal{M}_1^*$. The following claim thus shows that at least $|U_{safe}|$ edges of $\tilde{G}$ can be added to the matching $\tilde{\mathcal{M}}$ while preserving its validity. To prove Claim 4, we use Hall's marriage theorem.

THEOREM 4.1. (HALL'S MARRIAGE THEOREM [18]) *Let G be a bipartite graph with sides X and Y. There is a matching that entirely covers X if and only if for every subset W of X, $|W| \leq |\Gamma_G(W)|$, where $\Gamma_G(W) = \bigcup_{v \in W} \Gamma_G(v)$ is the* neighborhood *of W.*

CLAIM 4. *There is a matching $\mathcal{M}_{safe}$ in $\tilde{G}$ that entirely covers U_{safe} by edges between U_{safe} and $\tilde{\Gamma}_{free}$.*

Proof. For each vertex $u \in U_{safe}$, let $\Gamma_{free}(u)$ be the set of its (at least $D/2$) neighbors in $\tilde{\Gamma}_{free}$. Let Γ_{safe} denote the union of the sets $\Gamma_{free}(u)$ over all vertices $u \in U_{safe}$, i.e., $\Gamma_{safe} = \bigcup_{u \in U_{safe}} \Gamma_{free}(u)$. Consider the induced bipartite subgraph G_{safe} of $\tilde{G}$ with sides U_{safe} and Γ_{safe}. We argue that for any subset $W \subseteq U_{safe}$, its neighborhood $\Gamma_{G_{safe}}(W) = \bigcup_{v \in W} \Gamma_{G_{safe}}(v)$ in G_{safe} is of larger size. Indeed, each vertex of W has at least $D/2 = 4\alpha/\epsilon$ neighbors in $\Gamma_{G_{safe}}(W)$. Since the arboricity of the subgraph of G_{safe} induced by the vertex set $W \cup \Gamma_{G_{safe}}(W)$ is bounded by α, it follows that

$$\alpha \geq \frac{|W| \cdot 4\alpha/\epsilon}{|W| + |\Gamma_{G_{safe}}(W)| - 1},$$

hence

$$|\Gamma_{G_{safe}}(W)| \geq |W| \cdot (4/\epsilon - 1) > |W|.$$

Therefore, by Hall's marriage theorem, there exists a matching $\mathcal{M}_{safe}$ in G_{safe} that entirely covers U_{safe}. Claim 4 follows. ∎

Note that the matching $\mathcal{M}_{safe}$ guaranteed by Claim 4 satisfies $|\mathcal{M}_{safe}| = |U_{safe}|$. We add all edges in $\mathcal{M}_{safe}$ to the matching $\tilde{\mathcal{M}}$, so that $\tilde{\mathcal{M}} = \mathcal{M}_1^* \cup \mathcal{M}_{safe}$. Combined with Claim 3, it follows that

$$\begin{aligned}|\mathcal{M}_2^*| &= |U| = |U_{risky}| + |U_{safe}| \\ &\leq \epsilon/2 \cdot |\mathcal{M}_1^*| + |\mathcal{M}_{safe}|,\end{aligned}$$

implying that

$$\begin{aligned}|\tilde{\mathcal{M}}| &= |\mathcal{M}_1^*| + |\mathcal{M}_{safe}| \geq |\mathcal{M}_1^*| + |\mathcal{M}_2^*| - \epsilon/2 \cdot |\mathcal{M}_1^*| \\ &\geq (1-\epsilon/2) \cdot (|\mathcal{M}_1^*| + |\mathcal{M}_2^*|) = (1-\epsilon/2) \cdot |\mathcal{M}^*|.\end{aligned}$$

Hence

$$\begin{aligned}|\mathcal{M}^*| &\leq \frac{1}{1-\epsilon/2} \cdot |\tilde{\mathcal{M}}| \leq (1+\epsilon) \cdot |\tilde{\mathcal{M}}| \\ &\leq (1+\epsilon) \cdot |\tilde{\mathcal{M}}^*|.\end{aligned}$$

(The second inequality holds for $\epsilon \leq 1$.) Lemma 4.4 follows. ∎

4.3 A Small Improvement in the Update Time.

In this section we show that the time needed for computing a $(1+\epsilon)$-MCM over the matching sparsifier $\tilde{G}$ can be improved by a factor of $1/\epsilon$. As a result, the update time of our dynamic $(1+\epsilon)$-MCM algorithm is reduced to $O(\alpha \cdot \epsilon^{-2})$.

Recall that our sparsifier $\tilde{G} = (\tilde{V}, \tilde{E})$ is a subgraph of G containing (1) all edges in the subgraph $G[\widetilde{VC}]$ induced by the vertex cover $\widetilde{VC}$, and (2) for each vertex $v \in \widetilde{VC}$, (up to) $D = 8\alpha/\epsilon$ edges connecting it with arbitrary neighbors outside $\widetilde{VC}$. As shown in Lemma 4.3, $|\tilde{E}| = O(|\tilde{\mathcal{M}}| \cdot \alpha/\epsilon)$. Running the static $(1+\epsilon)$-MCM algorithms [20, 30, 38] on the matching sparsifier $\tilde{G}$ *directly*, as done in Sect. 4.2, is too costly; by Fact 1.1, it requires $O(|\tilde{E}|/\epsilon) = (|\tilde{\mathcal{M}}| \cdot \alpha \cdot \epsilon^{-2})$ time. Our goal is to compute a $(1+\epsilon)$-MCM for the sparsifier $\tilde{G}$ in time $O(|\tilde{\mathcal{M}}| \cdot \alpha/\epsilon)$, which is *linear* in (our upper bound on) the size of the sparsifier.

The static $(1+\epsilon)$-MCM algorithms of [20, 30, 38] are, in fact, algorithms for excluding length-$(2k-1)$ augmenting paths, for any $k \geq 1$. (Indeed, by Fact 3.1, setting $\epsilon = 1/k$ yields a $(1+\epsilon)$-MCM.) These algorithms operate in phases. Starting with an empty matching

$\mathcal{M}_0 = \emptyset$, in each phase $i, i = 1, 2, \ldots, k$, the previous matching $\mathcal{M}_{i-1}$ is augmented via a maximal (possibly empty) set $\mathcal{P}_i$ of length-$(2i-1)$ augmenting paths to obtain matching $\mathcal{M}_i$. It is well known [20] that at the end of the ith phase, there are no length-$(2i-1)$ augmenting paths w.r.t. resulting matching $\mathcal{M}_i$, and so the matching $\mathcal{M}_k$ at the end of the k phases excludes length-$(2k-1)$ augmenting paths. The time needed to compute such a set of paths $\mathcal{P}_i$ determines the runtime of a single phase; while this time is known to be at most linear in the total number of edges in the graph, this is not good enough for our needs. Specifically, the runtime in each phase would be $O(|\tilde{E}) = O(|\tilde{\mathcal{M}}| \cdot \alpha/\epsilon)$, yielding a total runtime of $O(k \cdot |\tilde{\mathcal{M}}| \cdot \alpha/\epsilon) = O(|\tilde{\mathcal{M}}| \cdot \alpha \cdot \epsilon^{-2})$.

Consider an arbitrary phase $i, 1 \leq i \leq k$. In this phase a maximal set $\mathcal{P}_i$ of shortest augmenting paths w.r.t. $\mathcal{M}_{i-1}$ is computed, and then the matching $\mathcal{M}_{i-1}$ is augmented via $\mathcal{P}_i$ to obtain $\mathcal{M}_i$. Let $M_{i-1} = V(\mathcal{M}_{i-1})$ and $M_i = V(\mathcal{M}_i)$ be the set of matched vertices w.r.t. $\mathcal{M}_{i-1}$ and $\mathcal{M}_i$, respectively. Observe that $M_{i-1} \subseteq M_i$; indeed, once a vertex becomes matched, it will remain so throughout all subsequent augmentations. Recall that $\widetilde{VC}$ is a vertex cover for the entire graph, hence $|M_{i-1}| \leq |M_i| = O(|\widetilde{VC}|)$. Since the subgraph $G[V_i]$ induced by the vertex set $V_i := M_i \cup \widetilde{VC}$ has arboricity at most α, it contains at most $O(|\widetilde{VC}| \cdot \alpha)$ edges.

We find it instructive to start with the case that there are no edges with both endpoints in $\widetilde{VC}$, rendering the sparsifier $\tilde{G}$ bipartite. In this particular case, any augmenting path P must start at a free vertex of $\widetilde{VC}$ (though it may end at a free vertex outside $\widetilde{VC}$). Observe that among all edges that are examined throughout the computation of path P, its last edge is the only one leading to a free vertex; indeed, an augmenting path is found once we reach a free vertex. After augmenting the edges along P, the last vertex along P (which was free) will become matched, and thus belong to M_i. Since the first vertex along any such path P belongs to $\widetilde{VC}$ and all other vertices belong to M_i, all edges examined throughout the computation of P are contained in the graph $G[V_i]$ induced by the vertex set $V_i = M_i \cup \widetilde{VC}$. This argument shows that the entire computation of the path set $\mathcal{P}_i$ is restricted to the graph $G[V_i]$. As the runtime of this computation is known to be linear in the size of the graph, we conclude that the runtime of phase i is upper bounded by $O(|\widetilde{VC}| \cdot \alpha)$, and the overall runtime will be bounded by $O(k \cdot |\widetilde{VC}| \cdot \alpha) = O(|\tilde{\mathcal{M}}| \cdot \alpha/\epsilon)$.

In the general case, both endpoints of any path $P \in \mathcal{P}_i$ may be outside $\widetilde{VC}$. Therefore, we cannot restrict our attention to paths that start at free vertices of $\widetilde{VC}$. On the other hand, we cannot afford to examine all the edges that are incident to *all* free vertices, as some free vertices may not start any augmenting path, in which case their incident edges may not belong to the graph $G[V_i]$. The key observation is that we can maintain a complete information about the free neighbors of each matched vertex within a total runtime that is linear in the size of the sparsifier $\tilde{G}$. Initially (before the first phase starts) all neighbors of every vertex are free. Once a free neighbor becomes matched, it notifies all its neighbors about its new status. Note, however, that a vertex that becomes matched will remain so throughout the entire computation of all phases, meaning that such an update is done only once per vertex. Consequently, the total runtime of all updates (over all vertices and throughout all phases) is at most linear in the size of the sparsifier $\tilde{G}$, as required. With this information at hand, we can speed up the computation by starting it at matched vertices that are guaranteed to have free neighbors; the rest of the computation remains essentially unchanged. Once we find an alternating path that starts at a matched vertex u_{match} with a free neighbor and ends at a free vertex v_{free}, we concatenate to it an arbitrary edge that connects u_{match} to a free vertex $u_{free}, u_{free} \neq v_{free}$, which yields an augmenting path. (If the only free neighbor of u_{match} is v_{free}, then we continue the exploration.) Similarly, once we find an alternating path that starts at a matched vertex u_{match} with a free neighbor and ends at another matched vertex $v_{match}, v_{match} \neq u_{match}$ with a free neighbor, we concatenate to it an arbitrary edge that connects u_{match} to a free vertex u_{free} and an arbitrary edge that connects v_{match} to a free vertex $v_{free}, v_{free} \neq u_{free}$, which yields an augmenting path. (If the only free neighbor of u_{match} and v_{match} is the same vertex, then we continue the exploration.) After augmenting the edges along this path, the two endpoints u_{free} and v_{free} will become matched, thus we notify the neighbors of u_{free} and v_{free} about their new status, and continue to exploring new augmenting paths as before. This argument shows that, disregarding the time of all status updates, the entire computation of the path set $\mathcal{P}_i$ is restricted to the graph $G[V_i]$. Consequently, the computation time in the general case is the same (up to a constant factor) as in the particular case where there are no edges with both endpoints in $\widetilde{VC}$, which we already showed to be bounded by $O(|\tilde{\mathcal{M}}| \cdot \alpha/\epsilon)$. The update time of our dynamic $(1+\epsilon)$-MCM algorithm is reduced in this way to $O(\alpha \cdot \epsilon^{-2})$, as required.

Our result for (global) $(1+\epsilon)$-MCMs in bounded arboricity graphs is summarized in the following theorem.

THEOREM 4.2. *For any graph with arboricity bounded*

by α and any $\epsilon > 0$, $(1+\epsilon)$-MCMs can be maintained within a worst-case update time of $O(\alpha \cdot \epsilon^{-2})$.

5 Dynamic arboricity

In this section we describe how to adapt some of our algorithms to cope with dynamic arboricity. The basic idea is simple, and follows ideas that were presented in detail in the previous sections. We therefore aim for conciseness.

We assume that the dynamically changing arboricity bound α is known. Since the arboricity of any m-edge graph is bounded by $\sqrt{m}$ and the value of m can be maintained in the obvious way, we can handle general graphs under this assumption, by substituting α with $\sqrt{m}$ in the update time bound. We remark that some of our algorithms can be adapted to the case where the dynamically changing arboricity bound α is unknown. However, this adaptation is intricate, and lies outside the scope of the current paper.

5.1 Dynamic $(1+\epsilon)$-MCMs. We start with the $(1+\epsilon)$-MCM algorithm from Sect. 4, which builds on the lazy scheme of [16].

Consider an arbitrary sliding window that consists of $t := \Theta(\epsilon \cdot |\mathcal{M}|)$ consecutive edge updates. Suppose that this sliding window starts at update step i (with $\mathcal{M} = \mathcal{M}_i$), let $G_i = (V, E_i)$ be the graph at this step with an arboricity bound of α_i, and let $\tilde{G}_i = (V, \tilde{E}_i)$ be the corresponding matching sparsifier. As shown in Sect. 4.3, the time needed to compute a $(1+\epsilon)$-MCM $\mathcal{M}_i$ over the matching sparsifier $\tilde{G}_i$ is $O(|\tilde{E}_i|)$, and this time should be amortized over the t updates in the sliding window. This means that each such edge update should "take care" of $O(|\tilde{E}_i|)/t$ steps of the static computation. More specifically, since we aim for a worst-case bound, each edge update in the sliding window should actually *simulate* this many steps of the static computation.

Since $|\tilde{E}_i| = O(|\mathcal{M}| \cdot \alpha_i/\epsilon)$ and $t = \Theta(\epsilon \cdot |\mathcal{M}|)$, it follows that each edge update should simulate $O(|\tilde{E}_i|)/t = O(\alpha_i \cdot \epsilon^{-2})$ computation steps. Indeed, in the algorithm of Sect. 4 each edge update simulates the same number of steps $O(\alpha \cdot \epsilon^{-2}) = O(\alpha_i \cdot \epsilon^{-2})$ of the static computation. However, in contrast to the case of static arboricity bound, the arboricity bound α_j in the general case may be significantly smaller than α_i in subsequent edge updates $j, i \ll j \leq i+t-1$. To adapt the algorithm of Sect. 4 to cope with dynamically changing arboricity bound, the only change that we make is to let the jth edge update simulate $O(\alpha_j \cdot \epsilon^{-2})$ (rather than $O(\alpha_i \cdot \epsilon^{-2})$) computation steps. However, in general, the constantly changing number $O(\alpha_j \cdot \epsilon^{-2})$ of simulated steps may be significantly smaller than $O(\alpha_i \cdot \epsilon^{-2})$. Consequently, it may seem that we cannot cover the entire cost of the static computation, namely, $O(|\tilde{E}_i|)$.

The key observation is that we can upper bound the size $|\tilde{E}_i|$ of the sparsifier $\tilde{G}_i$ in terms of the minimum arboricity bound of any graph within this sliding window, denoted by α_{min}. Indeed, consider any graph G_{min} within this sliding window with arboricity bounded by α_{min}, and let $\tilde{G}_{min} = (V, \tilde{E}_{min})$ be the graph obtained from the matching sparsifier $\tilde{G}_i$ by removing from it all the edges that were deleted throughout the current sliding window. Obviously, $\tilde{G}_{min}$ is a subgraph of G_{min}, hence the arboricity of $\tilde{G}_{min}$ is bounded by α_{min}. Thus, just as in Lemma 4.3, we have $|\tilde{E}_{min}| = O(|\mathcal{M}| \cdot \alpha_{min}/\epsilon)$. Since in each edge update of the sliding window at most one edge is deleted, it follows that

$$\begin{aligned} |\tilde{E}_i| &\leq |\tilde{E}_{min}| + t \leq O(|\mathcal{M}| \cdot \alpha_{min}/\epsilon) + t \\ &= O(|\mathcal{M}| \cdot \alpha_{min}/\epsilon). \end{aligned}$$

As a result, it suffices that each edge update j within the sliding window would simulate only

$$O(|\tilde{E}_i|)/t = O(\alpha_{min} \cdot \epsilon^{-2}) = O(\alpha_j \cdot \epsilon^{-2})$$

steps of the static computation, as required.

In order for the resulting algorithm to apply to dynamic arboricity, the matching sparsifier that it employs should also apply to this setting. On the other hand, our matching sparsifier is derived from an approximate MCVC, and the algorithm of Sect. 2 for maintaining $(2+\epsilon)$-MCVCs applies to static arboricity. We address this issue in the following section.

5.2 Dynamic $(2+\epsilon)$-MCVCs. Recall that the $(2+\epsilon)$-MCVC algorithm of Sect. 2 is local. Adapting this algorithm to cope with dynamically changing arboricity bound is not difficult. However, the adapted algorithm will not be local. (It is quite difficult to adapt this algorithm while preserving locality.) Another option is to adjust the $(1+\epsilon)$-MCM algorithm of Sect. 5.1 (that applies to dynamic arboricity) to maintain a $(2+\epsilon)$-MCVC. Since both options result in a global algorithm and the latter option is easier, we present here the latter.

As before, suppose that the sliding window starts at update step i (with $\mathcal{M} = \mathcal{M}_i$), let $G_i = (V, E_i)$ be the graph at this step with an arboricity bound of α_i, and let $\tilde{G}_i$ be the corresponding matching sparsifier. Rather than computing a $(1+\epsilon)$-MCM over the matching sparsifier $\tilde{G}_i$ as done by the algorithm of Sect. 5.1, here we compute a maximal matching over $\tilde{G}_i$. It can be verified that a maximal matching for the matching sparsifier $\tilde{G}_i$ provides an almost-maximal matching for the entire graph G_i. (The proof of this assertion follows similar lines as those in the proof of Lemma 4.4, and is thus omitted.) Consequently, the induced vertex cover

$\widetilde{VC}$ (i.e., the set of all matched vertices) provides a $(2+\epsilon)$-MCVC for G_i. Throughout the sliding window, for any edge update (u, v), the two endpoints u and v are taken to the vertex cover $\widetilde{VC}$. Since the sliding window is of length $t = \Theta(\epsilon \cdot |\mathcal{M}|)$, it follows that the maintained vertex cover $\widetilde{VC}$ provides a $(2 + O(\epsilon))$-MCVC. At the beginning of the next sliding window, a new matching sparsifier is computed with respect to the updated vertex cover $\widetilde{VC}$, and this process is repeated. (That is, we compute a maximal matching over the new matching sparsifier and take its induced vertex cover, and throughout the sliding window all vertices incident on updated vertices are taken to the newly updated vertex cover.) Note that throughout the sliding window the gradually-computed vertex cover is useless, so the previously-computed vertex cover is re-used. This means that each vertex cover is re-used throughout a time window of twice as many updates, hence the approximation guarantee increases by another additive term of $O(\epsilon)$. (Of course, we can reduce the approximation factor to $2+\epsilon$ at the expense of increasing the update time by an appropriate constant.)

The (static) time needed to compute a maximal matching over the matching sparsifier $\tilde{G}_i$, as well as its induced vertex cover, is $O(|\tilde{E}_i|)$. As with the algorithm of Sect. 5.1, we let the jth edge update simulate $O(\alpha_j \cdot \epsilon^{-2})$ steps of the static computation. Just as before, it holds that $|\tilde{E}_i| = O(|\mathcal{M}| \cdot \alpha_{min}/\epsilon)$, where α_{min} is the minimum arboricity bound of any graph within this sliding window. Consequently, it suffices that each edge update j within the sliding window would simulate only $O(|\tilde{E}_i|)/t = O(\alpha_{min} \cdot \epsilon^{-2}) = O(\alpha_j \cdot \epsilon^{-2})$ computation steps, as required. (Note that this update time is greater than that of the local $(2+\epsilon)$-MCVC algorithm from Sect. 2 by a factor of $1/\epsilon$.) This completes our argument.

Acknowledgments

The authors are grateful to Ofer Neiman for helpful discussions.

References

[1] A. Abboud and V. V. Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *Proc. 55th FOCS*, pages 434–443, 2014.

[2] K. J. Ahn and S. Guha. Linear programming in the semi-streaming model with application to the maximum matching problem. *Inf. Comput.*, 222:59–79, 2013.

[3] N. Alon, R. Rubinfeld, S. Vardi, and N. Xie. Space-efficient local computation algorithms. In *Proc. 23rd SODA*, pages 1132–1139, 2012.

[4] A. Anand. Fully dynamic algorithms for maintaining approximate matchings in graphs. *Master's thesis, IIT Kanpur*, 2012.

[5] A. Anand, S. Baswana, M. Gupta, and S. Sen. Maintaining approximate maximum weighted matching in fully dynamic graphs. In *Proc. 32nd FSTTCS*, pages 257–266, 2012.

[6] S. Baswana, M. Gupta, and S. Sen. Fully dynamic maximal matching in $O(\log n)$ update time. In *Proc. of 52nd FOCS*, pages 383–392, 2011.

[7] A. Bernstein and C. Stein. Fully dynamic matching in bipartite graphs. In *Proc. 42nd ICALP*, pages 167–179, 2015.

[8] S. Bhattacharya, M. Henzinger, and G. F. Italiano. Deterministic fully dynamic data structures for vertex cover and matching. In *Proc. 26th SODA*, pages 785–804, 2015.

[9] B. Bosek, D. Leniowski, P. Sankowski, and A. Zych. Online bipartite matching in offline time. In *Proc. 55th FOCS*, pages 384–393, 2014.

[10] G. S. Brodal and R. Fagerberg. Dynamic representation of sparse graphs. In *Proc. of 6th WADS*, pages 342–351, 1999.

[11] R. Duan and S. Pettie. Linear-time approximation for maximum weight matching. *J. ACM*, 61(1):1, 2014.

[12] G. Even, M. Medina, and D. Ron. Best of two local models: Local centralized and local distributed algorithms. *CoRR*, abs/1402.3796, 2014.

[13] G. Even, M. Medina, and D. Ron. Distributed maximum matching in bounded degree graphs. In *Proc. 16th ICDCN*, page 18, 2015.

[14] A. Goel, M. Kapralov, and S. Khanna. On the communication and streaming complexity of maximum bipartite matching. In *Proc. 23rd SODA*, pages 468–485, 2012.

[15] M. Gupta. Maintaining approximate maximum matching in an incremental bipartite graph in polylogarithmic update time. In *Proc. 34th FSTTCS*, pages 227–239, 2014.

[16] M. Gupta and R. Peng. Fully dynamic $(1+\epsilon)$-approximate matchings. In *Proc. 54th FOCS*, pages 548–557, 2013.

[17] M. Gupta and A. Sharma. An $O(\log(n))$ fully dynamic algorithm for maximum matching in a tree. *CoRR*, abs/0901.2900, 2009.

[18] P. Hall. On representatives of subsets. *J. London Math. Soc*, 10(1):26–30, 1935.

[19] M. He, G. Tang, and N. Zeh. Orienting dynamic graphs, with applications to maximal matchings and adjacency queries. In *Proc. 25th ISAAC*, pages 128–140, 2014.

[20] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973.

[21] Z. Ivković and E. L. Lloyd. Fully dynamic maintenance of vertex cover. In *Proc. of 19th WG*, pages 99–111, 1993.

[22] M. Kapralov. Better bounds for matchings in the

streaming model. In *Proc. 24th SODA*, pages 1679–1697, 2013.

[23] S. Khot and O. Regev. Vertex cover might be hard to approximate to within 2-epsilon. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008.

[24] C. Konrad, F. Magniez, and C. Mathieu. Maximum matching in semi-streaming with few passes. In *Proc. 15th APPROX*, pages 231–242, 2012.

[25] T. Kopelowitz, R. Krauthgamer, E. Porat, and S. Solomon. Orienting fully dynamic graphs with worst-case time bounds. In *Proc. 41st ICALP*, pages 532–543, 2014.

[26] T. Kopelowitz, S. Pettie, and E. Porat. 3sum hardness in (dynamic) data structures. *CoRR*, abs/1407.6756, 2014.

[27] Z. Lotker, B. Patt-Shamir, and S. Pettie. Improved distributed approximate matching. In *Proc. 20th SPAA*, pages 129–136, 2008.

[28] Z. Lotker, B. Patt-Shamir, and A. Rosén. Distributed approximate matching. *SIAM J. Comput.*, 39(2):445–460, 2009.

[29] Y. Mansour and S. Vardi. A local computation approximation scheme to maximum matching. In *Proc. 16th APPROX*, pages 260–273, 2013.

[30] S. Micali and V. V. Vazirani. An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs. In *Proc. 21st FOCS*, pages 17–27, 1980.

[31] O. Neiman and S. Solomon. Simple deterministic algorithms for fully dynamic maximal matching. In *Proc. 45th STOC*, pages 745–754, 2013.

[32] K. Onak and R. Rubinfeld. Maintaining a large matching and a small vertex cover. In *Proc. of 42nd STOC*, pages 457–464, 2010.

[33] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM, 2000.

[34] S. Pettie. A simple reduction from maximum weight matching to maximum cardinality matching. *Inf. Process. Lett.*, 112(23):893–898, 2012.

[35] R. Rubinfeld, G. Tamir, S. Vardi, and N. Xie. Fast local computation algorithms. In *Proc. 2nd ICS*, pages 223–238, 2011.

[36] P. Sankowski. Faster dynamic matchings and vertex connectivity. In *Proc. of 18th SODA*, pages 118–126, 2007.

[37] J. Suomela. Survey of local algorithms. *ACM Comput. Surv.*, 45(2):24, 2013.

[38] V. V. Vazirani. An improved definition of blossoms and a simpler proof of the MV matching algorithm. *CoRR*, abs/1210.4594, 2012.

Appendix

A Arboricity Versus Vertex Cover Number

In this appendix we show a simple connection between the arboricity of a graph and the size of an MCVC for it (which is sometimes called the *vertex cover number* of the graph).

We first note that the arboricity of a graph may be much smaller than the size of an MCVC for it. For example, consider an n-vertex graph whose edge set constitutes a perfect matching. The arboricity of such a graph is 1, but any vertex cover for it is of size at least $\lfloor n/2 \rfloor$. Summarizing, we have:

FACT A.1. *There exist n-vertex graphs for which the size of an MCVC is larger than the arboricity by a factor of $\Omega(n)$.*

Next, we show that the arboricity of a graph is no greater than the size of an MCVC for it.

THEOREM A.1. *Let $G = (V, E)$ be a graph of arboricity α. For any vertex cover VC for G, $|VC| \geq \alpha$.*

Proof. Let $G[V'] = (V', E')$ be a minimal subgraph of G realizing the arboricity, i.e., $\lceil |E'|/(|V'| - 1) \rceil = \alpha$ and for every subgraph $G[V''] = (V'', E'')$ over a proper subset $V'' \subsetneq V'$, it holds that $\lceil |E''|/(|V''| - 1) \rceil \leq \alpha - 1$. Since $\lceil |E'|/(|V'| - 1) \rceil = \alpha$, it follows that $|E'| > (\alpha - 1) \cdot (|V'| - 1)$.

We argue that all vertices in V' have degree at least α in $G' = G[V']$. Suppose for contradiction otherwise, and let $v \in V'$ be a vertex with degree at most $\alpha - 1$ in G'. Consider the graph $G[V''] = (V'', E'')$ induced by the vertex set $V'' = V' \setminus \{v\}$. Since $|E''| \geq |E'| - \alpha + 1$, we have

$$\begin{aligned} \left\lceil \frac{|E''|}{|V''| - 1} \right\rceil &\geq \frac{|E''|}{|V''| - 1} \geq \frac{|E'| - \alpha + 1}{|V'| - 2} \\ &> \frac{(\alpha - 1) \cdot (|V'| - 1) - \alpha + 1}{|V'| - 2} \\ &= \alpha - 1, \end{aligned}$$

yielding a contradiction.

The first immediate consequence is that $|V'| \geq \alpha + 1$. Moreover, we have $\deg_G(v) \geq \deg_{G'}(v) \geq \alpha$, for each vertex $v \in V'$. Consider an arbitrary vertex cover VC for G. If all vertices in V' belong to VC, then $|VC| \geq \alpha + 1$. Otherwise at least one vertex $v \in V'$ does not belong to VC, but then all its $\deg_G(v) \geq \alpha$ neighbors must belong to VC, which gives $|VC| \geq \alpha$. ∎