Near-Optimal Dynamic Rounding of Fractional Matchings in Bipartite Graphs[*]

Sayan Bhattacharya
University of Warwick
UK

Peter Kiss
University of Warwick
UK

Aaron Sidford
Stanford University
USA

David Wajc
Technion — Israel Institute of Technology
Israel

ABSTRACT

We study dynamic $(1-\epsilon)$-approximate rounding of fractional matchings—a key ingredient in numerous breakthroughs in the dynamic graph algorithms literature. Our first contribution is a surprisingly simple deterministic rounding algorithm in bipartite graphs with amortized update time $O(\epsilon^{-1}\log^2(\epsilon^{-1}\cdot n))$, matching an (unconditional) recourse lower bound of $\Omega(\epsilon^{-1})$ up to logarithmic factors. Moreover, this algorithm's update time improves provided the minimum (non-zero) weight in the fractional matching is lower bounded throughout. Combining this algorithm with novel dynamic *partial rounding* algorithms to increase this minimum weight, we obtain a number of algorithms that improve this dependence on n. For example, we give a high-probability randomized algorithm with $\tilde{O}(\epsilon^{-1}\cdot(\log\log n)^2)$-update time against adaptive adversaries.

Using our rounding algorithms, we also round known $(1-\epsilon)$-decremental fractional bipartite matching algorithms with no asymptotic overhead, thus improving on state-of-the-art algorithms for the decremental bipartite matching problem. Further, we provide extensions of our results to general graphs and to maintaining almost-maximal matchings.

CCS CONCEPTS

• **Theory of computation → Dynamic graph algorithms**.

KEYWORDS

Dynamic Matching, Dynamic Algorithms, Data Structures

ACM Reference Format:
Sayan Bhattacharya, Peter Kiss, Aaron Sidford, and David Wajc. 2024. Near-Optimal Dynamic Rounding of Fractional Matchings in Bipartite Graphs. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing (STOC '24), June 24–28, 2024, Vancouver, BC, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3618260.3649648

[*]Full version available at https://arxiv.org/abs/2306.11828 [25].

STOC '24, June 24–28, 2024, Vancouver, BC, Canada
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0383-6/24/06
https://doi.org/10.1145/3618260.3649648

1 INTRODUCTION

Dynamic matching is one of the most central and well-studied dynamic algorithm problems. Here, a graph undergoes edge insertions and deletions, and we wish to quickly update a large matching (vertex-disjoint set of edges) following each such change to the graph.

A cornerstone of numerous dynamic matching results is the dynamic *relax-and-round* approach: the combination of dynamic *fractional* matching algorithms [17–20, 26] with dynamic *rounding* algorithms [3, 21, 24, 45, 55]. This dynamic fractional matching problem asks to maintain a vector $\mathbf{x}\in\mathbb{R}^E_{\geq 0}$ such that $x(v):=\sum_{e\ni v}x_e$ satisfies the fractional degree constraint $x(v)\leq 1$ for all vertices $v\in V$ and $\|\mathbf{x}\|:=\sum_e x_e$ is large compared to the size of the largest (fractional) matching in the dynamic graph $G=(V,E)$. The goal typically is to solve this problem while minimizing the amortized or worst-case time per edge update in G.[1] For the rounding problem (the focus of this work), an abstract interface can be defined as follows.

Definition 1.1. *A dynamic rounding algorithm (for fractional matchings) is a data structure supporting the following operations:*

- `init`$(G=(V,E),\ \mathbf{x}\in\mathbb{R}^E_{\geq 0},\ \epsilon\in(0,1))$*: initializes the data structure for undirected graph G with vertices V and edges E, current fractional matching $\mathbf{x}$ in G, and target error ϵ.*
- `update`$(e\in E,\ \nu\in[0,1])$*: sets $x_e\leftarrow\nu$ under the promise that the resulting $\mathbf{x}$ is a fractional matching in G.*[2]

The algorithm must maintain a matching M in the support of $\mathbf{x}$, $\text{supp}(\mathbf{x}):=\{e\in E\mid x_e>0\}$*, such that M is a $(1-\epsilon)$-approximation with respect to $\|\mathbf{x}\|:=\sum_e x_e$, i.e.*

$$M\subseteq\text{supp}(\mathbf{x})\ ,\ M \text{ is a matching }\ ,\text{ and } |M|\geq(1-\epsilon)\cdot\|\mathbf{x}\|\,.$$

The combination of fast fractional algorithms with fast dynamic rounding algorithms plays a key role in state-of-the-art time / approximation trade-offs for the dynamic matching problem against an adaptive adversary [21, 45, 55], including the recent breakthroughs of [8, 24]. Here, a randomized algorithm *works against an adaptive adversary* (or *is adaptive*, for short) if its guarantees hold even when future updates depend on the algorithm's previous

[1]An algorithm has *amortized* update time $f(n)$ if every sequence of t updates takes at most $t\cdot f(n)$ time and has *worst-case* update time $f(n)$ if each operation takes at most $f(n)$ time. As we focus on amortized update times, we omit this distinction.
[2]Invoking `update`$(e,0)$ essentially deletes e and subsequently invoking `update`(e,ν) for $\nu>0$ essentially adds e back. So, G might as well be the complete graph on V. However, we find the notation $G=(V,E)$ convenient.

output and its internal state. Slightly weaker are *output-adaptive* algorithms, that allow updates to depend only on the algorithms' output. Note that deterministic algorithms are automatically adaptive. A major motivation to study output-adaptive dynamic algorithms is their black-box use as subroutines within other algorithms. (See discussions in, e.g., [12, 34, 47].)

Despite significant effort and success in designing and applying dynamic rounding algorithms, the update time of current $(1-\varepsilon)$-approximate dynamic rounding approaches are slower by large $\text{poly}(\varepsilon^{-1}, \log n)$ factors than an unconditional recourse (changes per update) lower bound of $\Omega(\varepsilon^{-1})$ (Fact 2.3).[3] Consequently, rounding is a computational bottleneck for the running time of many state-of-the-art dynamic matching algorithms [6, 8, 21, 24, 45, 55] and decremental (only allowing deletions) matching algorithms [14, 44].

The question thus arises, *can one design (output-adaptive) optimal dynamic rounding algorithms for fractional matching?* We answer this question in the affirmative in a strong sense.

1.1 Our Contributions

Our main results are deterministic and randomized dynamic fractional matching rounding algorithms for bipartite graphs that match the aforementioned simple recourse lower bound of $\Omega(\varepsilon^{-1})$ up to logarithmic factors in ε and (sub-)logarithmic factors in $n := |V|$. These results are summarized by the following theorem.[4]

Theorem 1.2. *The dynamic bipartite matching rounding problem admits:*

1. *A deterministic algorithm with $\tilde{O}(\varepsilon^{-1}\log n)$* `update` *time.*
2. *An adaptive algorithm with $\tilde{O}(\varepsilon^{-1}\cdot(\log\log n)^2)$* `update` *time that is correct w.h.p.*
3. *An output-adaptive algorithm with $\tilde{O}(\varepsilon^{-1})$ expected* `update` *time.*

The `init`$(G, \mathbf{x}, \varepsilon)$ *time of each of these algorithms is $O(\varepsilon\cdot|\text{supp}(\mathbf{x})|)$ times its* `update` *time.*

In contrast, prior approaches have update time at least $\Omega(\varepsilon^{-4})$ (see Section 1.2). Moreover, all previous adaptive algorithms with high probability (w.h.p.) or deterministic guarantees all have at least (poly)logarithmic dependence on n, as opposed to our (sub-)logarithmic dependence on n.

General Graphs. In general graphs, one cannot round all fractional matchings (as defined) to integrality while only incurring a $(1-\varepsilon)$ factor loss in value.[5] Nonetheless, it is known how to round "structured" $(1/2-\varepsilon)$-approximate dynamic fractional matchings [20, 26] (see full version for more details) to obtain an *integral* $(1/2-\varepsilon)$-approximate matching [3, 21, 45, 55], and even *almost maximal* matchings [24], as defined in [49] and restated below.

[3]Proving update time lower bounds for approximate dynamic matching is a notoriously challenging open problem. On the other hand, [52] show that recourse can be made $O(\varepsilon^{-1})$ for any approximate dynamic matching algorithm.

[4]Throughout, we use "soft-O" notation, $\tilde{O}$, to suppress logarithmic factors in ε, i.e., $\tilde{O}(f) = O(f\cdot\text{poly}(\log(\varepsilon^{-1})))$.

[5]Consider the triangle graph with fractional values $x_e = 1/2$ on all three edges; this fractional matching has value 3/2, while any integral matching in a triangle has size at most one. While adding additional constraints [37] avoids this issue, no dynamic fractional algorithms for the matching polytope in general graphs are currently known.

Definition 1.3. *A matching M in G is an ε-almost maximal matching (ε-AMM) if M is maximal with respect to some subgraph $G[V\setminus U]$ obtained by removing at most $|U|\leq\varepsilon\cdot\mu(G)$ vertices from G, where $\mu(G)$ is the maximum matching size in G.*

Such ε-AMM's are $(1/2-\varepsilon)$-approximate with respect to the maximum matching [49]. Moreover, (almost) maximality of ε-AMM makes their maintenance a useful algorithmic subroutine [6, 24, 49]. Extending our approach to rounding the aforementioned well-structured, "near maximal" dynamic fractional matchings in general graphs [20, 26], we obtain faster ε-AMM algorithms, as follows (see the full version of the paper for formal statement).

Theorem 1.4. *There exist dynamic algorithms maintaining ε-AMM in general graphs in update time $\tilde{O}(\varepsilon^{-3})+O(t_f+u_f\cdot t_r)$, where t_f and u_f are the update time and number of calls to* `update` *of any "structured" dynamic fractional matching algorithm, and t_r is the* `update` *time for "partial" rounding. Furthermore, there exist dynamic partial rounding algorithms with the same update times and adaptivity as those of Theorem 1.2.*

1.1.1 Applications. Applying our rounding algorithms to known fractional algorithms yields a number of new state-of-the-art dynamic matching results.

For example, by a black-box application of Theorem 1.2, we deterministically round known decremental (fractional) bipartite matching algorithms [14, 44] *with no asymptotic overhead*, yielding faster $(1-\varepsilon)$-approximate decremental bipartite matching algorithms. We also discuss how a variant of Theorem 1.4 together with the general-graph decremental algorithm of [5] leads to a conjecture regarding the first *deterministic* sub-polynomial update time $(1-\varepsilon)$-approximate decremental matching algorithm in general graphs.

Our main application is obtained by applying our rounding algorithm for general graphs of Theorem 1.4 to the $O(\varepsilon^{-2})$-time fractional matching algorithm of [26], yielding the following.

Theorem 1.5. *For any $\varepsilon > 0$, there exist dynamic ε-AMM algorithms that are:*

1. *Deterministic, using $\tilde{O}(\varepsilon^{-3}\cdot\log n)$* `update` *time.*
2. *Adaptive, using $\tilde{O}(\varepsilon^{-3}\cdot(\log\log n)^2)$* `update` *time, correct w.h.p.*
3. *Output-adaptive, using $\tilde{O}(\varepsilon^{-3})$ expected* `update` *time.*

In contrast, all prior non-oblivious $(1/2-\varepsilon)$-approximate matching algorithms had at least quartic dependence on ε, which the above result improves to cubic. Moreover, this result yields the first deterministic $O(\log n)$-time and adaptive $o(\log n)$-time high-probability algorithms for this widely-studied approximation range and for near-maximal matchings. This nearly concludes a long line of work on deterministic/adaptive dynamic matching algorithms for the $(1/2-\varepsilon)$ approximation regime [9, 18, 19, 21, 24, 26, 45, 55].

1.2 Our Approach in a Nutshell

Here we outline our approach, focusing on the key ideas behind Theorem 1.2. To better contrast our techniques with those of prior work, we start by briefly overviewing the latter.

Previous approaches. Prior dynamic rounding algorithms [3, 21, 45, 55] all broadly work by partially rounding the fractional matching $\mathbf{x}$ to obtain a matching sparsifier S (a sparse subgraph approximately preserving the fractional matching size compared to $\mathbf{x}$). Then, they periodically compute a $(1-\varepsilon)$-approximate matching in this sparsifier S using a static $\tilde{O}(|S| \cdot \varepsilon^{-1})$-time algorithm (e.g., [36]) whenever $\|\mathbf{x}\|$ changes by $\varepsilon \cdot \|\mathbf{x}\|$, i.e., every $\Omega(\varepsilon \cdot \|\mathbf{x}\|)$ updates. This period length guarantees that the matching computed remains a good approximation of the current fractional matching during the period, with as good an approximation ratio as the sparsifier S. Now, for sparsifier S to be $O(1)$-approximate, it must have size $|S| = \Omega(\|\mathbf{x}\|)$, and so this approach results in an update time of at least $\Omega(\varepsilon^{-2})$. Known dynamic partial rounding approaches all result in even larger sparsifiers, resulting in large $\text{poly}(\varepsilon^{-1}, \log n)$ update times.

Direct to integrality. Our first rounding algorithm for bipartite graphs breaks from this framework and directly rounds to integrality. This avoids overhead of periodic recomputation of static near-maximum matching algorithms, necessary to avoid super-linear-in-ε^{-1} update time (or $n^{o(1)}$ factors, if we substitute the static approximate algorithms with the breakthrough near-linear-time max-flow algorithm of [33]). The key idea is that, by encoding each edge's weight in binary, we can round the fractional matching "bit-by-bit", deciding for each edge whether to round a component of value 2^{-i} to a component of value 2^{-i+1}. This can be done statically in near-linear-time by variants of standard degree splitting algorithms, decreasing the degree of each node in a multigraph by a factor of two (see Theorem 2). Letting $L := \log((\min_{e:x_e \neq 0} x_e)^{-1})$, we show that by buffering updates of total value at most $O(\varepsilon \cdot \|\mathbf{x}\|/L)$ for each power of 2, we can efficiently dynamize this approach, obtaining a dynamic rounding algorithm with update time $\tilde{O}(\varepsilon^{-1} \cdot L^2)$. As we can assume that $\min_{e:x_e \neq 0} x_e \geq \varepsilon/n^2$ (Observation 2.2), this gives our bipartite $\tilde{O}(\varepsilon^{-1} \cdot \log^2 n)$ time algorithm.

Faster partial rounding. The second ingredient needed for Theorem 1.2 are a number of algorithms for "partially rounding" fractional matchings, increasing $\min_{e:x_e \neq 0} x_e$ while approximately preserving the value of the fractional matching. (The output is not quite a fractional matching, but in a sense is close to one. See Definition 4.1.) Our partial rounding algorithms draw on a number of techniques, including fast algorithms for partitioning a fractional matching's support into *multiple* sparsifiers, as opposed to a single such sparsifier in prior work, and a new output-adaptive sampling data structure of possible independent interest (Section A).[6,7] Combining these partial rounding algorithms with our simple algorithm underlies all our bipartite rounding results of Theorem 1.2, as well as our general graph rounding results (which are deferred to the full version of the paper).

1.3 Related Work

The dynamic matching literature is vast, and so we only briefly discuss it here. For a more detailed discussion, see, e.g., the recent papers [4, 8, 22, 24].

[6]From this we derive the first output-adaptive matching algorithm that is not also adaptive.

[7]Concurrently to our work, another such sampling algorithm was devised [56]. See discussion in Section A.

The dynamic matching problem has been intensely studied since a seminal paper of Onak and Rubinfeld [48], which showed how to maintain a constant-approximate matching in polylogarithmic time. Results followed in quick succession, including conditional polynomial update time lower bounds for exact maximum matching size [1, 2, 35, 42, 46], and numerous algorithmic results, broadly characterized into two categories: polynomial time/approximation tradeoffs [4, 8, 10, 11, 15, 16, 21, 22, 28, 39, 40, 43, 45, 49, 50, 55], and $1/2-$ or $(1/2-\varepsilon)$-approximate algorithms with polylogarithmic or even constant update time [3, 7, 9, 13, 19, 21, 26, 31, 32, 45, 53, 55].[8] We improve the state-of-the-art update times for all deterministic and adaptive algorithms in the intensely-studied second category.

The $(1-\varepsilon)$-approximate matching problem has also been studied in *partially dynamic* settings. This includes a recent algorithm supporting vertex updates on opposite sides of a bipartite graph, though not edge updates [57] (see arXiv). For incremental (edge-insertion-only) settings several algorithms are known [23, 27, 38, 41], the fastest having $\text{poly}(\varepsilon^{-1})$ update time [27]. In decremental settings (edge-deletion-only), rounding-based algorithms with $\text{poly}(\varepsilon^{-1}, \log n)$ update time in bipartite graphs [14, 23, 44] and randomized $\exp(\varepsilon^{-1})$ in general graphs [5] are known. We improve on these decremental results, speeding up bipartite matching, and giving the first deterministic logarithmic-time algorithm for general graphs.

1.4 Paper Outline

Following some preliminaries in Section 2, we provide our first simple bipartite rounding algorithm in Section 3. In the full version of the paper we introduce the notion of partial roundings that we study and show how such partial rounding algorithms can be combined with our simple algorithm to obtain the (bipartite) rounding algorithms of Theorem 1.2.

2 PRELIMINARIES

Assumptions and Model. Throughout, we assume that $\|\mathbf{x}\| \geq 1$, as otherwise it is trivial to round $\|\mathbf{x}\|$ within a factor of $1-\varepsilon$, by maintaining a pointer to any edge in $\text{supp}(\mathbf{x})$ whenever the latter is not empty. In this paper we work in the word RAM model of computation with words of size $w := \Theta(\log n)$, allowing us to index any of $2^{O(w)} = \text{poly}(n)$ memory addresses, perform arithmetic on w-bit words, and draw w-bit random variables, all in constant time. We will perform all above operations on $O(\log(\varepsilon^{-1} \cdot n))$-bit words, which is still $O(w)$ provided $\varepsilon^{-1} = \text{poly}(n)$. If ε is much smaller, all stated running times trivially increase by a factor of $O(\log(\varepsilon^{-1}))$.

Notation. For multisets S_1 and S_2, we denote by $S_1 \uplus S_2$ the "union" multiset, in which each element has multiplicity that is the sum of its multiplicities in S_1 and S_2. A vector $\mathbf{x}$ is *λ-uniform* if $x_e = \lambda$ for all $e \in \text{supp}(\mathbf{x})$, and is *uniform* if it is λ-uniform for some λ. Given fractional matching $\mathbf{x}$, we call an integral matching $M \subseteq \text{supp}(\mathbf{x})$ that is $(1-\varepsilon)$-approximate, i.e., $|M| \geq \|\mathbf{x}\| \cdot (1-\varepsilon)$ an *ε-approximate rounding* of $\mathbf{x}$. Finally, we use the following notion of distance and its monotonicity.

Observation 2.1. *For vectors* $\mathbf{x}, \mathbf{y} \in \mathbb{R}^E$ *and* $\varepsilon \geq 0$, *define* $d_V^{\epsilon}(\mathbf{x}, \mathbf{y}) := \sum_{v \in V} (|x(v) - y(v)| - \epsilon)^+$, *for* $(z)^+ := \max(0, z)$ *the positive part of*

[8]Some works study approximation of maximum matching *size* [6, 8, 22, 24, 28, 51].

$z \in \mathbb{R}$. Then, we have $d_V^{\epsilon}(\mathbf{x},\mathbf{y}) \le d_V^{\epsilon'}(\mathbf{x},\mathbf{y})$ for all $\epsilon \ge \epsilon'$. Moreover, by the triangle inequality and the basic fact that $(a+b)^+ \le a^+ + b^+$ for all real a, b, we have $d_V^{\epsilon_1+\epsilon_2}(\mathbf{x},\mathbf{z}) \le d_V^{\epsilon_1}(\mathbf{x},\mathbf{y}) + d_V^{\epsilon_2}(\mathbf{y},\mathbf{z})$ for all $\epsilon_1, \epsilon_2 \ge 0$ and vectors $\mathbf{x},\mathbf{y},\mathbf{z} \in \mathbb{R}^E$.

Support and Binary encoding. We denote the binary encoding of each edge e's fractional value by $x_e := \sum_i (x_e)_i \cdot 2^{-i}$. We further let $\text{supp}_i(\mathbf{x}) := \{e \in E \mid (x_e)_i = 1\}$ denote the set of coordinates of $\mathbf{x}$ whose i-th bit is a 1. So, $\text{supp}(\mathbf{x}) = \bigcup_i \text{supp}_i(\mathbf{x})$. Next, we let $\mathbf{x}_{\min} := \min_{e \in \text{supp}(\mathbf{x})} x_e$. The following observation allows us to restrict our attention to a small number of bits when rounding bipartite fractional matchings $\mathbf{x}$. (In the full version we extend this observation to the structured fractional matchings in general graphs that interest us there.)

Observation 2.2. *For rounding bipartite fractional matching, by decreasing ε by a constant factor, it is without loss of generality that $\mathbf{x}_{\min} \ge \varepsilon/n^2$ and moreover if $\Delta \le \mathbf{x}_{\min}$ and $L := 1 + \lceil \log(\varepsilon^{-1}\Delta^{-1}) \rceil$, we may safely assume that $(x_e)_i = 0$ for all $i > L$.*

Proof. Let $\varepsilon' = \varepsilon/3$. Consider the vector $\mathbf{x}'$ obtained by zeroing out all entries e of $\mathbf{x}$ with $x_e < \varepsilon'/n^2$ and setting $(x_e)_i = 0$ for all edges e and indices $i > L$. Clearly, $\text{supp}(\mathbf{x}') \subseteq \text{supp}(\mathbf{x})$ and $\mathbf{x}'$ is a fractional matching, as $\mathbf{x}' \le \mathbf{x}$. The following shows that $\|\mathbf{x}'\|$ is not much smaller than $\|\mathbf{x}\| \ge 1$.

$$\begin{aligned}\|\mathbf{x}'\| &\ge \|\mathbf{x}\| - \binom{n}{2}\frac{\varepsilon'}{n^2} - \sum_e \sum_{i>L} 2^{-i} \\ &\ge \|\mathbf{x}\| - \varepsilon' - \sum_e \varepsilon' \cdot \mathbf{x}_{\min} \\ &\ge \|\mathbf{x}\| \cdot (1-\varepsilon') - \sum_e \varepsilon' \cdot x_e \\ &= (1-2\varepsilon') \cdot \|\mathbf{x}\|.\end{aligned}$$

Therefore, a matching $M \subseteq \text{supp}(\mathbf{x}') \subseteq \text{supp}(\mathbf{x})$ that is $(1-\varepsilon')$-approximate w.r.t. x' is $(1-\varepsilon)$-approximate w.r.t. $\mathbf{x}$, as $|M| \ge (1-\varepsilon') \cdot \|\mathbf{x}'\| \ge (1-3\varepsilon') \cdot \|\mathbf{x}\| = (1-\varepsilon) \cdot \|\mathbf{x}\|$. □

Recourse Lower Bound. We note that the number of changes to M per update (a.k.a. the rounding algorithm's *recourse*) is at least $\Omega(\varepsilon^{-1})$ in the worst case.

Fact 2.3. *Any $(1-\varepsilon)$-approximate dynamic matching rounding algorithm $\mathcal{A}$ must use $\Omega(\varepsilon^{-1})$ amortized recourse, even in bipartite graphs.*

Proof. Consider a path graph G of odd length $4\varepsilon^{-1} + 2$ with values 1/2 assigned to each edge. A matching $M \subseteq \text{supp}(\mathbf{x})$ of size $|M| \ge (1-\varepsilon) \cdot \|\mathbf{x}\|$ must match all odd-indexed edges of the path. However, after invoking `update`$(\cdot, 0)$ for the first and last edges in the path, for $|M| \ge (1-\varepsilon) \cdot \|\mathbf{x}\|$ (for the new $\mathbf{x}$), M must match all *even*-indexed edges. Therefore, repeatedly invoking `update`$(\cdot, 0)$ and then `update`$(\cdot, 1/2)$ for these two edges sufficiently many times implies that the matching M maintained by $\mathcal{A}$ must change by an average of $\Omega(\varepsilon^{-1})$ edges per update. □

2.1 The *Degree-Split* subroutine

Throughout the paper, we use the following subroutine to partition a graph into two subgraphs of roughly equal sizes while roughly halving all vertices' degrees. Such subroutines obtained by e.g., computing maximal walks and partitioning them into odd/indexed edges, have appeared in the literature before in various places. For completeness, we provide this algorithm in the full version.

Proposition 2.4. *There exists an algorithm* `degree-split`*, which on multigraph $G = (V, E)$ with maximum edge multiplicity at most two (i.e., no edge has more than two copies) computes in $O(|E|)$ time two (simple) edge-sets E_1 and E_2 of two disjoint sub-graphs of G, such that E_1, E_2 and the degrees $d_G(v)$ and $d_i(v)$ of v in G and $H_i := (V, E_i)$ satisfy the following.*

1. *(P1)$|E_1| = \lceil \frac{|E|}{2} \rceil$ and $|E_2| = \lfloor \frac{|E|}{2} \rfloor$.*
2. *(P2) $d_i(v) \in \left[\frac{d_G(v)}{2} - 1, \frac{d_G(v)}{2} + 1\right]$ for each vertex $v \in V$ and $i \in \{1, 2\}$.*
3. *(p3) $d_i(v) \in \left[\lfloor \frac{d_G(v)}{2} \rfloor, \lceil \frac{d_G(v)}{2} \rceil\right]$ for each vertex $v \in V$ and $i \in \{1, 2\}$ $\underline{\text{if } G \text{ is bipartite}}$.*

3 SIMPLE ROUNDING FOR BIPARTITE MATCHINGS

In this section we use the binary encoding of $\mathbf{x}$ to approximately round fractional bipartite matchings in a "linear" manner, rounding from the least-significant to most-significant bit of the encoding. We first illustrate this approach in a static setting in Section 3.1. This will serve as a warm-up for our first dynamic rounding algorithm provided in Section 3.2, which is essentially a dynamic variant of the static algorithm (with its `init` procedure being essentially the static algorithm).

3.1 Warm-up: Static Bipartite Rounding

In this section, we provide a simple static bipartite rounding algorithm for fractional matchings.

Specifically, we prove the following Theorem 3.1, analyzing our rounding algorithm, Algorithm 1. The algorithm simply considers for all i, $E_i := \text{supp}_i(x)$, i.e., the edges whose i-th bit is set to one in $\mathbf{x}$. Starting from $F_L = \emptyset$, for $i = L, \ldots, 1$, the algorithm applies `degree-split` to the multigraph $G[F_i \uplus E_i]$ and sets F_{i-1} to be the first edge-set output by `degree-split` (by induction, E_i, F_i are simple sets, and so $G[F_i \uplus E_i]$ has maximum multiplicity two.) Overloading notation slightly, we denote this by $F_{i-1} \leftarrow$ `degree-split`$(G[F_i \uplus E_i])$. The algorithm then outputs $E_0 \cup F_0$.

Algorithm 1: Hierarchical Fractional Rounding Algorithm

```
input  : Fractional matching $\mathbf{x} \in \mathbb{R}^E_{\ge 0}$ in graph $G = (V, E)$
input  : Accuracy parameter $\epsilon \in (0, 1)$
output : Integral matching $M \subseteq \text{supp}(\mathbf{x})$ with
         $|M| \ge (1-\epsilon) \cdot \|\mathbf{x}\|$
$L \leftarrow 1 + \lceil \log_2(\epsilon^{-1} x_{\min}^{-1}) \rceil$ and $F_L \leftarrow \emptyset$;
for $i = L, L-1, \ldots, 1$ do
    $E_i \leftarrow \text{supp}_i(\mathbf{x})$;
    $F_{i-1} \leftarrow$ degree-split($G[E_i \uplus F_i]$) ;    // First set
     output by degree-split
end
return $M \leftarrow E_0 \cup F_0$;
```

Theorem 3.1. *On fractional bipartite matching* $\mathbf{x}$ *and error parameter* $\epsilon \in (0,1)$*,Algorithm 1 outputs an integral matching* $M \subseteq \text{supp}(\mathbf{x})$ *with* $|M| \geq (1-\epsilon) \cdot \|\mathbf{x}\|$ *in time* $O(|\text{supp}(\mathbf{x})| \cdot \log(\epsilon^{-1}\mathbf{x}_{\min}^{-1}))$*.*

By Observation 2.2, $L = O(\log(\varepsilon^{-1} \cdot n))$, and so Theorem 3.1 implies an $O(|\text{supp}(\mathbf{x})| \cdot \log(\varepsilon^{-1} \cdot n))$ runtime for Algorithm 1. We prove this theorem in several steps. Key to our analysis is the following sequence of vectors (which we will soon show are fractional matchings if supp(**x**) is bipartite).

Definition 3.2. *Letting* $F_i(e) := \mathbb{1}[e \in F_i]$ *and* $E_i(e) := \mathbb{1}[e \in E_i] = (x_e)_i$*, we define a sequence of vectors* $\mathbf{x}^{(i)} \in \mathbb{R}^E_{\geq 0}$ *for* $i = 0, 1, \ldots, L$ *as follows.*

$$x_e^{(i)} := F_i(e) \cdot 2^{-i} + \sum_{j=0}^{i} E_j(e) \cdot 2^{-j}. \tag{1}$$

So, $\|\mathbf{x}^{(L)}\| \geq (1-\varepsilon)\cdot\|\mathbf{x}\|$, by definition and Observation 2.2. Moreover, each (copy of) edge e output/discarded by `degree-split`$(G[E_i \uplus F_i])$ corresponds to adding/subtracting 2^{-i} to/from $x_e^{(i)}$ to obtain $x_e^{(i-1)}$. This allows us to prove the following lower bound on the size of the output.

Lemma 3.3. $\|\mathbf{x}^{(i)}\| \geq (1-\varepsilon) \cdot \|\mathbf{x}\|$ *for all* $i \in \{0,1,\ldots,L\}$*.*

Proof. By Property 1 we have that $|F_{i-1}| \geq \lceil \frac{1}{2}(|F_i| + |E_i|) \rceil$ and so

$$\|\mathbf{x}^{(i-1)}\| = \|\mathbf{x}^{(i)}\| + 2^{1-i} \cdot \sum_e F_{i-1}(e) - \sum_e 2^{-i} \cdot (F_i(e) + E_i(e)) \geq \|\mathbf{x}^{(i)}\|.$$

Therefore, repeatedly invoking the above bound and appealing to Observation 2.2, we have that indeed, for all $i \in \{0,1,\ldots,L\}$,

$$\|\mathbf{x}^{(i)}\| \geq \|\mathbf{x}^{(i+1)}\| \geq \cdots \geq \|\mathbf{x}^{(L)}\| \geq (1-\varepsilon) \cdot \|\mathbf{x}\|. \qquad \square$$

A simple proof by induction shows that if supp(**x**) is bipartite, then the above procedure preserves all vertices' fractional degree constraints, i.e., the vectors $\mathbf{x}^{(i)}$ are all fractional matchings.

Lemma 3.4. *If* $\mathbf{x}$ *is a fractional bipartite matching then* $x^{(i)}(v) \leq 1$ *for every vertex* $v \in V$ *and* $i \in \{0,1,\ldots,L\}$*.*

Proof. By reverse induction on $i \leq L$. The base case holds since $\mathbf{x}^{(L)} \leq \mathbf{x}$ is a fractional matching. To prove the inductive step for $i-1$ assuming the inductive hypothesis $x^{(i)}(v) \leq 1$, let $d_{G_i}(v) = \sum_{e \ni v}(E_i(e) + F_i(e))$ be the number of (possibly parallel) edges incident to v in $G_i := G[E_i \uplus F_i]$. By Property 3, we have the following upper bound on v's fractional degree under $\mathbf{x}^{(i-1)}$.

$$x^{(i-1)}(v) \leq x^{(i)}(v) - d_{G_i}(v) \cdot 2^{-i} + \left\lceil \frac{d_{G_i}(v)}{2} \right\rceil \cdot 2^{-i+1}. \tag{2}$$

If $d_{G_i}(v)$ is even, then we are done, by the inductive hypothesis giving $x^{(i-1)}(v) \leq x^{(i)}(v) \leq 1$. Suppose therefore that $d_{G_i}(v)$ is odd. By Definition 1, any value $x_e^{(i)}$ is evenly divisible by 2^{-i} and therefore the same holds for $x^{(i)}(v)$. By the same token, $d_{G_i}(v)$ is odd if and only if $x^{(i)}(v)$ is not evenly divisible by 2^{-i+1}. However, since $x^{(i)}(v)$ is evenly divisible by 2^{-i} and it is at most one, this implies that $x^{(i)}(v) \leq 1 - 2^{-i}$. Combined with Equation 2, we obtain the desired inequality when $d_{G_i}(v)$ is odd as well, since

$$x^{(i-1)}(v) \leq x^{(i)}(v) + 2^{-i} \leq 1 - 2^{-i} + 2^{-i} = 1. \qquad \square$$

Now, since the vector $\mathbf{x}^{(0)}$ is integral, the preceding lemmas imply that if $\mathbf{x}$ is a bipartite fractional matching then M is a large *integral* matching.

Lemma 3.5. *If* $\mathbf{x}$ *is a fractional bipartite matching then* $M = E_0 \cup F_0 \subseteq \text{supp}(\mathbf{x})$ *is an integral matching of cardinality at least* $|M| = |E_0| + |F_0| \geq (1-\varepsilon) \cdot \|\mathbf{x}\|$*.*

Proof. By Lemma 3.4, the (binary) vector $\mathbf{x}^{(0)}$ (the characteristic vector of M) is a feasible fractional matching, and so M is indeed a matching. That $M \subseteq \text{supp}(\mathbf{x})$ follows from `degree-split` outputting a sub(multi)set of the edges of its input, and therefore a simple proof by induction proves that $\text{supp}(\mathbf{x}) \supseteq \text{supp}(\mathbf{x}^{(L)}) \supseteq \text{supp}(\mathbf{x}^{(L-1)}) \supseteq \cdots \supseteq \text{supp}(\mathbf{x}^{(0)}) = M$. The lower bound on $|M| = \|x^{(0)}\|$ then follows from Lemma 3.3. $\square$

Finally, we bound the algorithm's running time.

Lemma 3.6. *Algorithm 1 takes time* $O(|\text{supp}(\mathbf{x})| \cdot L)$ *when run on vector* $\mathbf{x} \in \mathbb{R}^E_{\geq 0}$*.*

Proof. To analyze the runtime of the algorithm, note that it runs in time $O(L + \sum_{i=0}^{L}(|F_i| + |E_i|))$. Further, $|F_L| = 0$ and by Property 1 we have that $|F_i| \leq \frac{1}{2}|F_{i+1}| + \frac{1}{2}|E_{i+1}| + 1$ for all $i \in \{0,1,\ldots,L-1\}$. Letting $m := |\text{supp}(\mathbf{x})|$ we know that $|E_i| \leq m$ for all i, and so by induction

$$|F_i| \leq \frac{1}{2}|F_{i+1}| + \frac{1}{2}m + 1 \leq m + 2 \qquad \text{for all } j \in \{0,1,...,L-1\}.$$

Thus, the algorithm runs in the desired time of $O(mL + L) = O(mL)$. $\square$

Theorem 3.1 follows by combining the two preceding lemmas. We now turn to the dynamic counterpart of Algorithm 1.

3.2 A Simple Dynamic Bipartite Rounding Algorithm

In this section we dynamize the preceding warm-up static algorithm, obtaining the following result.

Theorem 3.7. *Algorithm 2 is a deterministic dynamic bipartite matching rounding algorithm. Under the promise that the dynamic input vector* $\mathbf{x}$ *satisfies* $\mathbf{x}_{\min} \geq \delta$ *throughout, its amortized* `update` *time is* $O(\varepsilon^{-1} \cdot \log^2(\varepsilon^{-1}\delta^{-1}))$ *and its* `init` *time on vector* $\mathbf{x}$ *is* $O(|\text{supp}(\mathbf{x})| \cdot \log(\varepsilon^{-1}\delta^{-1}))$*.*

Since $\delta \geq \varepsilon/n^2$ by Observation 2.2, Theorem 3.7 yields an $\tilde{O}(\varepsilon^{-1} \cdot \log^2 n)$ update time algorithm.

Our dynamic algorithm follows the preceding static approach. For example, its initialization is precisely the static Algorithm 1 (and so the `init` time follows from Theorem 3.1). In particular, the algorithm considers a sequence of graphs $G_i := G[E_i \uplus F_i]$ and fractional matchings $\mathbf{x}^{(i)}$ defined by G_i and the i most significant bits of x_e, as in Definition 1. However, to allow for low (amortized) update time we allow for a small number of unprocessed changed or deleted edges for each i, denoted by c_i. When such a number c_i becomes large, we *rebuild* the solution defined by F_i and $\mathrm{supp}_i(\mathbf{x}), \ldots, \mathrm{supp}_0(\mathbf{x})$ as in the static algorithm. Formally, our algorithm is given in Algorithm 2.

```
Algorithm 2: Hierarchical Dynamic Fractional Rounding Algorithm
global : Current vector x
global : Current output integral matching M
global : Accuracy $\epsilon$ and maximum layer $L \in \mathbb{Z}_{>0}$
global : Partial roundings $F_0, F_1, \ldots, F_L \subseteq E$ and update
         counts $c_0, c_1, \ldots, c_L \in \mathbb{Z}_{\geq 0}$
// In init we assume that the algorithm knows $\delta$,
   a lower bound on $\mathbf{x}_{\min}$ for all nonzero x
   encountered after an operation
function init($G = (V, E)$, $\mathbf{x} \in \mathbb{R}^E_{\geq 0}$, $\epsilon \in (0, 1)$)
    Save x and $\epsilon$ as global variables;
    Initialize $L \leftarrow 1 + \lceil \log_2(\varepsilon^{-1}\delta^{-1}) \rceil$, $c_i \leftarrow 0$, and $F_i \leftarrow \emptyset$,
     for all $i \in \{0, 1, \ldots, L\}$;
    Call rebuild($L$);
function rebuild($i$)
    for $j = i, i-1, \ldots, 0$ do
        $E_j \leftarrow \mathrm{supp}_j(\mathbf{x})$ and $c_j \leftarrow 0$;
        if $j \neq 0$ then $F_{j-1} \leftarrow$ degree-split($G[E_j \uplus F_j]$);
    end
    $M \leftarrow E_0 \cup F_0$;
function update($e, \nu$)
    $\mathbf{x}_e \leftarrow \nu$;
    for $i = L, L-1, \ldots, 0$ do
        if $e \in E_i$ then remove $e$ from $E_i$;
        if $i \neq 0$ then
            if $e \in F_{i-1}$ then remove $e$ from $F_{i-1}$;
            else remove one edge adjacent to each endpoint
             of $e$ from $F_{i-1}$ (if there is one);
        end
        $c_i \leftarrow c_i + 1$;
        if $c_i > 2^{i-2} \cdot \frac{\varepsilon \|\mathbf{x}\|}{L}$ then call rebuild($i$) and return;
    end
```

Conventions and notation. Most of our lemmas concerning Algorithm 2 hold for arbitrary non-negative vectors $\mathbf{x} \in \mathbb{R}^E_{\geq 0}$, a fact that will prove useful in later sections. We state explicitly which lemmas hold if $\mathbf{x}$ is a fractional bipartite matching. In the analysis of Algorithm 2 we let $\mathbf{x}^{(i)}$ be as defined in Definition 1, but with E_i and F_i of the dynamic algorithm. Furthermore, we prove all structural properties of Algorithm 2 for any time after `init` and any number of `update` operations, and so we avoid stating this in all these lemmas' statements for brevity. Next, we use the shorthand $S_i := \mathrm{supp}_i(\mathbf{x})$, and note that unlike in the static algorithm, due to deletions from E_i before the next `rebuild`(i), the containment $E_i \subseteq S_i$ may be strict.

First, we prove that M is a matching if $\mathbf{x}$ is a bipartite fractional matching. More generally, we prove that each $\mathbf{x}^{(i)}$, and in particular $\mathbf{x}^{(0)}$, is a fractional matching, implying the above.

Lemma 3.8. *If $\mathbf{x}$ is a fractional bipartite matching, then $\mathbf{x}^{(i)}$ is a fractional matching for all $i \in \{0, 1, \ldots, L\}$.*

Proof. Fix vertex v, and let $F_i(v)$ and $S_i(v)$ be the number of edges of v in F_i and S_i respectively, for all $i \in \{0, 1, \ldots, L\}$. To upper bound $x^{(i)}(v)$, we start by upper bounding $F_i(v)$, as follows.

$$F_i(v) \leq \left\lceil 2^i \cdot \sum_{j=i+1}^{L} S_j(v) \cdot 2^{-j} \right\rceil . \tag{3}$$

We prove the above by induction on the number of operations and by reverse induction on $i \in \{0, 1, \ldots, L\}$, as follows. The base case $i = L$ is trivial, as $F_L(v) = 0$ throughout and the RHS is non-negative. Next, for $i < L$, consider the effect on $F_i(v)$ of an update resulting in a call to `rebuild`($i+1$) (e.g., after calling `init`), at which point $E_{i+1} \leftarrow S_{i+1}$.

$$\begin{aligned}
F_i(v) &\leq \left\lceil \frac{1}{2} \cdot (S_{i+1}(v) + F_{i+1}(v)) \right\rceil && \text{Property 3} \\
&\leq \left\lceil \frac{1}{2} \cdot S_{i+1}(v) + \frac{1}{2} \cdot \left\lceil 2^{i+1} \cdot \sum_{j=i+2}^{L} S_j(v) \cdot 2^{-j} \right\rceil \right\rceil \\
&\qquad \text{Inductive hypothesis for } i+1 \\
&\leq \left\lceil \frac{1}{2} \cdot S_{i+1}(v) + \frac{1}{2} \cdot 2^{i+1} \cdot \sum_{j=i+2}^{L} S_j(v) \cdot 2^{-j} \right\rceil \\
&= \left\lceil 2^i \cdot \sum_{j=i+1}^{L} S_j(v) \cdot 2^{-j} \right\rceil ,
\end{aligned}$$

where the last inequality follows from the basic fact that for non-negative y, z with y an integer, $\lceil \frac{1}{2} \cdot y + \frac{1}{2} \lceil z \rceil \rceil \leq \lceil \frac{1}{2}(y + z) \rceil$. Next, it remains to prove the inductive step for index i and a call to `update` for which `rebuild`($i+1$) is not called: but such an update only decreases the left-hand side of Equation 3, while it causes a decrease in the right-hand side (by one) only if an edge of v was updated in this call to `update`, in which case we delete at least one edge incident to v in F_i, if any exist, and so the left-hand side also decreases by one (or is already zero).

Finally, combining Definition 1 and Equation 3, we obtain the desired inequality $x^{(i)}(v) \leq 1$.

$$\begin{aligned}
x^{(i)}(v) &\leq F_i(v) \cdot 2^{-i} + \sum_{j=0}^{i} S_j(v) \cdot 2^{-j} \\
&\leq 2^{-i} \cdot \left\lceil 2^i \cdot \sum_{j=i+1}^{L} S_j(v) \cdot 2^{-j} \right\rceil + \sum_{j=0}^{i} S_j(v) \cdot 2^{-j} \leq 1.
\end{aligned}$$

Above, the first inequality follows from Definition 1 and $E_i(v) \leq S_i(v)$ since $E_i \subseteq S_i$. The second inequality follows from Equation 3. Finally, the final inequality relies on $\sum_j S_j(v) \cdot 2^{-j} = x(v) \leq 1$, together with $2^i \cdot \sum_{j=i+1}^{L} S_j(v) \cdot 2^{-j}$ being fractional if and only if $\sum_{j=i+1}^{L} S_j(v) \cdot 2^{-j}$ is not evenly divisible by 2^{-i}, though it is evenly divisible by 2^{-i-1}, in which case $x(v) \leq 1 - 2^{-i}$. ◻

Remark 3.9. *The same proof approach, using Property 1 of Degree-Split (for possibly non-bipartite graphs) implies the global upper bound $|F_i| \le \left\lceil 2^i \cdot \sum_{j=i+1}^{L} |S_j| \cdot 2^{-j} \right\rceil \le 1 + 2^i \cdot \|\mathbf{x}\|$.*

Next, we prove the second property of a rounding algorithm, namely that $M \subseteq \mathrm{supp}(\mathbf{x})$.

Lemma 3.10. $\mathrm{supp}(\mathbf{x}^{(i)}) \subseteq \mathrm{supp}(\mathbf{x})$ *for all $i \in \{0, 1 \ldots, L\}$ and therefore $M \subseteq \mathrm{supp}(\mathbf{x})$.*

Proof. We prove the stronger claim by induction on the number of operations of Algorithm 2 and by reverse induction on $i \in \{0, 1, \ldots, L\}$ that $\mathrm{supp}(\mathbf{x}^{(i)}) = E_i \cup F_i \subseteq \mathrm{supp}(\mathbf{x}^{(i+1)}) \subseteq \mathrm{supp}(\mathbf{x})$. That $E_i \subseteq \mathrm{supp}_i(\mathbf{x})$ throughout is immediate, since $E_i \leftarrow \mathrm{supp}_i(\mathbf{x})$ when `rebuild`(i) is called (and in particular after `init` was called), and subsequently all edges $e \in E_i$ that are updated (and in particular each edge whose x_e value is set to zero) are removed from E_i. Therefore $E_i \subseteq \mathrm{supp}(\mathbf{x})$ throughout, and in particular $\mathrm{supp}(\mathbf{x}^{(L)}) = E_L \subseteq \mathrm{supp}(\mathbf{x})$. Similarly, by the properties of `degree-split` and the inductive hypothesis, we have that after `rebuild`(i) is called, $F_i \subseteq F_{i+1} \cup E_{i+1} \subseteq \mathrm{supp}(\mathbf{x}^{(i+1)}) \subseteq \mathrm{supp}(\mathbf{x})$, and each edge e updated since is subsequently deleted from F_i (as are some additional edges). Therefore $F_i \subseteq \mathrm{supp}(\mathbf{x})$ throughout. We conclude that $\mathrm{supp}(\mathbf{x}^{(i)}) \subseteq \mathrm{supp}(\mathbf{x})$ for all i, as desired. □

We now argue that the unprocessed edges have a negligible effect on values of $\mathbf{x}^{(i)}$ compared to their counterparts obtained by running the static algorithm on the entire input $\mathbf{x}$.

Lemma 3.11. $\|\mathbf{x}^{(i)}\| \ge (1 - 2\varepsilon) \cdot \|\mathbf{x}\|$ *for every $i \in \{0, 1, \ldots, L\}$.*

Proof. As in the proof of Lemma 3.3, by Property 1, after `init` or any `update`$(\cdot, \cdot)$ triggering a call to `rebuild`(i) we have that $|F_{i-1}| = \lceil \frac{1}{2}(|F_i| + |E_i|) \rceil$, and so $\|\mathbf{x}^{(i-1)}\| \ge \|\mathbf{x}^{(i)}\|$. On the other hand, between calls to `rebuild`(i) there are at most $2^{i-2} \cdot \frac{\varepsilon\|\mathbf{x}\|}{L}$ calls to `update`(e, v), resulting in at most $2^{i-1} \cdot \frac{\varepsilon\|\mathbf{x}\|}{L}$ many edges being deleted from F_{i-1}, which in turn result in $\|\mathbf{x}^{(i-1)}\|$ decreasing by at most $2^{-(i-1)} \cdot 2^{i-1} \cdot \frac{\varepsilon\|\mathbf{x}\|}{L} = \frac{\varepsilon\|\mathbf{x}\|}{L}$. In contrast, by Definition 1, any changes in E_j for $j \ne i$ have no effect on $\|\mathbf{x}^{(i-1)}\| - \|\mathbf{x}^{(i)}\|$. On the other hand, until the next `rebuild`(i) is triggered, we have that E_i and F_i can only decrease (contributing to an increase in $\|x^{(i-1)}\| - \|x^{(i)}\|$); E_i can only decrease since edges are only added to E_i when `rebuild`(i) is called, and F_i only decreases until `rebuild`$(i+1)$ is called, which triggers a call to `rebuild`(i). Therefore, $\|x^{(i-1)}\| - \|x^{(i)}\|$ decreases by at most $\frac{\varepsilon\|\mathbf{x}\|}{L}$ during updates until the next call to `rebuild`(i), and so after `init` and after every `update` of Algorithm 2, we have that

$$\|\mathbf{x}^{(i-1)}\| \ge \|\mathbf{x}^{(i)}\| - \frac{\varepsilon\|\mathbf{x}\|}{L}.$$

Invoking the above inequality L times, and using that $\|\mathbf{x}^{(L)}\| \ge (1 - \varepsilon) \cdot \|\mathbf{x}\|$ by Observation 2.2, we obtain the desired inequality.

$$\begin{aligned} |E_0 \cup F_0| = \|\mathbf{x}^{(0)}\| &\ge \|\mathbf{x}^{(1)}\| - \frac{\varepsilon\|\mathbf{x}\|}{L} \ge \ldots \\ &\ge \|\mathbf{x}^{(L)}\| - L \cdot \frac{\varepsilon\|\mathbf{x}\|}{L} \ge (1 - 2\varepsilon) \cdot \|\mathbf{x}\|. \end{aligned}$$ □

Remark 3.12. *The latter is nearly tight, as $\|\mathbf{x}^{(i)}\| \le (1 + \epsilon) \cdot \|\mathbf{x}\| + 2^{1-i}$ for every $i \in \{0, 1, \ldots, L\}$.*

Proof (Sketch). The proof follows that of Lemma 3.11, with the following changes: By Property Property 1, after `init` or any `update`$(\cdot, \cdot)$ triggering a call to `rebuild`(i) we have the upper bound $|F_{i-1}| = \lceil \frac{1}{2} \le 1 + \frac{1}{2}(|F_i| + |E_i|)$, and so $\|\mathbf{x}^{(i-1)}\| \le \|\mathbf{x}^{(i)}\| + 2^{-i+1}$. On the other hand, the increase in $\|x^{(i-1)}\| - \|x^{(i)}\|$ until such a `rebuild`(i) is at most $\frac{\varepsilon\|\mathbf{x}\|}{L}$ (similarly to the decrease in the same). The proof then concludes similarly to that of Lemma 3.11, also using that $\|\mathbf{x}^{(L)}\| \le \|\mathbf{x}\|$. □

Finally, we turn to analyzing the algorithm's update time.

Lemma 3.13. *The (amortized) time per* `update` *of Algorithm 2 is $O(\varepsilon^{-1} \cdot L^2)$.*

Proof. By Remark 3.9, $|F_i| \le 1 + 2^i \cdot \|\mathbf{x}\| = O(2^i \cdot \|\mathbf{x}\|)$ (recalling that without loss of generality $\|\mathbf{x}\| \ge 1$). Similarly, trivially $|S_i| = 2^i \cdot 2^{-i} \cdot |S_i| \le 2^i \cdot \|\mathbf{x}\|$. Therefore, by Proposition 2.4, the calls to `degree-split`$(G[E_i \uplus F_i])$ in Algorithm 2 (at which point $E_i = S_i$) take time $O(2^i \cdot \|\mathbf{x}\|)$, and so the time for `rebuild`(i) is $\sum_{j=0}^{i} O(2^j \cdot \|\mathbf{x}\|) = O(2^i \cdot \|\mathbf{x}\|)$. But since `rebuild`$(i)$ is called after $2^{i-2} \cdot \frac{\varepsilon\|\mathbf{x}\|}{L}$ updates, its cost amortizes to $O(\varepsilon^{-1} \cdot L)$ time per update. Summing over all $i \in \{0, 1, \ldots, L\}$, we find that indeed, the amortized time per `update` operation, which is $O(L)$ (due to deleting $O(1)$ edges from each E_i and F_i for each i) plus its contribution to periodic calls to `rebuild`, is $O(\varepsilon^{-1} \cdot L^2)$. □

We are finally ready to prove Theorem 3.7.

Proof of Theorem 3.7. Algorithm 2 is a dynamic rounding algorithm for bipartite fractional matchings, since M is a matching contained in $\mathrm{supp}(\mathbf{x}_0) \subseteq \mathrm{supp}(\mathbf{x}_1) \subseteq \cdots \subseteq \mathrm{supp}(\mathbf{x})$ if the latter is bipartite, by Lemma 3.8 and Lemma 3.10, and moreover $|M| = \|\mathbf{x}^{(0)}\| \ge (1 - 2\varepsilon) \cdot \|\mathbf{x}\|$, by Lemma 3.11. The algorithm's `update` time and `init` time follow from Lemma 3.13 and Theorem 3.1. □

To (nearly) conclude, this section provides a simple bipartite rounding algorithm with near-optimal ε-dependence. In the following section we show how *partially* rounding the fractional matching allows to dynamically guarantee that $\mathbf{x}_{\min}$ be sub-polynomial in ε/n, thus allowing us to decrease L and obtain speedups (improved n-dependence) when combined with Algorithm 2.

Algorithm 2 in general graphs. Before continuing to the next section, we mention that the alluded-to notion of partial rounding will also be useful when rounding (well-structured) fractional matchings in *general* graphs as well (see full version). With this in mind, we provide the following lemma, which is useful to analyze Algorithm 2 when rounding general graph matchings.

Lemma 3.14. *For $d_V^c(\mathbf{x}, \mathbf{y}) := \sum_{v \in V} (|x(v) - y(v)| - c)^+$, the vectors $\mathbf{x}^{(i)}$ satisfy*

$$d_V^{2^{-i+1}}(\mathbf{x}, \mathbf{x}^{(i)}) \le \varepsilon \cdot \|\mathbf{x}\| \qquad \forall i \in \{0, 1, \ldots, L\}.$$

Proof. First, we verify that the inequality holds (with some extra slack) right after `rebuild`(i) (and in particular right after init). First, by Property 2 of degree-split, during the invocation of which $E_i = S_i$, we have that

$$F_i(v) \in \left[\frac{1}{2}(E_{i+1}(v) + F_{i+1}(v) - 1,\ \frac{1}{2}(E_{i+1}(v) + F_{i+1}(v)) + 1\right]$$

. Therefore, by Definition 1, for each vertex v, we have after rebuild(i) that

$$|x^{(i)}(v) - x^{(i+1)}(v)| \le 2^{-i}.$$

On the other hand, before an update with current input $\mathbf{x}$ there are at most $2^{i-2} \cdot \frac{\varepsilon\|\mathbf{x}\|}{L}$ calls to update since the last call to rebuild(i), resulting in at most $3 \cdot 2^{i-2} \cdot \frac{\varepsilon\|\mathbf{x}\|}{L}$ many edges being added or deleted from $E_i \cup F_i$. Therefore, during the updates between calls to rebuild(i), the total variation distance between $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(i+1)}$ changes by at most $\frac{\varepsilon\|\mathbf{x}\|}{L}$, and so after init and after any update,

$$\sum_v \left(|x^{(i)}(v) - x^{(i+1)}(v)| - 2^{-i}\right)^+ \le \frac{\varepsilon\|\mathbf{x}\|}{L}.$$

Now, using the basic fact that $(a+b)^+ \le a^+ + b^+$ for all real a, b, summing the above difference over all i, and using that $\mathbf{x} = \mathbf{x}^{(L)}$ by Observation 2.2, we obtain the desired inequality, as follows.

$$\sum_v \left(|x^{(i)}(v) - x^{L}(v)| - 2^{-i+1}\right)^+ \le$$
$$\sum_{j=i}^{L-1} \sum_v \left(|x^{(i)}(v) - x^{(i+1)}(v)| - 2^{-i}\right)^+$$
$$\le L \cdot \frac{\varepsilon\|\mathbf{x}\|}{L} = \varepsilon\|\mathbf{x}\|. \qquad \square$$

4 PARTIAL ROUNDING: A PATH TO SPEEDUPS

So far, we have provided a rounding algorithm with near-optimal dependence on ε (by Fact 2.3) and polylogarithmic dependence on $\mathbf{x}_{\min}^{-1} = \text{poly}(\varepsilon^{-1}n)$ of the fractional matching $\mathbf{x}$. To speed up our algorithm we thus wish to dynamically maintain a "coarser" fractional matching $\mathbf{x}'$ (i.e., with larger $(\mathbf{x}'_{\min})^{-1}$ than $\mathbf{x}_{\min}^{-1}$) that approximately preserves the value of $\mathbf{x}$. The following definition captures this notion of coarser fractional matchings that we will use.[9]

Definition 4.1. *A vector $\mathbf{x}' \in \mathbb{R}^E_{\ge 0}$ is an (ϵ, δ)-*coarsening *of a vector $\mathbf{x} \in \mathbb{R}^E_{\ge 0}$ if:*

(0) *(P1)**Containment**: $\text{supp}(\mathbf{x}') \subseteq \text{supp}(\mathbf{x})$.*
(1) *(P2)**Global Slack**: $|\|\mathbf{x}\| - \|\mathbf{x}'\|| \le \varepsilon \cdot \|\mathbf{x}\| + \varepsilon$.*
(2) *(P3)**Vertex Slack**: $d^{\epsilon}_V(\mathbf{x}, \mathbf{x}') \le \varepsilon \cdot \|\mathbf{x}\| + \varepsilon$.*
(3) *(P4)**Edge Values**: $x'_e \in \{0\} \cup [\delta, 2\delta)$ if $x_e < \delta$ and $x'_e = x_e$ otherwise.*

The coarsening $\mathbf{x}'$ is bounded *if it also satisfies:*

(1) *(P4)**Boundedness**: $x'(v) \le x(v) + \varepsilon$ for all $v \in V$.*

We briefly motivate the above definition: As we shall see, properties 1 and 2 imply that $\mathbf{x}'$ (after mild post-processing) is a $(1-\varepsilon)$-approximation of $\mathbf{x}$, and so rounding $\mathbf{x}' \le \mathbf{x}$ results in a $(1-\varepsilon)^2 \ge (1-2\varepsilon)$-approximation of $\mathbf{x}$. The less immediately intuitive Property 2 will also prove useful when rounding in general graphs. For now, we will use this property when combining coarsenings of disjoint parts of the support of $\mathbf{x}$. Property 3 then allows us to round such coarsening $\mathbf{x}'$ efficiently, with only a polylogarithmic dependence on δ^{-1}, using Algorithm 2 (by Theorem 3.7). Finally, Property 1 guarantees that $\mathbf{x}'/(1+\varepsilon)$ is a fractional matching.

A key ingredient for subsequent sections is thus a dynamic coarsening algorithm, as follows.

Definition 4.2. *A dynamic (ε, δ)-coarsening algorithm is a data structure supporting the following operations:*

- init($G = (V, E)$, $\mathbf{x} \in \mathbb{R}^E_{\ge 0}$)*: initializes the data structure for undirected graph G with vertices V and edges E, current vector $\mathbf{x}$.*
- update($e \in E$, $\nu \in [0,1]$)*: sets $x_e \leftarrow \nu$.*

The algorithm must maintain an (ε, δ)-coarsening $\mathbf{x}'$ of (the current) $\mathbf{x}$.

As we show in the full version, the internal state of Algorithm 2 yields a dynamic coarsening algorithm. In this section we state bounds for a number of dynamic coarsening algorithms, with the objective of using their output as the input of Algorithm 2, from which we obtain faster dynamic bipartite rounding algorithms than when using the latter algorithm in isolation. The following lemma (proven in the complete version) captures the benefit of this approach.

Lemma 4.3. ***(From coarsening to rounding).*** *Let C be a dynamic (ϵ, δ)-coarsening algorithm with* update *time $t^C_U := t^C_U(\varepsilon, \delta, n)$ and* init *time $O(|\text{supp}(\mathbf{x})| \cdot t^C_I)$. Let $\mathcal{R}$ be a dynamic rounding algorithm for fractional matchings $\mathbf{x}$ with $\mathbf{x}_{\min} \ge \delta$, with* update *time $t^{\mathcal{R}}_U := t^{\mathcal{R}}_U(\varepsilon, \delta, n)$ and* init *time $O(|\text{supp}(\mathbf{x})| \cdot t^{\mathcal{R}}_I)$, for $t^{\mathcal{R}}_I := t^{\mathcal{R}}_I(\varepsilon, \delta, n)$. Then, there exists an $O(\epsilon + \delta)$-approximate dynamic rounding algorithm $\mathcal{R}^*$ with* update *time $O(t^C_U + t^{\mathcal{R}}_U + \varepsilon^{-1} \cdot t^{\mathcal{R}}_I)$ and* init *time $O(|\text{supp}(\mathbf{x})| \cdot (t^{\mathcal{R}}_I + t^C_I))$ which is deterministic/adaptive/output-adaptive if both $\mathcal{R}$ and C are.*

In our invocations of Lemma 4.3 we will use Algorithm 2 to play the role of Algorithm $\mathcal{R}$. In the complete version we provide a number of coarsening algorithm, whose properties we state in this section, together with the obtained rounding algorithms' guarantees.

A number of our coarsening algorithms will make use of subroutines for splitting (most of) the fractional matching's support into numerous disjoint coarsenings, as in the following.

Definition 4.4. *An (ϵ, δ)-*split *of fractional matching $\mathbf{z} \in \mathbb{R}^E_{\ge 0}$ with $\mathbf{z}_{\max} \le \delta$ consists of (ϵ, δ)-coarsenings $\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(k)}$ with disjoint supports, together covering at least half of* $\text{supp}(\mathbf{z})$*, i.e.,*

$$\sum_i |\text{supp}(\mathbf{z}^{(\mathbf{i})})| \ge \frac{1}{2} \cdot |\text{supp}(\mathbf{z})|.$$

The following lemma combined with Lemma 4.3 motivates our interest in such splits.

Lemma 4.5. ***(From static splitting to dynamic coarsening).*** *Let $\mathcal{S}$ be a static (γ, δ)-split algorithm with running time $|\text{supp}(\mathbf{x})| \cdot t_s$ on*

[9]In what follows, we use the definition of d^{ϵ}_V from Observation 2.1.

uniform fractional matching $\mathbf{x}$*, where* $t_s := t_s(n, \gamma, \delta)$*. Then there exists a dynamic algorithm* C *which for any (possibly non-uniform) fractional matching* $\mathbf{x}$ *maintains an* $\left(O\left(\varepsilon + \gamma \cdot \varepsilon^{-1} \cdot (\log(\gamma^{-1} \cdot n))\right), \delta\right)$*-coarsening of* $\mathbf{x}$ *with* `update` *time* $O(\varepsilon^{-1} \cdot t_s)$*,* `init` *time* $O(|\text{supp}(\mathbf{x})| \cdot t_s)$*. Algorithm* C *is deterministic/adaptive/output-adaptive if* $\mathcal{A}$ *is.*

Section outline. We prove Lemmas 4.34.5 in the complete version of the paper. Before that, we state bounds of a number of such partial rounding algorithms together with the rounding algorithms we obtain from these, yielding Theorem 1.2.

4.1 Partial Rounding Algorithms, with Applications

Here we state the properties of our coarsening and splitting algorithms (presented in the complete version of the paper), together with the implications to dynamic rounding, as stated in Theorem 1.2. Simmiliarly, we provide a deterministic static split algorithm as stated in the following lemma.

Lemma 4.6. *For any* $\varepsilon > 0$*, there exists a deterministic static* $(4\varepsilon, \varepsilon)$*-split algorithm which on input uniform fractional matchings* $\mathbf{x}$ *runs in time* $O\left(|\text{supp}(\mathbf{x})| \cdot \log(\epsilon^{-1} \cdot n)\right)$*.*

Combining the above lemma with Lemmas 4.34.5 yields the first result of Theorem 1.2.

Corollary 4.7. *There exists a deterministic dynamic bipartite rounding algorithm with* `update` *time* $O(\varepsilon^{-1} \cdot t(\varepsilon, n))$ *and* `init` *time* $O(|\text{supp}(\mathbf{x})| \cdot t(\varepsilon, n))$*, for* $t(\varepsilon, n) = \log n + \log^2(\varepsilon^{-1})$*.*

Proof. By Algorithmalg:det-split, there exists a deterministic static $\left(\frac{4\varepsilon^3}{\log n}, \frac{\varepsilon^3}{\log n}\right)$-split algorithm that on uniform fractional matching $\mathbf{x}$ runs in $O(|\text{supp}(\mathbf{x})| \cdot \log(\varepsilon^{-1} \cdot n))$ time. Plugging this algorithm into Lemma 4.5 yields a deterministic dynamic $\left(O(\varepsilon), \frac{\varepsilon^3}{\log n}\right)$-coarsening algorithm C with `update` time $t_U^C = O(\varepsilon^{-1} \cdot (\log n))$ and initialization time $O(|\text{supp}(\mathbf{x})| \cdot \log n)$. Moreover, by Theorem 3.7 there exists a deterministic dynamic bipartite matching rounding algorithm $\mathcal{R}$ for fractional bipartite matchings $\mathbf{x}$ with $\mathbf{x}_{\min} = \Omega(\text{poly}(\varepsilon^{-1} \cdot \log n))$ with `update` time $t_U^{\mathcal{R}} = O(\varepsilon^{-1} \cdot \log^2(\varepsilon^{-1} \cdot \log n))$ and `init` time $O(|\text{supp}(\mathbf{x})| \cdot t_I^{\mathcal{R}})$, for $t_I^{\mathcal{R}} = O(\log(\varepsilon^{-1} \cdot \log n))$. Plugging these algorithms into Lemma 4.10, we obtain a deterministic algorithm which has `update` time

$$
\begin{aligned}
&O(t_U^C + t_U^{\mathcal{R}} + t_I^{\mathcal{R}} \cdot \varepsilon^{-1}) = \\
&O(\varepsilon^{-1} \cdot (\log n + \log^2(\varepsilon^{-1} \cdot \log n))) = \\
&O(\varepsilon^{-1} \cdot (\log n + \log^2(\varepsilon^{-1}))
\end{aligned}
$$

and initialization time $O(|\text{supp}(\mathbf{x})| \cdot (\log n + \log^2(\varepsilon^- 1))$. The last equality holds for all ranges of n and ε, whether $\varepsilon^{-1} = O(\log n)$ or $\varepsilon^{-1} = \Omega(\log n)$. □

Next, in the complete version we provide a simple linear-time subsampling-based randomized split algorithm with the following properties.

Lemma 4.8. *For any* $\varepsilon > 0$*, there exists a static randomized algorithm that on uniform fractional matchings* $\mathbf{x}$ *computes an* $(\varepsilon, \frac{\varepsilon^4}{24 \log^2 n})$*-split in* $O(|\text{supp}(\mathbf{x})|)$*-time, and succeeds w.h.p.*

Combining the above lemma with Lemma 4.3 and Lemma 4.5 yields the w.h.p. result of Theorem 1.2.

Corollary 4.9. *There exists an adaptive dynamic bipartite rounding algorithm that succeeds w.h.p., with* `update` *time* $O(\varepsilon^{-1} \cdot t(\varepsilon, n))$ *and* `init` *time* $O(|\text{supp}(\mathbf{x})| \cdot t(\varepsilon, n))$*, for* $t(\varepsilon, n) = \log^2 \log n + \log^2(\varepsilon^{-1})$*.*

Proof. By Lemma 4.8, there exists a randomized static algorithm that computes a $\left(\frac{\varepsilon^3}{\log(n)}, \frac{\varepsilon^{12}}{24 \cdot \log^6(n)}\right)$-split of any uniform fractional matching $\mathbf{x}$ in $O(|\text{supp}(\mathbf{x})|)$ time, succeeding w.h.p. Plugging this algorithm into Lemma 4.5 we obtain a randomized (with high probability) dynamic $\left(O(\varepsilon), \frac{\varepsilon^{12}}{24 \cdot \log^6(n)}\right)$-coarsening algorithm C with `update` time $t_U^C = O(\varepsilon^{-1})$ and `init` time $O(|\text{supp}(\mathbf{x})|)$. On the other hand, by Theorem 3.7, there exists a deterministic dynamic bipartite matching rounding algorithm $\mathcal{R}$ for fractional matchings $\mathbf{x}$ with $\mathbf{x}_{\min} = \Omega(\text{poly}(\varepsilon^{-1} \cdot \log n))$ with `update` time $t_U^{\mathcal{R}} = O(\varepsilon^{-1} \cdot \log^2(\log n \cdot \varepsilon^{-1}))$ and `init` time $O(|\text{supp}(\mathbf{x})| \cdot t_I^{\mathcal{R}})$, for $t_I^{\mathcal{R}} = O(\log(\log n \cdot \varepsilon^{-1}))$. Plugging these algorithms into Lemma 4.10, we obtain a randomized adaptive algorithm which works with high probability and has `update` time

$$
\begin{aligned}
&O(t_U^C + t_U^{\mathcal{R}} + t_I^{\mathcal{R}} \cdot \varepsilon^{-1}) = \\
&O(\varepsilon^{-1} \cdot \log^2(\log n \cdot \varepsilon^{-1})) = \\
&O\left(\varepsilon^{-1} \cdot \left(\log^2 \log n + \log^2(\varepsilon^{-1})\right)\right)
\end{aligned}
$$

and `init` time $O(|\text{supp}(\mathbf{x})| \cdot \log^2(\log n) + \log^2(\varepsilon^{-1}))$. The last equality holds whether $\varepsilon^{-1} = O(\log n)$ or $\varepsilon^{-1} = \Omega(\log n)$. □

In the complete version of the paper building on a output-adaptive dynamic set sampling algorithm which we give an output-adaptive coarsening algorithm with constant (and in particular independent of n) expected amortized `update` time.

Lemma 4.10. *There exists an output-adaptive dynamic* $(O(\varepsilon), \varepsilon^3)$*-coarsening algorithm for dynamic fractional matchings* $\mathbf{x}$ *with expected* `update` *time* $O(\varepsilon^{-1})$ *and expected* `init` *time* $O(|\text{supp}(\mathbf{x})|)$*.*

Finally, combining the above lemma with Lemma 4.3 yields the third result of Theorem 1.2.

Corollary 4.11. *There exists an output-adaptive dynamic bipartite rounding algorithm with expected* `update` *time* $O(\varepsilon^{-1} \cdot t(\varepsilon))$ *and expected* `init` *time* $O(|\text{supp}(\mathbf{x})| \cdot t(\varepsilon))$ *for* $t(\varepsilon) = \log^2(\varepsilon^{-1})$*.*

Proof. By Lemma 4.10, there exists a dynamic $(\varepsilon, O(\varepsilon^3))$ coarsening algorithm C with expected `update` time $t_U^C = O(1)$ and `init` time $O(|\text{supp}(\mathbf{x})|)$. On the other hand, by Theorem 3.7 there exists a deterministic (hence output-adaptive) dynamic bipartite matching rounding algorithm $\mathcal{R}$ for fractional matchings $\mathbf{x}$ with $\mathbf{x}_{\min} = \Omega(\varepsilon^3)$ with `update` time $t_U^{\mathcal{R}} = O(\varepsilon^{-1} \cdot \log^2(\varepsilon^{-1}))$ and `init` time $O(|\text{supp}(\mathbf{x})| \cdot t_I^{\mathcal{R}})$, for $t_I^{\mathcal{R}} = O(\log(\varepsilon^{-1}))$. Plugging these algorithms into Lemma 4.10, we obtain an output-adaptive algorithm with expected `update` time $O(t_U^C + t_U^{\mathcal{R}} + t_I^{\mathcal{R}} \cdot \varepsilon^{-1}) = O(\varepsilon^{-1} \cdot \log^2(\varepsilon^{-1}))$ and expected

$$\texttt{init}\text{time}O(|\text{supp}(\mathbf{x})| \cdot \log^2(\varepsilon^{-1}))$$

. □

ACKNOWLEDGEMENTS

Author Sayan Bhattacharya was Supported by Engineering and Physical Sciences Research Council, UK (EPSRC) Grant EP/S03353X/1. Author Peter Kiss's work was done in part while visiting Max-Planck-Institut für Informatik. Author Aaron Sidford was supported in part by a Microsoft Research Faculty Fellowship, NSF CAREER Award CCF-1844855, NSF Grant CCF-1955039, a PayPal research award, and a Sloan Research Fellowship. Author David Wajc's work was done in part while at Stanford University and Google Research. Supported by a Taub Family Foundation "Leader in Science and Technology" fellowship. Thank you to Arun Jambulapati for helpful conversations which were foundational for the static bipartite rounding algorithm in this paper and subsequent developments. Part of this work was conducted while authors in positions $2^0, 2^1$ and 2^2 (alphabetically) were visiting Dagstuhl program 22461, "Dynamic Graph Algorithms".

While leaving much of the main sections of the paper to the full version (available at https://arxiv.org/abs/2306.11828) we included the following tool from the appendix which might be of independent research interest.

A DYNAMIC SET SAMPLING

In this section we provide an output-adaptive data structure for the dynamic set sampling problem (restated below). Recall that this is the basic problem of maintaining a dynamic subset of $[n]$ where every element is included in the subset independently with probability p_i under dynamic changes to p_i and re-sampling. This basic problem was studied by [30, 54, 56].

Definition A.1. *A dynamic set sampler is a data structure supporting the following operations:*

- `init`$(n, \mathbf{p} \in [0,1]^n)$*: initialize the data structure for n-size set S and probability vector* $\mathbf{p}$.
- `set`$(i \in [n], \alpha \in [0,1])$*: set* $p_i \leftarrow \alpha$.
- `sample`$()$*: return* $T \subseteq \mathbb{R}^n$ *containing each* $i \in S$ *independently with probability* p_i.

Our main result of this section is that we can implement in each operation in total time linear in the number of operations, n, and the size of the T output.

Theorem A.2. *Algorithm 3 is a set sampler data structure using* $O(n)$ *space that implements* `init` *in* $O(n)$ *time,* `set` *in* $O(1)$ *time, and* $T = $ `sample()` *in expected* $O(1 + |T|)$ *time in a word RAM model with word size* $w = \Omega(\log(p_{\min}^{-1}))$*, under the promise that* $p_i \geq p_{\min}$ *for all* $i \in [n]$ *throughout. These guarantees hold even if the input is chosen by an output-adaptive adversary.*

Comparison with the concurrent work of [56]. Our solution is somewhat simpler than that of the concurrent set sampler of [56]. For our purposes, the only important difference is that our algorithm is provably output-adaptive.

Our Algorithm 3 and associated proof of Theorem A.2 stem from a simple insight about computing when an element $i \in [n]$ will be in the output of `sample()`. Note that if there are no operations of the form `set`(i, α), then the probability i is in any individual output of `sample()` is p_i. Consequently, the probability that p_i is not in the output of `sample()` for the next t calls to `sample()` is $(1 - p_i)^t$. Therefore, the number of calls to `sample()` it takes for i to be in the output sample follows the geometric distribution with parameter p_i, i.e. Geo(p_i)!

Leveraging this simple insight above leads to an efficient set sampler data structures. Naively, implementing set sampling takes $O(n)$ time per call to `sample()`, used to determine for each element i whether or not it should be in the output. However, we could instead simply sample from geom(p_i) (in expected $O(1)$, using [29]) whenever `set`(i, α) is called or i is in the output of `sample()`, in order to determine *the next call* to `sample()` which will result in i being in the output. Provided we can efficiently keep track of this information for each i, this would yield the desired bounds in Theorem A.2.

Unfortunately, when sampling from geom(p_i), the output could be arbitrarily large (albeit with small probability). Further, maintaining the data structure for knowing when i is scheduled to be in the output of `sample()` would naively involve maintaining a heap on arbitrary large numbers, incurring logarithmic factors. There are many potential data-structures and techniques to solve this problem. In our Algorithm 3 we provide one simple, straightforward solution. Every n calls to `sample()`, we "rebuild" our data structure and rather than sampling from geom(p_i) to determine the next call to `sample()` that will output i, we instead simply sample to determine the next call to `sample()` before the rebuild that will output i (if there is one). Algorithm 3 simply does this, resampling this time for i whenever `set`(i, α) is called.

Algorithm 3: Set Sampler Data Structure

```
  global : Size $n$ and $p \in [0,1]^n$
  global : Current (relative) time $\tau$, subsets $T_1, ..., T_n \subseteq [n]$,
           and next sample times $\tau_1, ..., \tau_n \in [n+1]$
1 function init$(n, p \in [0,1]^n)$
2     Save $n$ and $p$ as global variables;
3     Initialize $\tau \leftarrow 1$, $T_i = \emptyset$, and $\tau_i = n+1$ for all $i \in [n]$;
4     Call set$(i, p_i)$ for all $i \in S$;
5 function set$(i \in [n], \alpha \in [0,1])$
6     if $\tau_i \neq n+1$ then $T_{\tau_i} \leftarrow T_{\tau_i} \setminus \{i\}$;
      // The loop below can be implemented in
         expected $O(1)$ time (see Lemma A.3)
7     for $j = \tau$ to $n$ do
8         Independently with probability $p_i$, set $T_j \leftarrow T_j \cup \{i\}$,
            $\tau_i = j$, and return
9     end
10    $\tau_i = n+1$;
11 function sample()
12    Set $T \leftarrow T_\tau$ and then set $\tau \leftarrow \tau + 1$;
13    if $\tau < n+1$ then call set$(i, p_i)$ for all $i \in T$;
14    else set $\tau = 1$, and then call set$(i, p_i)$ for all $i \in [n]$;
      return : $T$
```

Algorithm 3 is written without an efficient determination of the next output time for each element, so that it is clear that this algorithm is a set sampler. In the following Lemma A.3 we show how to perform this efficient determination or, more precisely, implementing the for-loop in Lines 7-8 We then use this lemma to prove Theorem A.2.

Lemma A.3. *The for-loop in Lines 7-8 of Algorithm 3 can be implemented in expected $O(1)$ time in the word RAM model.*

Proof. The loop executes the return statement with a value of $j \in [t, n]$ with probability $p^{j-t}(1-p)$. Consequently, if we let $\ell \geq 0$ be sampled by the geometric distribution with probability p, i.e., $\Pr[\ell = v] = p^v(1-p)$ for all $v \in \mathbb{Z}_{>0}$, and if $\ell \in \{0, \ldots, n-t\}$ simply execute the return statement with $j = t + \ell$ and otherwise set $\tau_i = n+1$, then this is equivalent to the lines of the for loop. Since sampling from a geometric distribution $\mathrm{Geo}(p)$ in this manner can be implemented in expected $O(\log(1/p)/w) = O(1)$ time in the word RAM model [29], the result follows. □

Proof of Theorem A.2. Algorithm 3 maintains that after each operation (init, set or sample), each $i \in [n]$ is a member of at most one T_j. Further, if $i \in T_j$, then $\tau \leq j$ and $\tau_i = j$ and moreover $\tau_i = n+1$ if and only if $i \notin T_j$ for any $j \in [n]$. Further, the algorithm is designed (as discussed) so that T_τ is a valid output of sample at time τ (for any updates of an output-adaptive adversary, that is unaware of T_j for $j \geq \tau$). Since the algorithm also ensures that $\tau \leq n$, the algorithm has the desired output. Further, from these properties it is clear that the algorithm can be implemented in $O(n)$ space. It only remains to bound the running time for implementing the algorithm.

To analyze the running time of the algorithm, first note that by Lemma A.3, each set operation can be implemented in expected $O(1)$ time. Consequently, init can be implemented in expected $O(n)$ time. Further, since $T = \texttt{sample}()$ simply calls set for elements in its output or for all n elements after every n times it is called, it has the desired expected runtime $O(1 + |T|)$ as well. □

REFERENCES

[1] Amir Abboud and Søren Dahlgaard. 2016. Popular conjectures as a barrier for dynamic planar graph algorithms. In *Proceedings of the 57th Symposium on Foundations of Computer Science.* 477–486.

[2] Amir Abboud and Virginia Vassilevska Williams. 2014. Popular conjectures imply strong lower bounds for dynamic problems. In *Proceedings of the 55th Symposium on Foundations of Computer Science.* 434–443.

[3] Moab Arar, Shiri Chechik, Sarel Cohen, Cliff Stein, and David Wajc. 2018. Dynamic Matching: Reducing Integral Algorithms to Approximately-Maximal Fractional Algorithms. In *Proceedings of the 45th International Colloquium on Automata, Languages and Programming.* 79:1–79:16.

[4] Sepehr Assadi, Soheil Behnezhad, Sanjeev Khanna, and Huan Li. 2023. On regularity lemma and barriers in streaming and dynamic matching. In *Proceedings of the 55th Symposium on Theory of Computing.*

[5] Sepehr Assadi, Aaron Bernstein, and Aditi Dudeja. 2022. Decremental Matching in General Graphs. In *Proceedings of the 49th International Colloquium on Automata, Languages and Programming.* 11:1–11:19.

[6] Amir Azarmehr, Soheil Behnezhad, and Mohammad Roghani. 2024. Fully Dynamic Matching: $(2-\sqrt{2})$-Approximation in Polylog Update Time. In *Proceedings of the 35th Symposium on Discrete Algorithms.*

[7] Surender Baswana, Manoj Gupta, and Sandeep Sen. 2015. Fully Dynamic Maximal Matching in $O(\log n)$ Update Time. *SIAM J. Comput.* 44, 1 (2015), 88–113.

[8] Soheil Behnezhad. 2023. Dynamic algorithms for maximum matching size. In *Proceedings of the 34th Symposium on Discrete Algorithms.* 129–162.

[9] Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Cliff Stein, and Madhu Sudan. 2019. Fully dynamic maximal independent set with polylogarithmic update time. In *Proceedings of the 60th Symposium on Foundations of Computer Science.* 382–405.

[10] Soheil Behnezhad and Sanjeev Khanna. 2022. New Trade-Offs for Fully Dynamic Matching via Hierarchical EDCS. In *Proceedings of the 33rd Symposium on Discrete Algorithms.* 3529–3566.

[11] Soheil Behnezhad, Jakub Łącki, and Vahab Mirrokni. 2020. Fully Dynamic Matching: Beating 2-Approximation in Δ^ϵ Update Time. In *Proceedings of the 31st Symposium on Discrete Algorithms.* 2492–2508.

[12] Amos Beimel, Haim Kaplan, Yishay Mansour, Kobbi Nissim, Thatchaphol Saranurak, and Uri Stemmer. 2022. Dynamic algorithms against an adaptive adversary: Generic constructions and lower bounds. In *Proceedings of the 54th Symposium on Theory of Computing.* 1671–1684.

[13] Aaron Bernstein, Sebastian Forster, and Monika Henzinger. 2019. A deamortization approach for dynamic spanner and dynamic maximal matching. In *Proceedings of the 30th Symposium on Discrete Algorithms.* 1899–1918.

[14] Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. 2020. Deterministic decremental reachability, SCC, and shortest paths via directed expanders and congestion balancing. In *Proceedings of the 61st Symposium on Foundations of Computer Science.* 1123–1134.

[15] Aaron Bernstein and Cliff Stein. 2015. Fully Dynamic Matching in Bipartite Graphs. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming.* 167–179.

[16] Aaron Bernstein and Cliff Stein. 2016. Faster fully dynamic matchings with small approximation ratios. In *Proceedings of the 27th Symposium on Discrete Algorithms.* 692–711.

[17] Sayan Bhattacharya, Deeparnab Chakrabarty, and Monika Henzinger. 2020. Deterministic dynamic matching in $O(1)$ update time. *Algorithmica* 82, 4 (2020), 1057–1080.

[18] Sayan Bhattacharya, Monika Henzinger, and Giuseppe F Italiano. 2015. Deterministic fully dynamic data structures for vertex cover and matching. In *Proceedings of the 26th Symposium on Discrete Algorithms.* 785–804.

[19] Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. 2016. New deterministic approximation algorithms for fully dynamic matching. In *Proceedings of the 48th Symposium on Theory of Computing.* 398–411.

[20] Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. 2017. Fully Dynamic Approximate Maximum Matching and Minimum Vertex Cover in $O(\log^3 n)$ Worst Case Update Time. In *Proceedings of the 28th Symposium on Discrete Algorithms.* 470–489.

[21] Sayan Bhattacharya and Peter Kiss. 2021. Deterministic Rounding of Dynamic Fractional Matchings. In *Proceedings of the 48th International Colloquium on Automata, Languages and Programming.* 27:1–27:14.

[22] Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. 2023. Dynamic $(1+\epsilon)$-Approximate Matching Size in Truly Sublinear Update Time. *Proceedings of the 64th Symposium on Foundations of Computer Science*, 1563–1588.

[23] Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. 2023. Dynamic Algorithms for Packing-Covering LPs via Multiplicative Weight Updates. In *Proceedings of the 34th Symposium on Discrete Algorithms.* 1–47.

[24] Sayan Bhattacharya, Peter Kiss, Thatchaphol Saranurak, and David Wajc. 2023. Dynamic Matching with Better-than-2 Approximation in Polylogarithmic Update Time. In *Proceedings of the 34th Symposium on Discrete Algorithms.* 100–128.

[25] Sayan Bhattacharya, Peter Kiss, Aaron Sidford, and David Wajc. 2024. Near-Optimal Dynamic Rounding of Fractional Matchings in Bipartite Graphs. arXiv:2306.11828 [cs.DS]

[26] Sayan Bhattacharya and Janardhan Kulkarni. 2019. Deterministically Maintaining a $(2+\epsilon)$-Approximate Minimum Vertex Cover in $O(1/\varepsilon^2)$ Amortized Update Time. In *Proceedings of the 30th Symposium on Discrete Algorithms.* 1872–1885.

[27] Joakim Blikstad and Peter Kiss. 2023. Incremental $(1-\epsilon)$-approximate dynamic matching in $O(\mathrm{poly}(1/\epsilon))$ update time. In *Proceedings of the 31st European Symposium on Algorithms.* 22:1–22:19.

[28] Jan van den Brand, Danupon Nanongkai, and Thatchaphol Saranurak. 2019. Dynamic matrix inverse: Improved algorithms and matching conditional lower bounds. In *Proceedings of the 60th Symposium on Foundations of Computer Science.* 456–480.

[29] Karl Bringmann and Tobias Friedrich. 2013. Exact and efficient generation of geometric random variates and random graphs. In *Proceedings of the 40th International Colloquium on Automata, Languages and Programming.* 267–278.

[30] Karl Bringmann and Konstantinos Panagiotou. 2017. Efficient sampling methods for discrete distributions. *Algorithmica* 79 (2017), 484–508.

[31] Moses Charikar and Shay Solomon. 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.* 33:1–33:14.

[32] Shiri Chechik and Tianyi Zhang. 2019. Fully dynamic maximal independent set in expected poly-log update time. In *Proceedings of the 60th Symposium on Foundations of Computer Science.* 370–381.

[33] Li Chen, Rasmus Kyng, Yang P Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. 2022. Maximum flow and minimum-cost flow in almost-linear time. In *Proceedings of the 63rd Symposium on Foundations of Computer Science.*

[34] Julia Chuzhoy and Sanjeev Khanna. 2019. A new algorithm for decremental single-source shortest paths with applications to vertex-capacitated flow and cut problems. In *Proceedings of the 51st Symposium on Theory of Computing.* 389–400.

[35] Søren Dahlgaard. 2016. On the Hardness of Partially Dynamic Graph Problems and Connections to Diameter. In *Proceedings of the 43rd International Colloquium on Automata, Languages and Programming.* 48:1–48:14.

[36] Ran Duan and Seth Pettie. 2014. Linear-time approximation for maximum weight matching. *J. ACM* 61, 1 (2014), 1.

[37] Jack Edmonds. 1965. Maximum matching and a polyhedron with 0, 1-vertices. *Journal of research of the National Bureau of Standards B* 69, 125-130 (1965), 55–56.

[38] Fabrizio Grandoni, Stefano Leonardi, Piotr Sankowski, Chris Schwiegelshohn, and Shay Solomon. 2019. $(1+\epsilon)$-Approximate Incremental Matching in Constant Deterministic Amortized Time. In *Proceedings of the 30th Symposium on Discrete Algorithms.* 1886–1898.

[39] Fabrizio Grandoni, Chris Schwiegelshohn, Shay Solomon, and Amitai Uzrad. 2022. Maintaining an EDCS in General Graphs: Simpler, Density-Sensitive and with Worst-Case Time Bounds. *Proceedings of the 5th Symposium on Simplicity in Algorithms* (2022), 12–23.

[40] Manoj Gupta and Richard Peng. 2013. Fully dynamic $(1+\varepsilon)$-approximate matchings. In *Proceedings of the 54th Symposium on Foundations of Computer Science.* 548–557.

[41] Manoj Gupta, Venkatesh Raman, and SP Suresh. 2014. Maintaining Approximate Maximum Matching in an Incremental Bipartite Graph in Polylogarithmic Update Time. In *Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS)*, Vol. 29. 227–239.

[42] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. 2015. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the 47th Symposium on Theory of Computing.* 21–30.

[43] Zoran Ivkovic and Errol L Lloyd. 1993. Fully Dynamic Maintenance of Vertex Cover. In *Proceedings of the 19th International Workshop on Graph-Theoretic Concepts in Computer Science.* 99–111.

[44] Arun Jambulapati, Yujia Jin, Aaron Sidford, and Kevin Tian. 2022. Regularized Box-Simplex Games and Dynamic Decremental Bipartite Matching. In *Proceedings of the 49th International Colloquium on Automata, Languages and Programming.*

[45] Peter Kiss. 2022. Improving update times of dynamic matching algorithms from amortized to worst case. *Proceedings of the 13th Innovations in Theoretical Computer Science* (2022), 94:1–94:21.

[46] Tsvi Kopelowitz, Seth Pettie, and Ely Porat. 2016. Higher lower bounds from the 3SUM conjecture. In *Proceedings of the 27th Symposium on Discrete Algorithms.* 1272–1287.

[47] Danupon Nanongkai and Thatchaphol Saranurak. 2017. Dynamic spanning forest with worst-case update time: adaptive, Las Vegas, and $O(n^{1/2}-\varepsilon)$-time. In *Proceedings of the 49th Symposium on Theory of Computing.* 1122–1129.

[48] Krzysztof Onak and Ronitt Rubinfeld. 2010. Maintaining a large matching and a small vertex cover. In *Proceedings of the 42nd Symposium on Theory of Computing.* 457–464.

[49] David Peleg and Shay Solomon. 2016. Dynamic $(1+\epsilon)$-approximate matchings: a density-sensitive approach. In *Proceedings of the 27th Symposium on Discrete Algorithms.* 712–729.

[50] Mohammad Roghani, Amin Saberi, and David Wajc. 2022. Beating the Folklore Algorithm for Dynamic Matching. In *Proceedings of the 13th Innovations in Theoretical Computer Science.* 111:1–111:23.

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

[52] Noam Solomon and Shay Solomon. 2021. A Generalized Matching Reconfiguration Problem. In *Proceedings of the 12th Innovations in Theoretical Computer Science.* 57:1–57:20.

[53] Shay Solomon. 2016. Fully dynamic maximal matching in constant update time. In *Proceedings of the 57th Symposium on Foundations of Computer Science.* 325–334.

[54] Meng-Tsung Tsai, Da-Wei Wang, Churn-Jung Liau, and Tsan-sheng Hsu. 2010. Heterogeneous subset sampling. In *Proceedings of the 16th International Computing and Combinatorics Conference.* 500–509.

[55] David Wajc. 2020. Rounding dynamic matchings against an adaptive adversary. In *Proceedings of the 52nd Symposium on Theory of Computing.* 194–207.

[56] Lu Yi, Hanzhi Wang, and Zhewei Wei. 2023. Optimal Dynamic Subset Sampling: Theory and Applications. In *Proceedings of the 29th Knowledge Discovery and Data Mining.*

[57] Da Wei Zheng and Monika Henzinger. 2023. Multiplicative Auction Algorithm for Approximate Maximum Weight Bipartite Matching. In *Proceedings of the 24th Integer Programming and Combinatorial Optimization.*

Received 13-NOV-2023; accepted 2024-02-11