Dynamic Matching: Reducing Integral Algorithms to Approximately-Maximal Fractional Algorithms

Moab Arar[1], Shiri Chechik[*1], Sarel Cohen[1], Cliff Stein[*†2], and David Wajc[*‡3]

[1]Tel Aviv University
[2]Columbia University
[3]Carnegie Mellon University

May 14, 2018

Abstract

We present a simple randomized reduction from fully-dynamic integral matching algorithms to fully-dynamic "approximately-maximal" fractional matching algorithms. Applying this reduction to the recent fractional matching algorithm of Bhattacharya, Henzinger, and Nanongkai (SODA 2017), we obtain a novel result for the integral problem. Specifically, our main result is a randomized fully-dynamic $(2+\epsilon)$-approximate integral matching algorithm with small polylog worst-case update time. For the $(2+\epsilon)$-approximation regime only a *fractional* fully-dynamic $(2+\epsilon)$-matching algorithm with worst-case polylog update time was previously known, due to Bhattacharya et al. (SODA 2017). Our algorithm is the first algorithm that maintains approximate matchings with worst-case update time better than polynomial, for any constant approximation ratio. As a consequence, we also obtain the first constant-approximate worst-case polylogarithmic update time maximum weight matching algorithm.

1 Introduction

The maximum matching problem is one of the most widely-studied problems in computer science and operations research, with a long history and theory [5, 38]. On n-vertex and m-edge graphs, the state-of-the art maximum matching algorithms require $O(m\sqrt{n})$ and $O(n^\omega)$ time [40, 41] (here $\omega < 2.37$ is the matrix multiplication exponent [54]). For bipartite graphs, simpler algorithms with the same asymptotic running times are known [34, 41], as well as a faster, $O(m^{10/7} \cdot \text{poly}(\log n))$-time algorithm, due to the recent breakthrough of Mądry[39] for the maximum flow problem. For approximate matchings, it is long- and well-known that a matching admitting no augmenting paths of length $O(1/\epsilon)$ forms a $(1+\epsilon)$-approximate maximum matching (see [34]). The linear-time "blocking flow" subroutines of [34, 40] therefore result in an $O(m/\epsilon)$-time $(1+\epsilon)$-approximate maximum matching.

The maximum weight matching (MWM) problem has also garnered much interest over the years. For general weights, the seminal work of Edmonds [22] shows how to reduce the problem on bipartite graphs to the solution of n non-negative single-source shortest path instances. Relying

[*]Work done in part while the authors were at the Simons Institute for the Theory of Computing.
[†]Supported in part by NSF grant CCF-1421161.
[‡]Supported in part by NSF grants CCF-1527110, CCF-1618280 and NSF CAREER award CCF-1750808.

on Fibonacci Heaps of Fredman and Tarjan [23], this approach yields the current fastest strongly-polynomial running time for the problem, $O(n(m+n\log n))$. Gabow [24] later showed how to obtain the same running time for general graphs. For integer weights $w : E \to \{0, 1, 2, \ldots, N\}$, algorithms nearly matching the state-of-of-the-art for the unweighted problem, with either logarithmic or linear dependence on N, are known.[1] These include an $O(m\sqrt{n}\log(nN))$-time algorithm [25], an $O(Nn^{\omega})$-time algorithm [48] and a recent $O(m^{10/7}\cdot\text{poly}(\log n)\cdot\log N)$-time algorithm for bipartite graphs [17]. For approximation algorithms, an algorithm nearly matching the unweighted problem's guarantees is known, yielding a $(1+\epsilon)$-approximate maximum weight matching in $O((m/\epsilon)\log(1/\epsilon))$ time, [21].

All of the above results pertain to the static problem; i.e., where the input is given and we only need to compute a maximum matching on this given input. However, in many applications the graphs considered are inherently *dynamic*, with edges removed or added over time. One could of course address such changes by recomputing a solution from scratch, but this could be wasteful and time-consuming, and such applications may require immediately updating the solution given, as having users wait on a solution to be recomputed may likely be unsatisfactory. Consider for example point to point shortest path computation, a problem routinely solved by navigation systems: for such an application, the temporary closure of some road due to construction should not result in unresponsive GPS applications, busy re-computing the relevant data structures (see e.g.,[6, 36, 33, 19, 20, 51, 52, 9, 26, 28, 2, 31, 29, 30, 32, 3, 4]). Therefore, for such applications we want to update our solution quickly for *every update*, using fast *worst-case* (rather than amortized) update time.

Returning to the maximum matching problem, we note that a maximum matching can be trivially updated in $O(m)$ time. Sankowski [47] showed how to maintain the *value* of the maximum matching in $O(n^{1.495})$ update time.[2] On the other hand, Abboud and Vassilevska Williams [1] and Kopelowitz et al. [37] presented lower bounds based on long-standing conjectures, showing that even maintaining the maximum matching value likely requires $\Omega(m^c)$ update time for some constant $c \geq \frac{1}{3}$.

Given these hardness results for exact solutions, one is naturally inclined to consider fast *approximate* solutions. Trivially updating a maximal matching (and therefore a 2-approximate maximum matching) can be done using $O(n)$ worst-case update time. The goal is to obtain sublinear update times – ideally polylogarithmic (or even constant) – with as low an approximation ratio as possible.

The first non-trivial result for fully-dynamic maximum matching is due to Ivkovic and Lloyd [35], who presented a maximal matching algorithm with $O((m+n)^{1/\sqrt{2}})$ amortized update time. Note that this bound is sublinear only for sufficiently sparse graphs. The problem of approximate maximum matchings remained largely overlooked until 2010, when Onak and Rubinfeld [44] presented a fully-dynamic constant-approximate $O(\log^2 n)$ (amortized) update time algorithm. Additional results followed in quick succession.

Baswana et al. [7] showed how to maintain a maximal matching in $O(\log n)$ expected update time, and $O(\log^2 n)$ update time w.h.p. This was recently improved by Solomon [49] who presented a maximal matching algorithm using $O(1)$ update time w.h.p. For deterministic algorithms, Neiman and Solomon [42] showed how to maintain 3/2-approximate matchings deterministically in $O(\sqrt{m})$ update time, a result later improved by Gupta and Peng [27] to obtain $(1+\epsilon)$-approximate matchings in $O(\sqrt{m}/\epsilon^2)$. This result was in turn refined by Peleg and Solomon [45], who obtained the same approximation ratio and update time as [27] with $\sqrt{m}$ replaced by the maximum arboricity of the graph α (which is always at most $\alpha = O(\sqrt{m})$). Bernstein and Stein [10, 11] and Bhattacharya et al. [12] presented faster polynomial update time algorithms (with higher approximation ratios),

[1]Indeed, a black-box reduction of Pettie [46] from maximum weight matching to the maximum matching problem shows that a linear dependence in N is the largest possible gap between these two problems.

[2]We emphasize that this algorithm does not maintain an actual matching, but only the optimal value, and it seems unlikely to obtain such update times for maintaining a matching of this value.

and Bhattacharya et al. [13] presented a $(2+\epsilon)$-approximate algorithm with $\text{poly}(\log n, \epsilon^{-1})$ update time. See Table 1 for an in-depth tabular exposition of previous work and our results.[3] In §5 we discuss our results for MWM, also widely studied in the dynamic setting (see, e.g. [7, 27, 49, 50]).

Note that in the previous paragraph we did not state whether the update times of the discussed algorithms were worst case or amortized. We now address this point. As evidenced by Table 1, previous fully-dynamic matching algorithms can be broadly divided into two classes according to their update times: polynomial update time algorithms and polylogarithmic *amortized* update time algorithms. The only related polylogarithmic worst-case update time algorithms known to date were *fractional* matching algorithms, due to Bhattacharya et al. [14]. We bridge this gap by presenting the first fully-dynamic integral matching (and weighted matching) algorithm with polylogarithmic worst-case update times and constant approximation ratio. In particular, our approach yields a $(2+\epsilon)$-approximate algorithm, within the $O_\epsilon(\log^3 n)$ time bound of [14], but for *integral* matching.[4]

Approx.	Update Time	det.	w.c.	notes	reference
$O(1)$	$O(\log^2 n)$	✗	✗		Onak and Rubinfeld (STOC '10) [44]
$4+\epsilon$	$O(m^{1/3}/\epsilon^2)$	✓	✓		Bhattacharya et al. (SODA '15) [12]
$3+\epsilon$	$O(\sqrt{n}/\epsilon)$	✓	✗		Bhattacharya et al. (SODA '15) [12]
$2+\epsilon$	$\text{poly}(\log n, 1/\epsilon)$	✓	✗		Bhattacharya et al. (STOC '16) [13]
$2+\epsilon$	$\text{poly}(\log n, 1/\epsilon)$	✗	✓	w.h.p	**This work**[4]
2	$O((m+n)^{1/\sqrt{2}})$	✓	✗		Ivković and Lloyd (WG '93) [35]
2	$O(\log n)$	✗	✗	$O(\log^2 n)$ w.h.p	Baswana et al. (FOCS '11) [8]
2	$O(1)$	✗	✗	w.h.p	Solomon (FOCS '16) [49]
$3/2+\epsilon$	$O(\sqrt[4]{m}/\epsilon^{2.5})$	✓	✓	bipartite only	Bernstein and Stein (ICALP '15) [10]
$3/2+\epsilon$	$O(\sqrt[4]{m}/\epsilon^{2.5})$	✓	✗		Bernstein and Stein (SODA '16) [11]
$3/2$	$O(\sqrt{m})$	✓	✓		Neiman and Solomon (STOC '13) [43]
$1+\epsilon$	$O(\sqrt{m}/\epsilon^2)$	✓	✓		Gupta and Peng (FOCS '13) [27]

Table 1: Our Results and Previous Results for Fully-Dynamic Matching.
(All references are to the latest publication, with the first publication venue in parentheses.)

1.1 Our Contribution

Our main technical result requires the following natural definition of (c,d)-approximately-maximal fractional matchings.

Definition 1.1 (Approximately-Maximal Fractional Matching). *We say that a fractional matching $w : E \to \mathbb{R}^+$ is* (c,d)-approximately-maximal *if every edge $e \in E$ either has fractional value $w_e \geq 1/d$ or has one endpoint v with sum of incident edges' weights at least $W_v \triangleq \sum_{e \ni v} w_e \geq 1/c$ and moreover all edges e' incident on this v have $w_{e'} \leq 1/d$.*

[3]For the sake of simplicity we only list bounds here given in terms of n and m. In particular, we do not state the results for arboricity-bounded graphs, which in the worst case (when the arboricity of a graph is $\alpha = \Theta(\sqrt{m})$) are all outperformed by algorithms in this table, with the aforementioned algorithm of Peleg and Solomon [45] being the lone exception to this rule.

[4]Independently of our work, and using a different approach, Charikar and Solomon [16] obtained a $(2+\epsilon)$-approximate dynamic matching algorithm with $O_\epsilon(\log^7 n)$ worst-case update time. For fixed ϵ their algorithm is slower than ours, and is arguably more complicated than our approach.

Note that this definition generalizes maximal fractional matchings (for which $c = d = 1$). The second condition required of v above (i.e., having no incident edges e' with $w_{e'} > 1/d$) may seem a little puzzling, but will prove important later; it can be safely ignored until §2.1 and §3.

Our main qualitative result, underlying our quantitative result, is the following black-box reduction from integral matching algorithms to approximately-maximal fractional matching algorithms, as stated in the following theorem.

Theorem 1.2. *Let $\mathcal{A}$ be a fully-dynamic (c, d)-approximately-maximal fractional matching algorithm whose update time $T(n, m)$ and which changes at most $C(n, m)$ edge weights per update, for some $c \geq 1$, $d \geq \frac{6c\cdot\ln(\max\{n,1/\epsilon\})}{\epsilon^2}$, with $\epsilon \leq \frac{1}{2}$. Then, there exists a randomized fully-dynamic* integral $2c(1 + O(\epsilon))$*-approximate matching algorithm $\mathcal{A}'$ (with this bound holding both w.h.p and in expectation) with update time $T(n, m) + O(C(n, m) \cdot d/\epsilon^2)$. Moreover, if $T(n, m)$ and $C(n, m)$ are worst-case bounds, so is the update time of Algorithm $\mathcal{A}'$.*

Now, one may wonder whether fully-dynamic (c, d)-approximately-maximal fractional matching algorithms with low worst-case update time and few edge weight changes exist for any non-trivial values of c and d. Indeed, the recent algorithm of Bhattacharya et al. [14] is such an algorithm, as the following lemma asserts.

Lemma 1.3 ([14])**.** *For all $\epsilon \leq \frac{1}{2}$, there is a fully-dynamic $(1 + 2\epsilon, \max\{54 \log n/\epsilon^3, (3/\epsilon)^{21}\})$-approximately-maximal fractional matching algorithm with $T(n, m) = O(\log^3 n/\epsilon^7)$ worst-case update time, using at most $C(n, m) = O(\log n/\epsilon^2)$ edge weight changes per update in the worst case.*

We highlight the general approach of the algorithm of Bhattacharya et al. [14] in §2.1 to substantiate the bounds given in Lemma 1.3. Plugging the values of c, $T(n, m)$ and $C(n, m)$ of Lemma 1.3 into Theorem 1.2 immediately yields our result, given in the following theorem.

Theorem 1.4. *For all $\epsilon \leq \frac{1}{2}$, there exists a randomized fully-dynamic $(2 + O(\epsilon))$-approximate integral matching algorithm (w.h.p and in expectation) with worst-case update time $O(\log^3 n/\epsilon^7 + \log(\max\{n, 1/\epsilon\})/\epsilon^2 \cdot \max\{\log n/\epsilon^3, (3/\epsilon)^{21}\}) = O_\epsilon(\log^3 n)$.*

We recall that until now (barring the aforementioned independent result of Charikar and Solomon [16]), for worst-case polylog update times only *fractional* dynamic algorithms – algorithms which only approximate the *value* of the maximum matching – were known for this problem.

Finally, combined with the recent black-box reduction of Stubbs and Vassilevska Williams [50] from the weighted to the unweighted matching problem, our algorithm also yields the first fully-dynamic constant-approximate maximum weight matching algorithm with polylogarithmic worst-case update time.

Theorem 1.5. *For all $\epsilon \leq \frac{1}{2}$, there exists a randomized fully-dynamic $(4 + O(\epsilon))$-approximate maximum weight matching algorithm with poly$(\log n, 1/\epsilon)$ worst-case update time. The approximation guarantee holds with high probability and in expectation.*

1.2 Our Techniques

Our framework yielding our main result combines three ingredients: approximately-maximal fractional matchings, kernels and fast $(1 + \epsilon)$ matching algorithms for bounded-degree graphs. We give a short exposition of these ingredients and conclude with how we combine all three.

Approximately-Maximal Fractional Matchings. The first ingredient we rely on is (c, d)-approximately-maximal fractional matchings, introduced in the previous section. Recalling that for such solutions, each edge has value at least $1/d$ or one of its endpoints has sum of incident edge values at least $1/c$. This approximate maximality condition implies this fractional matching has high value compared to the maximum matching size; specifically, this fractional matching's size is at least a $1/2 \max\{c, d\}$ fraction of this value (easily verifiable using LP duality). As we shall show, approximate maximality also allows one to use these fractional values to sample a subgraph in the support of this fractional matching which contains a large *integral* matching compared to G, with high probability. We discuss the dynamic fractional matching algorithm of Bhattacharya et al. [14] and show that it maintains an approximately-maximal fractional matching in §2.1.

Kernels. The second ingredient we rely on is the notion of *kernels*, introduced by [12]. Roughly speaking, a kernel is a low-degree subgraph H of G such that each edge of G not taken into H has at least one endpoint whose degree in H is at least $1/c$ times the maximum degree in H. Relying on Vizing's Theorem [53], we show in §2.2 that such a graph has maximum matching size $\mu(H)$ at least $1/(2c + \epsilon)$ of the matching size of G, previously only known for kernels of bipartite graphs, where this is easily verifiable via LP duality.[5] Efficiently maintaining a large matching can therefore be reduced to maintaining a low-degree kernel, given the last ingredient of our approach.

Bounded-Degree $(1 + \epsilon)$-matching. The final ingredient we rely on for our framework is $(1+\epsilon)$ matching algorithms with worst-case update time bounded by the graph's maximum degree, such as the algorithms of Gupta and Peng [27] and Peleg and Solomon [45].

Our approach in a nutshell. Given the above ingredients, our framework is a simple and natural one. Throughout our algorithm's run, we run a fully-dynamic (c, d)-approximately-maximal fractional matching algorithm with efficient worst-case update. Sampling edges independently according to this fractional value (times some logarithmic term in n, to guarantee concentration) allows us to sample a kernel of logarithmic maximum degree, with each non-sampled edge having at least one endpoint with degree at least $1/c$ times the maximum subgraph degree, with high probability. As the obtained subgraph H therefore has a maximum matching of size at least $\approx 1/2c$ times the maximum matching in G, a $(1 + \epsilon)$-matching algorithm in H yields a $\approx 2c + O(\epsilon)$ matching in G. We then maintain a $(1 + \epsilon)$-matching in H (which by virtue of H's bounded degree we can do in logarithmic worst-case time) following each update to H incurred by a change of some edge's fractional value by the dynamic fractional matching algorithm. The obtained integral algorithm's update time is dominated by two terms: the running time of the fractional algorithm, and the number of edge weight changes per update, times $O(\log n)$. This concludes the high-level analysis of the obtained approximation ratio and update time of our approach, as given in Theorem 1.2.

Wider applicability. We stress that our framework is general, and can use any approximately-maximal fractional matching algorithm. Consequently, any improvement on the running time and number of edge value changes for maintaining approximately-maximal fractional matchings would yield a faster worst-case update time.

2 Preliminaries

In this section we introduce some previous results which we will rely on in our algorithm and its analysis. We start by reviewing the approach of Bhattacharya et al. [14] to obtain efficient fractional algorithms in §2.1. We then discuss the bounded-degree subgraphs we will consider, also known as *kernels*, in §2.2. Finally, we briefly outline the $(1 + \epsilon)$-approximate $O(\Delta/\epsilon^2)$ worst case update time algorithms we will rely on for our algorithm, in §2.3.

[5]As a byproduct of our proof, we show how the algorithms of Bhattacharya et al. [12] can be made $(2 + \epsilon)$-approximate within the same time bounds. As this is tangential to our main result, we do not elaborate on this.

2.1 Hierarchical Partitions

In this section we review the approximately-maximal fractional matchings maintained by Bhattacharya et al. [14]. At a high level, this algorithm relies on the notion *hierarchical partitions*, in which vertices are assigned some *level* (the partition here is given by the level sets), and edges are assigned a fractional value based on their endpoints' levels. Specifically, an edge is assigned a value exponentially small in its vertices' maximum level. The levels (and therefore the edge weights) are updated in a way as to guarantee feasibility, as well as guaranteeing that a vertex v of high level has high sum of incident edge weights, W_v. These conditions are sufficient to guarantee approximate maximality, as we shall soon show.

The hierarchical partitions considered by Bhattacharya et al. [14], termed simply *nice partitions*, is described as follows. In the definition constants β, K, L and a function $f(\beta) = 1 - 3/\beta$ are used, satisfying the following.

$$\beta \geq 5,\ K = 20,\ f(\beta) = 1 - 3/\beta,\ L = \lceil \log_\beta n \rceil. \tag{1}$$

In our case, for some $\epsilon \leq \frac{1}{2}$, we will let $\beta = \frac{3}{\epsilon}(\geq 5)$. As we will be shooting for $O(1)$-approximation algorithms with polylogarithmic update time and our reduction's update time has polynomial dependence on ϵ^{-1}, we will assume without loss of generality that $\epsilon = \Omega(\frac{3}{\sqrt[20]{n}})$, and so for n large enough, we have $K \leq L$.

Definition 2.1 (A nice partition [14])**.** *In a* nice partition *of a graph $G = (V, E)$, each vertex v is assigned an integral* level $\ell(v)$ *in the set $\{K, K+1, \ldots, L\}$. In addition, for each vertex $v \in V$ and edge $e \ni v$ the* shadow-level of v with respect to e, *denoted by $\ell_v(e)$, is a (positive) integer satisfying $\ell(v) - 1 \leq \ell_v(e) \leq \ell(v) + 1$. Moreover, for each vertex v, we have*

$$\max_{e \ni v} \ell_v(e) - \min_{e \ni v} \ell_v(e) \leq 1. \tag{2}$$

The level of an edge $e = (u, v)$ is taken to be the maximum shadow-level of an endpoint of e with respect to e; i.e., $\ell(u, v) = \max\{\ell_u(e), \ell_v(e)\}$. Let $W_v = \sum_{e \in v} w_e$ be the sum of weights of edges incident on a vertex v. Then,

1. *For every edge $e \in E$, it holds that $w_e = \beta^{-\ell(e)}$.*
2. *For every node $v \in V$, it holds that $W_v < 1$.*
3. *For every node $v \in V$ with level $\ell(v) > K$, it holds that $W_v \geq f(\beta)$.*

The intuition behind this definition in Bhattacharya et al. [14] is to mimic the hierarchical partition of Bhattacharya et al. [12], termed (α, β)-*decompositions* there. (α, β)-decompositions are the special case of nice partitions where the shadow-level of a vertex v with respect to each edge $e \ni v$ is precisely equal to the vertex's level; i.e, $\ell_v(e) = \ell(v)$ (with α denoting $f(\beta)/\beta$). The advantage of this more relaxed notion of shadow-level is to allow a vertex to move between levels "slowly", only notifying *part* of its incident edges of its level change between updates, and therefore only updating some of its edges' weights. This allows for maintaining this partition with fast worst-case update time, as shown in Bhattacharya et al. [14] (more on this below).

This above intuition concerning nice partitions will not prove important for our analysis. The crucial property we will rely on is given by the following lemma, which asserts that the fractional matching associated with a nice partition is approximately-maximal.

Lemma 2.2. *Let $\epsilon \leq \frac{1}{2}$. Consider a nice partition with parameter $\beta = 3/\epsilon \geq 6 \geq 5$, and so $f(\beta) = 1-\epsilon$. Then, the fractional matching associated with this nice partition is $(1+2\epsilon, \max\{54\log n/\epsilon^3, (3/\epsilon)^{21}\})$-approximately-maximal.*

Proof. Let $K' = \max\{\lceil \log_\beta(18c\log n/\epsilon^2)\rceil, K+1\}$ and $d = \beta^{K'} \leq \max\{54\log n/\epsilon^3, (3/\epsilon)^{21}\}$. For any edge e, if $\ell(e) = \max_{v\in e}\{\ell_v(e)\} \leq K'$, then by definition $w_e = \beta^{-\ell(e)} \geq \beta^{-K'} = \frac{1}{d}$. Alternatively, if $w_e < \frac{1}{d}$ then $\ell(e) > K'$ and therefore by integrality of $\ell(e)$, we have $\ell(e) \geq K'+1$. Now, let v be $\arg\max_{v\in e}\{\ell_v(e)\} \geq K'+1$. Then, by definition of shadow-levels and $K' > K$, we have $\ell(v) \geq \ell_v(e)-1 \geq K' > K$ and so by Property 3 of a nice partition we have $W_v > f(\beta) = 1-\epsilon \geq \frac{1}{1+2\epsilon}$ (as $\epsilon \leq \frac{1}{2}$). But on the other hand, by Equation (2), we also know that for every edge $e' \ni v$,

$$\ell_v(e') \geq \min_{e'\ni v} \ell_v(e') \geq \max_{e'\ni v} \ell_v(e') - 1 \geq \ell_v(e) - 1 \geq K' > K.$$

Therefore, by definition of the edge weights, each edge $e' \ni v$ satisfies $w_{e'} \leq \beta^{-K'} \leq \frac{1}{d}$. □

The recent result of Bhattacharya et al. [14] for maintaining nice partitions in polylogarithmic worst-case update time together with Lemma 2.2 immediately implies Lemma 1.3, restated below. We substantiate these bounds with the dependence on ϵ stated explicitly in the §A, as Bhattacharya et al. [14] had $\epsilon = O(1)$ and so their results do not state this dependence explicitly.

Lemma 1.3 ([14])**.** *For all $\epsilon \leq \frac{1}{2}$, there is a fully-dynamic $(1+2\epsilon, \max\{54\log n/\epsilon^3, (3/\epsilon)^{21}\})$-approximately-maximal fractional matching algorithm with $T(n,m) = O(\log^3 n/\epsilon^7)$ worst-case update time, using at most $C(n,m) = O(\log n/\epsilon^2)$ edge weight changes per update in the worst case.*

As we shall show, approximately-maximal fractional matchings allow us to sample a bounded-degree subgraph H of G containing a large matching compared to the maximum matching size in G, $\mu(G)$. For this we will require the notion of *kernels*, defined in §2.2.

2.2 Kernels

In this section we review the concept of kernels, first introduced by Bhattacharya et al. [12].

Definition 2.3 (Kernels [12])**.** *A* (c,d)-kernel *of a graph G is a subgraph H of G satisfying:*

1. *For each vertex $v \in V$, the degree of v in H is at most $d_H(v) \leq d$.*
2. *For each edge $(u,v) \in E \setminus H$, it holds that* $\max\{d_H(u), d_H(v)\} \geq d/c$.

The interest in finding a bounded-degree subgraph H of G may seem natural, as one may expect to be able to compute a matching quickly in H due to its sparsity (we elaborate more on this point in §2.3). The interest in satisfying the second property, on the other hand, may seem a little cryptic. However, combining both properties implies that the matching number of H, $\mu(H)$, is large in comparison with the matching number of G, $\mu(G)$.

Lemma 2.4. *Let H be a (c,d)-kernel of G for some $c \geq 1$. Then $\mu(H) \geq \frac{1}{2c(1+1/d)} \cdot \mu(G)$.*

Proof. Let M^* be some maximum matching in G (i.e., $|M^*| = \mu(G)$). Consider the following fractional matching solution:

$$f_{u,v} = \begin{cases} \frac{1}{d} & (u,v) \in H \setminus M^* \\ \max\{1 - \frac{d_h(u)+d_H(v)-2}{d}, 0\} & (u,v) \in H \cap M^*. \end{cases}$$

This is a feasible fractional matching due to the degree bound of H and the fractional values assigned to edges of a vertex v incident on an edge $e \in H \cap M^*$ being at most $\frac{d_H(v)-1}{d} + \frac{d-d_H(v)+1}{d} = 1$. To show that this fractional matching has high value, consider the variables $y_v = \sum_u f_{u,v}$. On the one hand, by the handshake lemma, $\sum_{u,v} f_{u,v} = \frac{1}{2}\sum_v y_v$. On the other hand, each edge (u,v) of $M^* \cap H$ has $y_u + y_v \geq 1 \geq \frac{1}{c}$ by construction and each edge of $M^* \setminus H$ has at least one endpoint v of degree $d_H(v) \geq \frac{d}{c}$, implying that $y_u + y_v \geq \frac{d}{c} \cdot \frac{1}{d} = \frac{1}{c}$ for each $(u,v) \in M^* \setminus H$. As each vertex v neighbors at most one edge of the (optimal) matching M^*, we obtain

$$\sum_e f_e = \frac{1}{2} \cdot \sum_v y_v \geq \frac{1}{2c} \cdot |M^*| = \frac{1}{2c} \cdot \mu(G).$$

Now, to show that H contains a large *integral* matching, we rely on Vizing's Theorem [53], which asserts that every multigraph of maximum degree Δ and maximum edge multiplicity μ has a proper $\Delta + \mu$ edge-coloring; i.e., a partition of the edge multiset into $\Delta + \mu$ edge-disjoint matchings. To use this theorem, we construct a multigraph on the same vertex set V with each edge e replaced by $f_e \cdot d$ parallel copies (note that $f_e \cdot d$ is integral). By construction, the number of edges in this multigraph is $\sum_e f_e \cdot d$. By feasibility of f, we have that this multigraph has maximum degree d. By Vizing's Theorem, the simple subgraph obtained by ignoring parallel edges corresponding to edges in $H \cap M^*$ can be edge colored using $d+1$ colors. But for each edge $e = (u,v) \in H \cap M^*$, such a coloring uses at most $d_H(u) - 1 + d_H(v) - 1$ distinct colors on edges other than (u,v) incident on u or v. To extend this $d+1$ edge coloring to a proper coloring of the multigraph, we color the $\max\{d - (d_H(u) - 1 + d_H(v) - 1), 0\}$ multiple edges (u,v) in this multigraph using some $\max\{d - (d_H(u) - 1 + d_H(v) - 1), 0\}$ colors of the palette of size $d+1$ not used on the other edges incident on u and v. We conclude that this multigraph, which is contained in H and has $\sum_e f_e \cdot d$ edges, is $d+1$ edge colorable and therefore H contains an integral matching of size at least

$$\frac{1}{d+1} \cdot \sum_e f_e \cdot d = \frac{1}{1+1/d} \cdot \sum_e f_e \geq \frac{1}{2c(1+1/d)} \cdot \mu(G). \qquad \square$$

Lemma 2.4 and the algorithm of §2.3 immediately imply that the algorithms of Bhattacharya et al. [12] can be made $(2+\epsilon)$-approximate within the same time bounds (up to poly$(1/\epsilon)$ terms). As this was previously also observed in Bhattacharya et al. [13], we do not elaborate on this point here.

2.3 Nearly-Maximum Matchings in Degree-Bounded Graphs

In this short subsection we highlight one final component we will rely on for our reduction: fast nearly-optimal matching algorithms with worst-case update time bounded by G's maximum degree. Such algorithms were given by Peng and Gupta [27] and Peleg and Solomon [45]. More precisely, we have the following lemma.

Lemma 2.5 ([27, 45]). *There exists a dynamic $(1+\epsilon)$-approximate matching algorithm with worst-case $O(\Delta/\epsilon^2)$ update time in dynamic graphs of maximum degree at most Δ.*

The bound for the algorithm of [45] follows as $\alpha \leq \Delta$ always, while the bound for the algorithm of [27] is immediate by inspecting this algorithm, as observed in [45].

3 Sampling Using Approximately-Maximal Matchings

In what follows we will show that sampling edges independently with probability roughly proportional to their assigned value according to an approximately-maximal fractional matching yields a kernel of logarithmic maximum degree with high probability.

Lemma 3.1. *Let $\epsilon \leq \frac{1}{2}$. Let $w : E \to \mathbb{R}^+$ be a (c,d)-approximately-maximal fractional matching with $c \geq 1$ and $d \geq \frac{6c\cdot\ln(\max\{n,1/\epsilon\})}{\epsilon^2}$. Then, a subgraph H obtained by sampling each edge e independently with probability*

$$\min\{1, w_e \cdot d\} \tag{3}$$

is a $(c(1+O(\epsilon)),(1+\epsilon)d)$-kernel of G with probability at least $1 - \frac{2}{\max\{n,1/\epsilon\}}$.

Proof. For any vertex $v \in V$, denote by D_v the random variable which corresponds to v's degree in H. As before, denote by $W_v = \sum_{e \ni v} w_e \leq 1$ the sum of edge weights of edges incident on v.

First, we prove the degree upper bound; i.e., Property 1 of a kernel. As w is a fractional matching, we know that $W_v \leq 1$. Therefore, by Equation (3), we have that $\mathbb{E}[D_v] \leq d$. By standard Chernoff bounds, as $d \geq \frac{6c\cdot\ln(\max\{n,1/\epsilon\})}{\epsilon^2}$ and $c \geq 1$, we find that

$$\Pr[D_v \geq (1+\epsilon)\cdot d] \leq \exp\left(\frac{-\epsilon^2 \cdot d}{3}\right) \leq \frac{1}{\max\{n,1/\epsilon\}^{2c}} \leq \frac{1}{\max\{n,1/\epsilon\}^2}.$$

Next, we prove that any edge not sampled into H will, with high probability, be incident on some high-degree vertex in H; i.e., we show that H satisfies Property 2 of a kernel. First, note that an edge e with $w_e \geq 1/d$ will be sampled with probability one, given our sampling probability given in Equation (3), therefore trivially satisfying Property 2 of a kernel. Conversely, an edge e with $w_e < 1/d$ has some endpoint v with $W_v \geq 1/c$ and all edges e' incident on v have $w_{e'} \leq 1/d$, since w is (c,d)-approximately maximal. Therefore, by Equation (3) each edge e' incident on v is sampled with probability precisely $w_{e'} \cdot d$. Consequently, we have that $\mu_v = \mathbb{E}[D_v] = W_v \cdot d \geq d/c$. By standard Chernoff bounds, as $\mu_v \geq d/c \geq \frac{6\ln(\max\{n,1/\epsilon\})}{\epsilon^2}$, we find that

$$\Pr[D_v \leq (1-\epsilon)\cdot d/c] \leq \Pr[D_v \leq (1-\epsilon)\cdot \mu_v] \leq \exp\left(\frac{-\epsilon^2 \cdot \mu_v}{2}\right) \leq \frac{1}{\max\{n,1/\epsilon\}^3}.$$

Taking a union bound over the $O(n^2)$ possible bad events corresponding to violating a property of a $(c(1+\epsilon)/(1-\epsilon), d(1+\epsilon))$-kernel, we find that with probability at least $1 - \frac{2}{\max\{n,1/\epsilon\}}$:

1. For each vertex $v \in V$, it holds that $d_H(v) \leq (1+\epsilon)\cdot d$.
2. For each edge $(u,v) \in E \setminus H$, it holds that $\max\{d_H(u), d_H(v)\} \geq (1-\epsilon)\cdot d/c$.

In other words, H is a $(c(1+\epsilon)/(1-\epsilon), d(1+\epsilon))$-kernel of G with high probability. □

4 Our Reduction

Given the previous sections, we are now ready to describe our reduction from fully-dynamic integral matching to approximately-maximal fractional matching and analyzing its performance, given by Theorem 1.2, restated here.

Theorem 1.2. *Let $\mathcal{A}$ be a fully-dynamic (c,d)-approximately-maximal fractional matching algorithm whose update time $T(n,m)$ and which changes at most $C(n,m)$ edge weights per update, for some $c \geq 1$, $d \geq \frac{6c\cdot\ln(\max\{n,1/\epsilon\})}{\epsilon^2}$, with $\epsilon \leq \frac{1}{2}$. Then, there exists a randomized fully-dynamic* integral $2c(1+O(\epsilon))$*-approximate matching algorithm $\mathcal{A}'$ (with this bound holding both w.h.p and in expectation) with update time $T(n,m)+O(C(n,m)\cdot d/\epsilon^2)$. Moreover, if $T(n,m)$ and $C(n,m)$ are worst-case bounds, so is the update time of Algorithm $\mathcal{A}'$.*

Proof. Our reduction works as follows. Whenever an edge e is added/removed from G, we update the (c,d)-approximately-maximal fractional matching, using algorithm $\mathcal{A}$, in time $T(n,m)$. We then sample each of the at most $C(n,m)$ edges e whose value is changed, independently, with probability given by Equation (3). To control the maximum degree in the sampled subgraph H', every vertex v maintains a list of at most $(1+\epsilon)\cdot d$ sampled edges "allowable" for use in H. (This list can be maintained dynamically in $O(1)$ time per update in H' in a straightforward manner.) We let H be the graph induced by the sampled edges "allowed" by both their endpoints. Finally, we use a $(1+\epsilon)$-matching algorithm as in Lemma 2.5 to maintain a matching in the sampled subgraph H.

By Lemma 3.1 the subgraph H is a $(c(1+O(\epsilon)),(1+\epsilon)d)$-kernel of G with high probability (note that by the same lemma, all sampled edges of H' will appear in our H). By Lemma 2.4, this means that with high probability this kernel H has matching number at least

$$\mu(H) \geq \frac{1}{2c(1+O(\epsilon))(1+1/d))}\cdot\mu(G) \geq \frac{1}{2c(1+O(\epsilon))}\cdot\mu(G),$$

where the second inequality follows from $d \geq \frac{6c\cdot\ln(\max\{n,1/\epsilon\})}{\epsilon^2} \geq \frac{1}{\epsilon}$. As the above lower bound on $\mu(H)$ holds with probability at least $1-2\epsilon$ by Lemma 3.1 and Lemma 2.4, we have that $\mathbb{E}[\mu(H)] \geq (1-2\epsilon)\cdot\frac{1}{2c(1+O(\epsilon))}\cdot\mu(G) \geq \frac{1}{2c(1+O(\epsilon))}\cdot\mu(G)$. Therefore, a $(1+\epsilon)$-approximate matching in H is itself a $2c(1+O(\epsilon))$-approximate matching in G w.h.p and in expectation. Now, each of the $C(n,m)$ changes to edge weights of the fractional matching incurs at most three updates to the kernel H: for every edge e whose weight changes, this edge can be added/removed to/from H if it is sampled in/out; in the latter case both of e's endpoints can have a new edge added to their "allowable" edge list in place of e, and therefore possibly added to H, in case the endpoints had less than $(1+\epsilon)d$ sampled edges. In either case, the number of edges added to H is at most a constant per edge weight update. But on the other hand, the $(1+\epsilon)$-approximate matching algorithm implied by Lemma 2.5 requires $O(d/\epsilon^2)$ worst-case time per update in H, by H's worst-case degree bound. Consequently, our algorithm maintains a $2c(1+O(\epsilon))$-approximate integral matching (w.h.p and in expectation) in $T(n,m)+O(C(n,m)\cdot d/\epsilon^2)$ update time; moreover, this update time is worst case if the bounds on $T(n,m)$ and $C(n,m)$ are themselves worst case. $\square$

5 Applications to Maximum Weight Matching

In this section we highlight the consequences of our results for fully-dynamic maximum weight matching. First, we discuss a new reduction of Stubbs and Vassilevska Williams [50].

Lemma 5.1 ([50]). *Let $\mathcal{A}$ be an fully-dynamic α-approximate maximum cardinality matching algorithm with update time $T(n,m)$. Then, there exists a fully-dynamic $2\alpha(1+\epsilon)$-approximate maximum cardinality matching algorithm with update time $O(T(n,m)\cdot\frac{\log(n/\epsilon)}{\epsilon})$. Furthermore, if Algorithm $\mathcal{A}$ is deterministic, so is the new one, and if Algorithm $\mathcal{A}$'s update time is worst case, so is the new algorithm's update time.*

This reduction (which we elaborate on shortly), together with the state of the art dynamic maximum matching algorithms, implies most of the best currently best bounds for dynamic maximum weight matching, in Table 2 below.

Approx.	Update Time	det.	w.c.	reference
$4+\epsilon$	$O(\log(n/\epsilon)/\epsilon)$	✗	✗	5.1 + Solomon (FOCS '16) [49]
$4+\epsilon$	$\text{poly}(\log n, 1/\epsilon)$	✓	✗	5.1 + Bhattacharya et al. (STOC '16) [13]
$4+\epsilon$	$O(m^{1/3}\log(n/\epsilon)/\epsilon^3)$	✓	✓	5.1 + Bhattacharya et al. (SODA '15) [12]
$4+\epsilon$	$\text{poly}(\log n, 1/\epsilon)$	✗	✓	5.1 + **This work**
$\approx 3.009+\epsilon$	$O(\sqrt{m}\log C\epsilon^{-3})$	✓	✓	Gupta and Peng (FOCS '13) [27]
$3+\epsilon$	$O(\sqrt[4]{m}\log(n/\epsilon)\epsilon^{-3})$	✓	✗	5.1 + Bernstein and Stein (SODA '16) [11]
$2+\epsilon$	$O(\sqrt{m}\cdot\frac{\log^2(n/\epsilon)}{\epsilon^4})$	✓	✓	5.1 + Gupta and Peng (FOCS '13) [27]
$1+\epsilon$	$O(\sqrt{m}C\epsilon^{-3})$	✓	✓	Gupta and Peng (FOCS '13) [27]
$1+\epsilon$	$O(\sqrt{m}\log C\epsilon^{-O(1/\epsilon)})$	✓	✓	Gupta and Peng (FOCS '13) [27]

Table 2: Our Results and Previous State-of-the-Art for Fully-Dynamic MWM.

A somewhat more involved and worse update time bound than that given in Lemma 5.1 was presented in [50], as that paper's authors sought to obtain a persistent matching, in a sense that this matching should not change completely after a single step (i.e., no more than $O(T(n,m))$ changes to the matching per edge update, if $T(n,m)$ is the algorithm's update time). However, a simpler and more efficient reduction yielding a non-persistent matching algorithm with the performance guarantees of Lemma 5.1 is implied immediately from the driving observation of Stubbs and Vassilevska Williams [50] (and indeed, is discussed in [50]). This observation, previously made by Crouch and Stubbs [18] in the streaming setting, is as follows: denote by E_i the edges of weights in the range $((1+\epsilon)^i, (1+\epsilon)^{i+1}]$, and let M_i be an α-approximate matching in $G[E_i]$. Then, greedily constructing a matching by adding edges from each M_i in decreasing order of i yields a $2\alpha(1+\epsilon)$-approximate maximum weight matching. Adding to this observation the observation that if we are content with a $(1+\epsilon)$-approximate (or worse) maximum weight matching we may safely ignore all edges of weight less than ϵ/n of the maximum edge weight (a trivial lower bound on the maximum weight matching's weight), we find that we can focus on the ranges $((n/\epsilon)^i, (n/\epsilon)^{i+2}]$, for some $i \in \mathbb{Z}$, noting that each edge belongs to at most two such ranges.

In each such range $((n/\epsilon)^i, (n/\epsilon)^{i+2}]$, the argument of [18, 50] implies that maintaining α-approximate matchings in the sub-ranges $((1+\epsilon)^j, (1+\epsilon)^{j+1}]$ for integral ranges and combining these greedily result in a $2\alpha(1+\epsilon)$-approximate maximum weight matching in the range $((n/\epsilon)^i, (n/\epsilon)^{i+2}]$. Therefore, in the range containing a $(1+\epsilon)$-approximate MWM (such a range exists, by the above), this approach maintains a $2\alpha(1+O(\epsilon))$-approximate MWM. The only possible difficulty is combining these matchings greedily *dynamically.* This is relatively straightforward to do in $O(\log(n/\epsilon)/\epsilon)$ worst-case time per change of the α-approximate matching algorithm, however, implying the bound of Lemma 5.1.

As seen in Table 2, this reduction of Stubbs and Vassilevska Williams [50] implies a slew of improved bounds for fully-dynamic approximate maximum weight matching. Plugging in our bounds of Theorem 1.4 for fully-dynamic maximum matching into the reduction of Lemma 5.1 similarly yields the first constant-approximate maximum weight matching with polylogarithmic *worst-case* update time, given in Theorem 1.5 below.

Theorem 1.5. *For all $\epsilon \leq \frac{1}{2}$, there exists a randomized fully-dynamic $(4 + O(\epsilon))$-approximate maximum weight matching algorithm with poly$(\log n, 1/\epsilon)$ worst-case update time. The approximation guarantee holds with high probability and in expectation.*

6 Conclusion and Future Work

In this work we presented a simple randomized reduction from $(2c + \epsilon)$-approximate fully-dynamic matching to fully-dynamic (c, d)-approximately-maximal fractional matching with a slowdown of d. Using the recent algorithm of Bhattacharya et al. [14], our work yields the first fully-dynamic matching algorithms with faster-than-polynomial worst-case update time for any constant approximation ratio; specifically, it yields a $(2 + O(\epsilon))$-approximate matching with polylog worst case update time. Our work raises several natural questions and future research directions to explore.

Faster Fractional Algorithms. Given our reduction, in order to obtain faster $(2 + \epsilon)$-approximate matching algorithms, it would suffice to improve the update time for fully-dynamic $(1 + \epsilon, O_\epsilon(\log n))$-approximately-maximal fractional matching algorithm compared to the algorithm of Bhattacharya et al. [14]. Large enough improvements in the update time, number of edge weight changes and parameter d of $(1 + \epsilon, d)$-approximately-maximal fractional matching algorithms used would result in even better bounds. We note however that our current approach does not seem to allow for sub-logarithmic update bounds, due to its update time's dependence on $d \geq \log n$, so obtaining worst-case sub-logarithmic bounds may require different ideas.

More Efficient/Deterministic Reduction. Given the above, one may ask whether the dependence on d may be removed in a black-box reduction such as ours, yielding randomized integral matching algorithms with the same running time as their fractional counterparts. One may further wonder whether or not one can obtain a *deterministic* counterpart to our black-box reduction from integral matching to approximately-maximal fractional matching. Such a reduction with polylogarithmic overhead would yield for example a deterministic $(2+\epsilon)$-approximate algorithm with worst-case polylogarithmic update time.

Maximal Matching. Finally, a natural question from our work and prior work is whether or not a *maximal* matching can be maintained in worst-case polylogarithmic time (also implying a 2-approximate minimum vertex cover within the same time bounds). We leave this as a tantalizing open question.

Acknowledgments.

We thank Monika Henzinger for sharing a preprint of [14] with us, and Virginia Vassilevska Williams for sharing a preprint of [50] with us. The fifth author wishes to thank Seffi Naor for a remark which inspired Definition 1.1.

References

[1] Abboud, A. and Vassilevska Williams, V. 2014. Popular conjectures imply strong lower bounds for dynamic problems. In *Proceedings of the 55th Symposium on Foundations of Computer Science (FOCS)*. 434–443.

[2] Abraham, I. and Chechik, S. 2013. Dynamic decremental approximate distance oracles with $(1 + \epsilon, 2)$ stretch. *CoRR abs/1307.1516*.

[3] Abraham, I., Chechik, S., Delling, D., Goldberg, A. V., and Werneck, R. F. 2016. On dynamic approximate shortest paths for planar graphs with worst-case costs. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 740–753.

[4] Abraham, I., Chechik, S., and Krinninger, S. 2017. Fully dynamic all-pairs shortest paths with worst-case update-time revisited. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 440–452.

[5] Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. 1993. *Network flows - theory, algorithms, and applications*. Prentice hall.

[6] Ausiello, G., Italiano, G. F., Marchetti-Spaccamela, A., and Nanni, U. 1991. Incremental algorithms for minimal length paths. *Journal of Algorithms 12,* 4, 615–638.

[7] Baswana, S., Gupta, M., and Sen, S. 2011. Fully dynamic maximal matching in $O(\log n)$ update time. In *Proceedings of the 52nd Symposium on Foundations of Computer Science (FOCS)*. 383–392.

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

[9] Baswana, S., Hariharan, R., and Sen, S. 2007. Improved decremental algorithms for maintaining transitive closure and all-pairs shortest paths. *Journal of Algorithms 62,* 2, 74–92.

[10] Bernstein, A. and Stein, C. 2015. Fully dynamic matching in bipartite graphs. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming (ICALP)*. 167–179.

[11] Bernstein, A. and Stein, C. 2016. Faster fully dynamic matchings with small approximation ratios. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 692–711.

[12] Bhattacharya, S., Henzinger, M., and Italiano, G. F. 2015. Deterministic fully dynamic data structures for vertex cover and matching. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 785–804.

[13] Bhattacharya, S., Henzinger, M., and Nanongkai, D. 2016. New deterministic approximation algorithms for fully dynamic matching. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing (STOC)*. 398–411.

[14] Bhattacharya, S., Henzinger, M., and Nanongkai, D. 2017a. Fully dynamic approximate maximum matching and minimum vertex cover in $O(\log^3 n)$ worst case update time. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 470–489.

[15] Bhattacharya, S., Henzinger, M., and Nanongkai, D. 2017b. Fully dynamic approximate maximum matching and minimum vertex cover in $O(\log^3 n)$ worst case update time. *CoRR abs/1704.02844*.

[16] Charikar, M. and Solomon, S. 2018. Fully dynamic almost-maximal matching: Breaking the polynomial barrier for worst-case time bounds. In *Proceedings of the 45th International Colloquium on Automata, Languages and Programming (ICALP)*.

[17] Cohen, M. B., Mądry, A., Sankowski, P., and Vladu, A. 2017. Negative-weight shortest paths and unit capacity minimum cost flow in $\tilde{O}(m^{10/7} \log W)$ time. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 752–771.

[18] Crouch, M. and Stubbs, D. M. 2014. Improved streaming algorithms for weighted matching, via unweighted matching. In *Proceedings of the 17th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*. 96.

[19] Demetrescu, C. and Italiano, G. F. 2004. A new approach to dynamic all pairs shortest paths. *Journal of the ACM (JACM) 51,* 6, 968–992.

[20] Demetrescu, C. and Italiano, G. F. 2006. Fully dynamic all pairs shortest paths with real edge weights. *Journal of Computer and System Sciences 72,* 5, 813–837.

[21] Duan, R. and Pettie, S. 2014. Linear-time approximation for maximum weight matching. *Journal of the ACM (JACM) 61,* 1, 1.

[22] Edmonds, J. and Karp, R. M. 1972. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM) 19,* 2, 248–264.

[23] Fredman, M. L. and Tarjan, R. E. 1987. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM) 34,* 3, 596–615.

[24] Gabow, H. N. 1990. Data structures for weighted matching and nearest common ancestors with linking. In *Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 434–443.

[25] Gabow, H. N. and Tarjan, R. E. 1989. Faster scaling algorithms for network problems. *SIAM Journal on Computing (SICOMP) 18,* 5, 1013–1036.

[26] Grandoni, F. and Williams, V. V. 2012. Improved distance sensitivity oracles via fast single-source replacement paths. In *Proceedings of the 53rd Symposium on Foundations of Computer Science (FOCS)*. 748–757.

[27] Gupta, M. and Peng, R. 2013. Fully dynamic $(1+\epsilon)$-approximate matchings. In *Proceedings of the 54th Symposium on Foundations of Computer Science (FOCS)*. 548–557.

[28] Henzinger, M., Krinninger, S., and Nanongkai, D. 2013. Dynamic approximate all-pairs shortest paths: Breaking the o(mn) barrier and derandomization. In *Proceedings of the 54th Symposium on Foundations of Computer Science (FOCS)*. 538–547.

[29] Henzinger, M., Krinninger, S., and Nanongkai, D. 2014a. Decremental single-source shortest paths on undirected graphs in near-linear total update time. In *Proceedings of the 55th Symposium on Foundations of Computer Science (FOCS)*. 146–155.

[30] Henzinger, M., Krinninger, S., and Nanongkai, D. 2014b. Sublinear-time decremental algorithms for single-source reachability and shortest paths on directed graphs. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC)*. 674–683.

[31] Henzinger, M., Krinninger, S., and Nanongkai, D. 2014c. A subquadratic-time algorithm for decremental single-source shortest paths. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 1053–1072.

[32] Henzinger, M., Krinninger, S., and Nanongkai, D. 2015. Improved algorithms for decremental single-source reachability on directed graphs. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming (ICALP)*. 725–736.

[33] Henzinger, M. R. and King, V. 2001. Maintaining minimum spanning forests in dynamic graphs. *SIAM Journal on Computing (SICOMP) 31,* 2, 364–374.

[34] Hopcroft, J. E. and Karp, R. M. 1971. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. In *Proceedings of the 12th Annual Symposium on Switching and Automata Theory (SWAT)*. 122–125.

[35] Ivkovic, Z. and Lloyd, E. L. 1993. Fully dynamic maintenance of vertex cover. In *Proceedings of the 19th International Workshop on Graph-Theoretic Concepts in Computer Science*. 99–111.

[36] King, V. 1999. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *Proceedings of the 40th Symposium on Foundations of Computer Science (FOCS)*. 81–91.

[37] Kopelowitz, T., Pettie, S., and Porat, E. 2016. Higher lower bounds from the 3sum conjecture. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 1272–1287.

[38] Lovász, L. and Plummer, M. D. 2009. *Matching theory.* Vol. 367. American Mathematical Society.

[39] Madry, A. 2013. Navigating central path with electrical flows: From flows to matchings, and back. In *Proceedings of the 54th Symposium on Foundations of Computer Science (FOCS)*. 253–262.

[40] Micali, S. and Vazirani, V. V. 1980. An $O(\sqrt{|V|}|E|)$ algoithm for finding maximum matching in general graphs. In *Proceedings of the 21st Symposium on Foundations of Computer Science (FOCS)*. 17–27.

[41] Mucha, M. and Sankowski, P. 2004. Maximum matchings via gaussian elimination. In *Proceedings of the 45th Symposium on Foundations of Computer Science (FOCS)*. 248–255.

[42] Neiman, O. and Solomon, S. 2013. Simple deterministic algorithms for fully dynamic maximal matching. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC)*. 745–754.

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

[44] Onak, K. and Rubinfeld, R. 2010. Maintaining a large matching and a small vertex cover. In *Proceedings of the 42nd Annual ACM Symposium on Theory of Computing (STOC)*. 457–464.

[45] Peleg, D. and Solomon, S. 2016. Dynamic $(1 + \epsilon)$-approximate matchings: a density-sensitive approach. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 712–729.

[46] Pettie, S. 2012. A simple reduction from maximum weight matching to maximum cardinality matching. *Information Processing Letters (IPL) 112,* 23, 893–898.

[47] SANKOWSKI, P. Faster dynamic matchings and vertex connectivity. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*.

[48] SANKOWSKI, P. 2009. Maximum weight bipartite matching in matrix multiplication time. *Theoretical Computer Science (TCS) 410,* 44, 4480–4488.

[49] SOLOMON, S. 2016. Fully dynamic maximal matching in constant update time. In *Proceedings of the 57th Symposium on Foundations of Computer Science (FOCS)*. 325–334.

[50] STUBBS, D. AND VASSILEVSKA WILLIAMS, V. 2017. Metatheorems for dynamic weighted matching. In *Proceedings of the 8th Innovations in Theoretical Computer Science Conference (ITCS)*.

[51] THORUP, M. 2004. Fully-dynamic all-pairs shortest paths: Faster and allowing negative cycles. In *Proceedings of the 9th Scandinavian Workshop on Algorithm Theory (SWAT)*. 384–396.

[52] THORUP, M. 2005. Worst-case update times for fully-dynamic all-pairs shortest paths. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*. 112–119.

[53] VIZING, V. G. 1964. On an estimate of the chromatic class of a p-graph. *Diskret analiz 3*, 25–30.

[54] WILLIAMS, V. V. 2012. Multiplying matrices faster than coppersmith-winograd. In *Proceedings of the 44th Annual ACM Symposium on Theory of Computing (STOC)*. 887–898.

Appendix

A Properties of the Nice Partition of Bhattacharya et al. [14]

In this Section we justify the use of the fully-dynamic algorithm of [14] for maintaining a nice partition as per Definition 2.1, where the worst-case update time for inserting or deleting an edge is $O(\log^3 n)$ for a fixed constant $\epsilon > 0$. Our goal here is twofold: first, to claim that the number of edge weight changes in each update operation is bounded by $O(\beta^2 \log n)$ in the worst case and second, that the update time is $O(\beta^7 \log^3 n)$ in the worst case. Although the worst-case update time in [14] is $O(\log^3 n)$ (ignoring β factors), the number of changes of the edge weights during an update could be much larger, potentially polynomial in n. This can happen if, for example, changes of edge weights are maintained implicitly in an aggregated or lazy fashion. Specifically, perhaps the changes of edge weights are implicitly maintained by vertices changing their levels during an update call, so that a small change in the level of a vertex hides a polynomial number of changes of edge weights. Fortunately, this is not the case in [14], as we assert in the following two theorems, justifying our use of the result of [14] for dynamically maintaining a nice partition. We note that this is crucial for our needs, as the runtime of our algorithm depends on the number of edge weight changes.

First of all, it is easy to observe that the number of changes to edge weights is bounded by the worst-case runtime of an update procedure which is $O(\log^3 n)$ for a fixed constant $\epsilon > 0$. According to Section 3.4 in [14], every edge $(x, y) \in E$ maintains the values of its weight $w(x, y)$ and level $\ell(x, y)$, and thus it is straightforward that there are at most $O(\log^3 n)$ changes of edge weights per update, for constant $\epsilon > 0$. We further prove in Lemma A.1 that the number of changes of edge weights per insert/delete is actually bounded by $O(\log n)$ per insert/delete for a constant $\epsilon > 0$.

Lemma A.1. *The algorithm in [14] dynamically maintains a nice partition while edges are inserted and deleted from the graph. Let $\{w_e\}_{e \in E}$ be the edge weights just before an update (edge insertion or deletion), and let $\{w'_e\}_{e \in E}$ be the edge weights just after the update. Then the number of changes in edge weights (i.e., the number of edges $e \in E$ such that $w_e \neq w'_e$) is bounded by $O(\beta^2 \log n)$.*

Proof. The dynamic nice partition is maintained in [14] as follows: before and after an edge insertion/deletion each vertex is a "clean" vertex in exactly one of six states **UP,DOWN,UP-B,DOWN-B,SLACK and IDLE**. Each has several constraints associated with it, and it is immediate that if the constraints of the state of every vertex hold then the resulting partition and assigned levels and edge weights form a nice-partition as per Definition 2.1 (see [14, Lemma 3.2]).

Each insertion (or deletion) of an edge to (from) the graph increases (resp. decreases) the weight associated with each vertex touching the edge. The insertion/deletion of an edge may cause an endpoint x of the edge to be marked as *dirty* based on a set of rules (rules 3.1-3.3 in [14]). Intuitively, a dirty vertex requires some fixing; for example, if a vertex x has a large weight W_x very close to 1 then an insertion of an edge touching x further increases the weight W_x making it even further closer to 1 (or even larger than 1). To preserve the constraints of all the states, this requires fixing the dirty vertex, e.g., by increasing the shadow level of x with respect to some edge (x, y) touching it, which reduces the weight the edge (x, y) and thus reduces the weight W_x of the vertex x. As a side effect of reducing the weight of (x, y), also the weight W_y of the vertex y reduces, possibly causing y to become a dirty vertex which requires some fixing.

The above description forms a chain of *activations* (a vertex x is said to be *activated* if its weight W_x changes). Assume that the inserted/deleted edge touches a vertex x_1 and thus x_1 becomes activated as it has its weight W_{x_1} changed. It may happen (according to rules 3.1-3.3 in [14]) that due to the change in W_{x_1} the vertex x_1 becomes dirty and this requires fixing x_1 by calling the procedure **FIX-DIRTY-NODE(x_1)**. The procedure **FIX-DIRTY-NODE(x_1)** fixes the vertex x_1 so it becomes clean again and by that guarantees that it satisfies its state's constraints.

During **FIX-DIRTY-NODE(x_1)** either none of the edges changed their weights (and thus the chain of activations terminates), or exactly one edge (x_1, x_2) has its weight changed. In the latter case, we say that x_2 incurs an induced activation, that may cause x_2 to become dirty as per rules 3.1-3.3 in [14]. In this case, the chain of activations continues as we now call **FIX-DIRTY-NODE(x_2)**, which in turn may cause at most one edge (x_2, x_3) to change its weight, and then vertex x_3 occurs an induced activation. The chain of activations continues, and as long as there is a dirty vertex x_i the procedure **FIX-DIRTY-NODE(x_i)** is called, restoring x_i to clean status but potentially changing the weight of at most one edge (x_i, x_{i+1}), which may cause x_{i+1} to become a dirty vertex, an so on. However, as stated in the conference version in [14, Theorem 6.2] and proved in the full version (see [15, Theorem 6.4 and Lemma 8.1]), the chain of activations comprises at most $O(\log_\beta n) = O(\log n)$ activations; i.e., there are at most $O(\log n)$ calls to **FIX-DIRTY-NODE(x)** in the chain until all vertices are clean. Furthermore, according to Assumptions 1 and 2 in [14, Section 6], an insertion or deletion of an edge may cause at most $O(\beta^2)$ chains of activations, and thus following an edge insertion or deletion there are at most $O(\beta^2 \log n)$ calls to **FIX-DIRTY-NODE(x)**.

Observe that **FIX-DIRTY-NODE** is the only procedure that may cause a change in the weight of an edge (except for the insertion/deletion of the edge itself), and each call to **FIX-DIRTY-NODE** changes the weight of at most one edge. Thus, we conclude that the number of changes of edge weights during an insertion/deletion of an edge is at most the number of calls to **FIX-DIRTY-NODE**, and thus the number of changes of edge weights is bounded by $O(\beta^2 \log n)$. We mention that the level of a node may change due to the call of the subroutine **UPDATE-STATUS(x)** (e.g., **Cases 2-a and 2-b**), but this does not cause a change in the weights of edges incident on x. □

Finally, we furthermore state the worst-case update time of an update for maintaining the nice partition. While [14] proves that the worst-case update time is $O(\log^3 n)$ for constant $\beta \geq 5$, we also mention the dependency of the update time in β which is implicit in [14].

Lemma A.2. *The algorithm in [14] dynamically maintains a nice partition in $O(\beta^7 \log^3 n)$ worst-case update time.*

Proof. The worst-case update time is bounded by the number of calls to **FIX-DIRTY-NODE**(x) times the runtime of **FIX-DIRTY-NODE**(x). As mentioned in the proof of Theorem A.1, each edge insertion/deletion may invoke $O(\beta^2 \log n)$ calls to **FIX-DIRTY-NODE**(x), and the runtime of each **FIX-DIRTY-NODE**(x) is bounded by $O(\beta^5 \log^2 n)$, since the runtime of **FIX-DIRTY-NODE**(x) in the worst-case is dominated by the runtime of the procedure **FIX-DOWN**(x) which takes $O(\beta^5 \log^2 n)$ according to [14, Lemma 5.1 and Lemma 5.4]. Thus, the worst-cast update time of an insertion/deletion of an edge is $O(\beta^7 \log^3 n)$. □

Combining Lemma A.2 and Lemma A.1 we immediately obtain Lemma 1.3, restated below.

Lemma 1.3 ([14])**.** *For all $\epsilon \leq \frac{1}{2}$, there is a fully-dynamic $(1 + 2\epsilon, \max\{54 \log n/\epsilon^3, (3/\epsilon)^{21}\})$-approximately-maximal fractional matching algorithm with $T(n, m) = O(\log^3 n/\epsilon^7)$ worst-case update time, using at most $C(n, m) = O(\log n/\epsilon^2)$ edge weight changes per update in the worst case.*