Improved Bounds for Fully Dynamic Matching via Ordered Ruzsa-Szemerédi Graphs

Sepehr Assadi* Sanjeev Khanna† Peter Kiss‡

Abstract

In a very recent breakthrough, Behnezhad and Ghafari [FOCS'24] developed a novel fully dynamic randomized algorithm for maintaining a $(1-\varepsilon)$-approximation of maximum matching with amortized update time *potentially* much better than the trivial $O(n)$ update time. The runtime of the BG algorithm is parameterized via the following graph theoretical concept:

- For any n, define $\mathsf{ORS}(n)$—standing for *Ordered Ruzsa-Szemerédi Graph*—to be the largest number of edge-disjoint matchings $M_1, \ldots, M_t$ of size $\Theta(n)$ in an n-vertex graph such that for every $i \in [t]$, M_i is an induced matching in the subgraph $M_i \cup M_{i+1} \cup \ldots \cup M_t$.

Then, for any fixed $\varepsilon > 0$, the BG algorithm runs in

$$O\left(\sqrt{n^{1+O(\varepsilon)} \cdot \mathsf{ORS}(n)}\right)$$

amortized update time with high probability, even against an adaptive adversary. $\mathsf{ORS}(n)$ is a close variant of a more well-known quantity regarding Ruzsa-Szemerédi graphs (which require every matching to be induced regardless of the ordering). It is currently only known that $n^{o(1)} \leqslant \mathsf{ORS}(n) \leqslant n^{1-o(1)}$, and closing this gap appears to be a notoriously challenging problem.

If it turns out that $\mathsf{ORS}(n) = n^{o(1)}$, namely, the current lower bounds are close to being optimal, then, this algorithm achieves an update time of $n^{1/2+o(1)}$ for $(1-\varepsilon)$-approximation of fully dynamic matching, making progress on a major open question in the area.

Our Result: In this work, we further strengthen the result of Behnezhad and Ghafari and push it to limit to obtain a randomized algorithm with amortized update time of

$$n^{o(1)} \cdot \mathsf{ORS}(n)$$

with high probability, even against an adaptive adversary. In the limit, i.e., *if* current lower bounds for $\mathsf{ORS}(n) = n^{o(1)}$ are almost optimal, our algorithm achieves an $n^{o(1)}$ update time for $(1-\varepsilon)$-approximation of maximum matching, almost fully resolving this fundamental question. In its current stage also, this fully reduces the algorithmic problem of designing dynamic matching algorithms to a purely combinatorial problem of upper bounding $\mathsf{ORS}(n)$ with no algorithmic considerations.

1 Introduction

We study the problem of maintaining an approximate maximum matching in a fully dynamic graph. In this problem, we have a graph $G = (V, E)$ that undergoes insertion and deletion of edges by an adversary and our goal is to maintain (edges of) an approximate maximum matching of G after each update. This is one of the most central problems in the dynamic graph literature; see [OR10, GP13, BS15, BS16, Sol16, BGS18, BK22, Beh23, BKSW23, ABKL23, BKS23, Liu24, BG24] and references therein.

*(sepehr@assadi.info) Cheriton School of Computer Science, University of Waterloo. Supported in part by a Sloan Research Fellowship, an NSERC Discovery Grant, a University of Waterloo startup grant, and a Faculty of Math Research Chair grant.

†† (sanjeev@cis.upenn.edu) Department of Computer and Information Science, University of Pennsylvania. Research supported in part by NSF awards CCF-1934876, CCF-2008305, and CCF-2402284.

‡(peter.kiss@univie.ac.at) Faculty of Computer Science, University of Vienna, Austria, the author completed parts of this project at the University of Warwick.

In a very recent breakthrough, [BG24] developed an algorithm that for any fixed $\varepsilon > 0$, maintains (edges of) a $(1-\varepsilon)$-approximate maximum matching in a fully dynamic graph with *potentially* much better than $O_\varepsilon(n)$ update time. Specifically, the runtime of the algorithm of [BG24] is parameterized based on the density of a certain family of extremal graphs which are (quite) closely related to Ruzsa-Szemerédi (RS) graphs [RS78] (see [AMS12, FHS17] for more context on RS graphs, and [GKK12, ABKL23, AS23] and references therein for their applications to dynamic graph and other sublinear algorithms). [BG24] defined the following family of closely related graphs.

DEFINITION 1.1. (ORDERED RUZSA-SZEMERÉDI (ORS) GRAPHS [BG24]) *A graph $G = (V, E)$ is called an* **(r, t)-ORS graph** *if its edges can be partitioned into an ordered set of t matchings $M_1, \ldots, M_t$ each of size r, such that for every $i \in [t]$, the matching M_i is an induced matching in the subgraph of G on $M_i \cup M_{i+1} \cup \ldots \cup M_t$.*

We define $\mathsf{ORS}(n, r)$ as the largest choice of t such that an n-vertex (r, t)-ORS graph exist.

Unfortunately, exactly as in RS graphs, density of ORS graphs is quite poorly understood at this point. Currently, for any constant $\delta \in (0, 1/4)$, it is only known that

$$n^{\Omega_\delta(1/\log\log n)} \underset{[\text{FLN}^+02, \text{GKK12}]}{\leqslant} \mathsf{ORS}(n, \delta n) \underset{[\text{BG24}]}{\leqslant} \frac{n}{\log^{(\text{poly}(1/\delta))}(n)}, \tag{1.1}$$

where $\log^{(k)}(n)$ is the k-iterated logarithm function, i.e.,

$$\log^{(k)}(n) := \underbrace{\log\log\cdots\log}_{k}(n).$$

This is quite similar to the situation for RS graphs (modulo a slightly better dependence in RS graphs on the parameter δ in the upper bound due to [Fox11] (see also [FHS17]), namely, $\log^{O(\log(1/\delta))}(n)$ instead in the denominator).

The result of [BG24] is a randomized algorithm that given any fixed $\varepsilon > 0$ with high probability maintains a $(1-\varepsilon)$-approximation of maximum matching in a fully dynamic graph with an amortized update time of

$$O\Big(\sqrt{n^{1+\varepsilon} \cdot \mathsf{ORS}(n, \Theta_\varepsilon(n))}\Big).$$

Thus, *if* it happens to be the case that ORS graphs cannot be dense, i.e., $\mathsf{ORS}(n, \Theta_\varepsilon(n)) = n^{1-\Omega(1)}$, this algorithm achieves an update time of $n^{1-\Omega(1)}$ for this problem, making progress on a major open question in the dynamic matching literature [GP13, BS16, BK22, BKS23, BG24]. Moreover, in the limit, namely, if the current lower bounds of Equation (1.1) on ORS are almost optimal, then, this algorithm achieves an update time of $n^{1/2+o(1)}$; currently, the best algorithm known with such an update time due to [BS16] can only achieve a 2/3-approximation.

1.1 Our Contribution We build on the approach of [BG24] and push it to its limit to obtain the following result.

RESULT 1. *There is an algorithm that for any fixed $\varepsilon > 0$ maintains a $(1-\varepsilon)$-approximate maximum matching of any fully dynamic graph with amortized update time of*

$$O\Big(n^{o(1)} \cdot \mathsf{ORS}(n, \Theta_\varepsilon(n))\Big).^{[a]}$$

The algorithm is randomized and its guarantees hold with high probability against an adaptive adversary. The algorithm does not assume a prior knowledge of the value of $\mathsf{ORS}(n, \Theta_\varepsilon(n))$ to achieve its guarantee.

[a] More specifically, for any $\beta_0 \in (0, 1)$, there is a $\beta_1 \in (0, 1)$ such that this runtime is $O(n^{\beta_0} \cdot \mathsf{ORS}(n, \beta_1 \cdot f(\varepsilon) \cdot n))$ for some fixed function f independent of β_0, β_1.

In the limit, *if* ORS graphs cannot be much denser than the lower bounds in Equation (1.1), Result 1 achieves an $n^{o(1)}$ amortized update time, almost fully settling the question of $(1-\varepsilon)$-approximation of fully dynamic matching. Beside the conditional upper bound of [BG24] (which would be an $n^{1/2+o(1)}$ update time algorithm under this hypothesis), it is also known unconditionally how to obtain an update time of $n/2^{\Theta(\sqrt{\log n})} \cdot \text{poly}(1/\varepsilon)$

on bipartite graphs [Liu24] (and presumably a similar runtime with $O(1/\varepsilon)^{O(1/\varepsilon)}$-dependence instead on general graphs using the reduction of [McG05]; see Proposition 2.1). Moreover, [BKS23], building on [Beh23, BKSW23], designed an algorithm that for any fixed $\varepsilon > 0$, obtains an update time of $m^{1/2-\Omega_\varepsilon(1)}$ for the easier problem of maintaining the *size* of the maximum matching (but not its edges).

Result 1 also suggests that almost any interesting lower bound for this problem (even under computational hardness assumptions) should effectively rule out existence of even mildly dense ORS graphs, which will constitute a big breakthrough given the close connection of these graphs to RS graphs (or alternatively, explicitly condition on the assumption that ORS graphs are dense; similar assumptions for RS graphs have been used to prove lower bounds for $(1-\varepsilon)$-approximation of the maximum matching problem in other settings, e.g., in the streaming model [AS23]).

It is worth noting here that existing conditional lower bounds in [HKNS15] rule out $n^{1-\Omega(1)}$ update time algorithms for computing an *exact* maximum matching, and more recently in [Liu24], for even a $(1-\varepsilon)$-approximation but only when $\varepsilon = n^{-\Omega(1)}$. On the other hand, our focus in Result 1 is on the regime when $\varepsilon \in (0,1)$ is fixed and independent of n (which is often the main regime of interest in the context of sublinear algorithms). Indeed, the RS graph constructions of [RS78] (see also [AMS12]) imply that $\mathsf{ORS}(n, \delta n) = \Omega(n)$ for $\delta \leqslant 2^{-\Theta(\sqrt{\log n})}$. This implies that for small enough ε, it is already known that Result 1 cannot achieve any non-trivial guarantee.

1.2 Our Algorithm at a High Level By the existing boosting frameworks for matchings (see Proposition 2.1), obtaining an εn *additive* approximation reduces to the following problem: given a fully dynamic graph $G = (V, E)$, every $\Theta_\varepsilon(n)$ updates we receive $O_\varepsilon(1)$ queries of the form $U \subseteq V$ and must return an $O(1)$-approximate maximum matching in the induced subgraph $G[U]$ (see Problem 4.1). An additive εn approximation to matching can also be turned into a multiplicative one, using another standard technique (see Proposition 2.3). Both these parts are by-now standard; see, e.g. [Kis22, Beh23, BKSW23, BKS23, ABR24, Liu24, BG24]. The main part then is to solve Problem 4.1.

The approach of [BG24]. The solution of [BG24] for Problem 4.1 can be summarized as follows. The algorithm processes the updates in *batches*. In each batch, G_0 will be the current graph at the start of the batch and the *new* updates are inserted into a new graph G_1. There will also be a graph G_2 which is created by moving certain edges from G_0 (will be described later). Given a query U, the algorithm tries to find a large matching in $(G_1 \cup G_2)[U]$ and if it succeeds, it returns that one and moves on. This step is done using the greedy matching algorithm taking time linear in the size of $G_1 \cup G_2$. But, if $(G_1 \cup G_2)[U]$ does not have a large matching, the algorithm needs to search for a one in $G_0[U]$. This is done via a novel *sublinear time* "opportunistic" algorithm: the algorithm takes $O(n^{2+\varepsilon}/d)$ time with high probability where d is the *average* degree of the graph $G_0[V(M)]$; i.e., if the matching M is "far from" being induced, then it can be found much faster than when it is close to being induced. The edges in this matching are then moved from G_0 to G_2.

The runtime analysis of this algorithm is as follows. Suppose each batch consists of s updates. Since $O_\varepsilon(1)$ queries are called every $\Theta_\varepsilon(n)$ updates, the algorithm needs to handle $O_\varepsilon(s/n)$ queries in a batch. Moreover, both graphs G_1 and G_2 can only have $O_\varepsilon(s)$ edges: the first one since there are s updates in a batch and the second because each of the $O_\varepsilon(s/n)$ queries may insert a matching inside G_2. Thus, the entire time spent in this batch for running the greedy algorithm on $G_1 \cup G_2$ is $O_\varepsilon(s^2/n)$ time. The runtime on G_0 however is calculated differently using a global argument. Using the fact that G_0 only undergoes deletions during a batch, [BG24] come up with an elegant analysis that shows that if the algorithm is spending a "lot of time" in finding "near induced" matchings in G_0, then, one can find a "dense" ORS graph in G_0 – this allows for bounding the entire time the algorithm is spending on G_0 during this batch by $O(n^{2+\varepsilon} \cdot \mathsf{ORS}(n, \Theta_\varepsilon(n)))$ time. This implies that by taking $s \approx n^{3/2} \cdot \mathsf{ORS}(n, \Theta_\varepsilon(n))^{1/2}$, the amortized update time of the algorithm will become $O_\varepsilon(s/n)$ (over s updates) which is $O_\varepsilon(\sqrt{n^{1+\varepsilon} \cdot \mathsf{ORS}(n, \Theta_\varepsilon(n))})$ time.

Our approach. In the algorithm of [BG24], if the size of batch grows then so do the number of edges in the graph $G_1 \cup G_2$, and hence the cost of finding a greedy matching in $(G_1 \cup G_2)[U]$. However, the ORS-density based upper bound of the total running time of calls to the opportunistic algorithm remains unchanged with larger batches. Our main goal is to accelerate the step of finding matchings in $G_1 \cup G_2$ to allow for larger batches to amortize over the running cost of the opportunistic algorithm. To do this, observe that G_1 and G_2 are both dynamic graphs undergoing a small number of updates per each update to the underlying graph. This suggests that instead of statically finding matchings in $G_1 \cup G_2$ we may dynamically maintain them via an efficient dynamic

matching algorithm.

We develop a *recursive* variant of the algorithm of [BG24]. The first main ingredient is a sublinear time algorithm that given a graph $G = (V, E)$ with m edges and a query $U \subseteq V$, finds a large matching M in G in $O(m^{1+\varepsilon}/d)$ time where d is the *maximum* degree of M in $G[V(M)]$. Thus, this result strengthens the algorithm of [BG24] in both relating it to the density of G (instead of $O(n^2)$ always) and providing a stronger guarantee on the maximum internal degree of M instead of the average degree. The main step however is how to perform the recursion.

We present a family of algorithms $\{\mathbb{A}_i\}_{i \geqslant 1}$ with progressively better update times, where $\mathbb{A}_1$ is similar to the algorithm of [BG24] with a key difference of having $O_\varepsilon(\sqrt{(m/n)^{1+\varepsilon} \cdot \mathsf{ORS}(n, \Theta_\varepsilon(n))})$ update time instead (here, m is a promised upper bound on the number of edges in G). Let us now consider constructing $\mathbb{A}_2$ from $\mathbb{A}_1$.

We also process the inputs in batches of size s with G_0 being the current graph at the start of the batch, G_1 receiving the updates during the batch, and G_2 receiving some removed matchings from G_0 while answering the queries. The main difference is that we are going to run $\mathbb{A}_1$ on G_1 and G_2 separately instead of the greedy algorithm. These graphs now are going to have $O_\varepsilon(s)$ edges in total (similar to what argued earlier) and thus the total runtime of processing these graphs is $O_\varepsilon(s \cdot \sqrt{(s/n)^{1+\varepsilon} \cdot \mathsf{ORS}(n, \Theta_\varepsilon(n))})$. If we find a large matching for a given query U from either G_1 or G_2, we will be done, but if not, we need to rely on $G_0[U]$. In this case, we run our new sublinear time algorithm to find a matching with maximum internal degree d in $O(m^{1+\varepsilon}/d)$ time. A similar argument as in [BG24] (in fact considerably simpler given our stronger maximum degree guarantee) allows us to bound the total runtime of this step with $O_\varepsilon(m^{1+\varepsilon} \cdot \mathsf{ORS}(n, \Theta_\varepsilon(n)))$ time. Optimizing for the choice of s then leads to an $O_\varepsilon((m/n)^{1/3+\varepsilon} \cdot \mathsf{ORS}(n, \Theta_\varepsilon(n))^{2/3})$ amortized update time.

Continuing like this for $\mathbb{A}_3, \mathbb{A}_4, \cdots$, while explicitly accounting for the loss in parameters (especially size of induced matchings in ORS graphs), gives us our final algorithm.

2 Preliminaries

2.1 Basic Notation and Representation of Graphs

For a graph $G = (V, E)$, we use $n := |V|$ and $m := |E|$. For a vertex $v \in V$, we use $N(v)$ to denote the neighbors of v and $\deg(v) := |N(v)|$ to denote its degree. For a subset $U \subseteq V$, $G[U]$ denotes the induced subgraph of G on U. We use $\mu(G)$ to denote the maximum matching size in G.

Since we will be designing sublinear-time algorithms for a dynamically changing graph, we briefly describe how the graphs will be represented to support various operations, namely, insertion and deletion of edges, neighbor queries, and pair queries, each in $O(1)$ expected time. For each vertex $u \in V$, we maintain a *dynamic* array A_u, a *dynamic* hash table h_u, and the current degree $\deg(u)$ of u. Here, dynamic refers to the property that at all times, the size of A_u is $\Theta(\deg(u))$.

When an edge (u, v) is inserted, we increment $\deg(u)$ by 1, set $A_u[\deg(u)] = v$, and insert v in the hash table h_u, storing along with it it the value $\deg(u)$, namely, the location of the vertex v in the array A_u. This takes $O(1)$ expected time.

When an edge (u, v) is deleted, let w be the vertex at $A_u[\deg(u)]$. We decrement $\deg(u)$ by 1. If $w = v$, we simply delete v from the hash table h_u. Otherwise, let j be the location of the vertex v in A_u which can be recovered using $h_u(v)$. We set $A_u[j] = w$, delete both v and w from h_u, and reinsert w in h_u associating j to be its new location in A_u. This takes $O(1)$ expected time.

Finally, given any integer $i \in [\deg(u)]$, the task of outputting the i-th neighbor of u is done by simply returning the vertex stored in $A_u[i]$. This also allows for sampling a random neighbor of u.

2.2 Tools from Prior Work

We start by recalling the following standard fact about the greedy algorithm for approximating maximum matchings.

FACT 2.1. *Let $G = (V, E)$. The greedy algorithm that starts with $M = \emptyset$, iterates over edges of G in any arbitrary order, and add an edge to M if both its endpoints are currently unmatched, returns a matching M of size $|M| \geqslant 1/2 \cdot \mu(G)$ in $O(n + m)$ time.*

Boosting frameworks for approximate matching. We use standard boosting frameworks for obtaining a $(1 - \varepsilon)$-approximation to matching, using a "weak" approximation algorithm that only returns an $O(1)$-approximation. The original version of this framework is due to [McG05] and was de-randomized in [Tir18];

for bipartite graphs, more efficient reductions are known in [AG11, ALT21]. These results were tailored to additive approximation and dynamic graphs in [BKS23].

PROPOSITION 2.1. ([McG05, AG11, Tir18, ALT21, BKS23]) *Let $\gamma, \varepsilon \in (0,1)$ be parameters. There exist functions $f(\gamma,\varepsilon)$ and $g(\gamma,\varepsilon)$ such that the following holds. Let $\mathbb{A}_{\text{weak}}$ be an algorithm that given an n-vertex graph $G=(V,E)$ and any set $U \subseteq V$ of vertices with $\mu(G[U]) \geqslant f(\gamma,\varepsilon)\cdot n$, returns a matching of size at least $\gamma \cdot f(\gamma,\varepsilon)\cdot n$ in $G[U]$. Then, there is a algorithm that given $G=(V,E)$ makes $g(\gamma,\varepsilon)$ calls to $\mathbb{A}_{\text{weak}}$ on adaptively chosen subsets of vertices, spending $O(f(\gamma,\varepsilon)\cdot n)$ time preparing each subset, and returns a matching of size $\mu(G)-\varepsilon\cdot n$ in G.*

For bipartite graphs, $f(\gamma,\varepsilon)=\text{poly}(\varepsilon)$ and $g(\gamma,\varepsilon)=\text{poly}(1/(\gamma\cdot\varepsilon))$ by [AG11, ALT21], while for general graphs, both $f(\gamma,\varepsilon)^{-1}, g(\gamma,\varepsilon) = (1/(\gamma\cdot\varepsilon))^{O(1/(\gamma\cdot\varepsilon))} = O_{\gamma,\varepsilon}(1)$ by [McG05, Tir18].

Sublinear-time estimation of maximum matching size. We also rely on the sublinear-time algorithm of [Beh21] for matching *size* estimation.

PROPOSITION 2.2. ([Beh21]) *There is a randomized algorithm $\mathbb{A}_{\text{sublinear}}$ that for any n-vertex graph $G=(V,E)$ and a parameter $\varepsilon \in (0,1)$, makes $\widetilde{O}(n\cdot\text{poly}(1/\varepsilon))$ queries to the adjacency matrix of G and with high probability outputs an estimate $\tilde{\mu}(G)$ such that*

$$\frac{1}{2}\cdot\mu(G)-\varepsilon\cdot n \leqslant \tilde{\mu}(G) \leqslant \mu(G).$$

Vertex sparsification. Finally, we use vertex sparsification approaches of [AKLY16, AKL16, CCE+16] as implemented by [Kis22] for dynamic graphs. These sparsification approaches reduce the number of vertices to $O(\mu(G)/\varepsilon)$ via vertex contraction while preserving a $(1-\varepsilon)$-approximate maximum matching in the graph. This allows one to turn additive approximation to matching into a multiplicative one, with minor overhead (see, e.g. [BG24], for an example of how this is used).

PROPOSITION 2.3. ([AKLY16, AKL16, CCE+16, Kis22]) *Suppose $\mathbb{A}_{\text{additive}}$ is an algorithm that given a parameter $\varepsilon > 0$ can process a fully dynamic n-vertex graph $G=(V,E)$ and maintains a matching of size at least $\mu(G)-\varepsilon\cdot n$ in $T(n,\varepsilon)$ amortized update time. Then, there is a randomized algorithm that can with high probability maintain a $(1-\varepsilon)$-approximation to maximum matching in n-vertex fully dynamic graphs in $O(T(n,\Theta(\varepsilon^2))\cdot\text{poly}(\log(n)/\varepsilon))$ amortized update time.*

2.3 An Auxiliary Lemma on ORS Graphs For a matching M in a graph G, we define the **maximum internal degree** $\Delta_{\text{IN}}(M)$ of M, as the maximum degree of the induced subgraph $G[V(M)]$. This way, $\Delta_{\text{IN}}(M)=1$ iff M is an induced matching, and in general, smaller the value $\Delta_{\text{IN}}(M)$, the "closer" M is to an induced matching.

The following lemma is a simplification of a similar result in [BG24, Lemma 14] (which even works for average internal degree instead of maximum degree), using a simple variant of the rounding approach of [GKK12] for RS graphs.

LEMMA 2.1. *Let $G=(V,E)$ be any graph and $\mathcal{M}:=(M_1,\ldots,M_\rho)$ be an ordered set of matchings in G, each of size ℓ, such that for every $i\in[\rho]$, $\Delta_{\text{IN}}(M_i)$ in the subgraph of G on edges $M_i\cup M_{i+1}\cup\ldots\cup M_\rho$ is some given $d_i \geqslant 1$. Then, for every $\eta\in(0,1/100)$,*

$$\sum_{i=1}^{\rho}\frac{1}{d_i} \leqslant \frac{34\log n}{\eta}\cdot\mathsf{ORS}(n,(1-\eta)\cdot\ell).$$

Proof. For any integer $d\in\{1,2,4,\ldots,n\}$, define $\mathcal{M}(d):=\{M_i\in\mathcal{M}\mid d\leqslant d_i<2d_i\}$. Moreover, let

$$d^* := \arg\max_d \sum_{M_i\in\mathcal{M}(d)} 1/d_i.$$

Since $\mathcal{M}(1),\mathcal{M}(2),\cdots$ partition $\mathcal{M}$, we have that

$$|\mathcal{M}(d^*)| = d^*\cdot\sum_{M_i\in\mathcal{M}(d^*)}\frac{1}{d^*} \geqslant d^*\cdot\sum_{M_i\in\mathcal{M}(d)}\frac{1}{d_i} \geqslant \frac{d^*}{\log n}\cdot\sum_{i=1}^{\rho}\frac{1}{d_i}. \tag{2.2}$$

We now show that a random subset of $\mathcal{M}(d^*)$ indeed forms an ORS graph with induced matchings of size $(1-\eta)\cdot\ell$ which will be enough to conclude the proof.

Pick $\mathcal{M}^* = (M_1, \ldots, M_{\rho'}) \subseteq \mathcal{M}(d^*)$ wherein each matching of $\mathcal{M}(d^*)$ is chosen with probability

$$p := \frac{\eta}{20d^*}$$

independently (with the sampled matchings ordered in the same manner as they were in $\mathcal{M}$). Fix any matching M_i in $\mathcal{M}^*$. For any vertex v matched by M_i, define $\deg_i^*(v)$ as the degree of v in the subgraph $M_{i+1}, \ldots, M_{\rho'} \in \mathcal{M}^*$ (notice that this *excludes* degree of v in M_i itself). We have,

$$\mathbb{E}\left[\deg_i^*(v)\right] = \sum_{\substack{j>i\in\mathcal{M}(d^*)\\ v \text{ has an edge in } M_j}} \Pr\left(M_j \text{ is chosen in } \mathcal{M}^*\right) \leqslant 2d^* \cdot \frac{\eta}{20d^*} = \frac{\eta}{10},$$

where in the inequality uses the fact that degree of all vertices in $\mathcal{M}(d^*)$ in the suffix matchings is at most $2d^*$ in the entire $\mathcal{M}$ (and thus among $\mathcal{M}(d^*)$ for sure).

We now say that v is **bad** for M_i iff $\deg_i^*(v) \geqslant 1$. By the above calculation and a Markov bound, the probability that v is bad for M_i is at most $\eta/10$. This means that the expected number of bad vertices for M_i is at most $2\ell\cdot\eta/10 = \ell\cdot\eta/5$.

We further say that the matching M_i itself is **bad** if it has at least $\eta\cdot\ell$ bad vertices. Another application of Markov bound implies that the probability M_i is bad is at most $1/5$. This means that in expectation, at least $4/5$ of the sampled matchings in $\mathcal{M}^*$ are not bad.

By the probabilistic method (and since the size of $\mathcal{M}^*$ is concentrated), there exist a set of $3/5\cdot p\cdot|\mathcal{M}(d^*)|$ matchings in $\mathcal{M}^*$ none of which are bad. For each of these matchings, remove all their bad vertices which reduces their size to $(1-\eta)\cdot\ell$ in the worst case and arbitrarily remove more edges such that all of them have size $(1-\eta)\cdot\ell$ exactly. By definition, the resulting graph is now an (r,t)-ORS graph with parameters

$$r = (1-\eta)\cdot\ell, \quad \text{and} \quad t = \frac{3}{5}\cdot\frac{\eta}{20d^*}\cdot|\mathcal{M}(d^*)|\,.$$

By definition, we have $t \leqslant \mathsf{ORS}(n, (1-\eta)\cdot\ell)$ which implies that

$$|\mathcal{M}(d^*)| \leqslant \frac{34d^*}{\eta}\cdot\mathsf{ORS}(n,(1-\eta)\cdot\ell).$$

Plugging in this bound in Equation (2.2) concludes the proof. ∎

3 An Opportunistic Sublinear-Time Algorithm for Matching

In this section, we provide one of the key subroutines used by our dynamic algorithm. This subroutine takes as input a "base" *static* graph G and a set U with the promise that $G[U]$ is of size $\Omega(n)$, and outputs a matching of size $\Omega(n)$ in $G[U]$. While the worst-case runtime of this algorithm is (almost) linear in the number of edges of the base graph, it can be much better if the maximum internal degree of the matching it outputs is large.

LEMMA 3.1. *There is an algorithm (Algorithm 1) that given an n-vertex graph $G=(V,E)$ with m edges, parameters $\gamma,\delta\in(0,1/6)$, and vertices $U\subseteq V$ with $\mu(G[U])\geqslant\delta n$, with high probability returns a matching M in $G[U]$ with size at least $\gamma\cdot\delta n$ in $O(m\cdot n^{3\gamma}\cdot\log(n)/\Delta_{\text{IN}}(M))$ time.*

We note that the *worst-case* bound of $\Omega(m)$ in Lemma 3.1 is necessary due to a lower bound of [ACK19] (as opposed to Proposition 2.2 for *size* estimation); however, in the hard instances of that lower bound, one necessarily needs to find a large matching with maximum internal degree $O(1)$, which means, one cannot benefit from the extra power of this lemma (as is expected given the lower bound).

This result is inspired by [BG24, Lemma 9] and generalizes and strengthens it. Algorithm of [BG24] finds a matching of size $\gamma\cdot\delta n$ in $O(n^{2+O(\gamma)}/d)$ time (thus, does not benefit from the number of edges of G) and moreover d is the *average* degree of $G[V(M)]$ instead of our stronger guarantee on maximum degree. To obtain our result, we use an argument similar in spirit to that of "residual sparsity guarantee" of the greedy algorithm used for maximal independent set and maximal matching problems, e.g., in [ACG+15, Kon18, AOSS19].

The algorithm in Lemma 3.1 works by sampling the edges of G with geometrically increasing probabilities and only consider edges of G that are sampled and belong to $G[U]$. It then attempts to find a "large" matching in this sampled graph using the greedy algorithm; if it succeeds, it returns *this* matching only, otherwise, it will remove vertices of this matching and continues to the next sampling phase. Formally, the algorithm is as follows.

Algorithm 1 The algorithm of Lemma 3.1

Input: A graph $G(V, E)$, a set $U \subseteq V$, s.t. $\mu(G[U]) \geq \delta n$, and $\gamma, \delta \in (0, 1/6)$
Output: A matching M in $G[U]$ of size at least $\gamma \cdot \delta n$
Let $X_1 = U$
for $i = 1$ to $\Gamma := 1/(3\gamma)$ **do**
  Let the sampling probability be $p_i := n^{(3\gamma)\cdot i}/n$
  For every vertex $v \in X_i$, pick a set $N_i(v) \subseteq N(v)$ by sampling each neighbor of v in G independently
  Start with $M_i = \emptyset$
  for $v \in X_i$ **do**
    if v and some w in $N_i(v) \cap X_i$ are unmatched by M_i **then**
      Add (v, w) to M_i
  if If $|M_i| \geqslant \gamma \cdot \delta n$ **then**
    Return M_i and **Terminate**
  else
    $X_{i+1} = X_i \setminus V(M_i)$ and continue to the next sampling step

We start by arguing that the algorithm always returns a matching in one of its iterations.

Claim 3.1. *Algorithm 1 always returns a matching M_i of size at least $\gamma \cdot \delta n$ in some $i \in [\Gamma]$.*

Proof. Suppose the algorithm has not terminated until the beginning of the last iteration. This means that for all $i < \Gamma$, we have, $|M_i| < \gamma \cdot \delta n$. Thus, the total number of vertices from U that are removed until reaching X_Γ is

$$\sum_{i=1}^{\Gamma-1} |V(M_i)| < \Gamma \cdot 2 \cdot (\gamma \cdot \delta n) = 2\delta n/3,$$

given $\Gamma = 1/(3\gamma)$. Given that $G[U]$ has a matching of size at least δn by the theorem statement, there is still a matching of size at least $\delta n - 2\delta n/3 = \delta n/3$ inside $G[X_\Gamma]$ (after removing the vertices counted above). But, in iteration $i = \Gamma$, every edge is sampled with probability $n^{3\gamma \cdot 1/(3\gamma)}/n = 1$ and thus $N_i(v)$ is the entire neighborhood of v in $G[X_\Gamma]$. Hence, the greedy algorithm necessarily finds a matching of size at least $\delta n/6 > \gamma \cdot \delta n$ in this case (since $\gamma < 1/6$). Thus the algorithm terminates in this last iteration and outputs M_Γ as the answer. ∎

The next step is to bound the runtime of the algorithm based on the iteration it terminates in.

Claim 3.2. *With high probability, for every $i \in [\Gamma]$, if Algorithm 1 returns the matching M_i and terminates in this iteration, then its runtime is $O(m \cdot p_i)$.*

Proof. Firstly, with high probability, the runtime of the algorithm in each iteration $j \in [\Gamma]$ is $O(m \cdot p_j)$. This is because with high probability, the number of edges in G_j is $O(m \cdot p_j)$ by a simple application of Chernoff bound, and the edges can be sampled in this much time using standard ideas instead of explicitly going over each edge and sampling them[1]. Running the greedy matching algorithm also take another $O(m \cdot p_j)$ time now. Finally, since p_j's form a geometric series (as $p_{j+1} = n^{3\gamma} \cdot p_j$), we have $\sum_{j=1}^{i} p_j = O(p_i)$ which concludes the proof. ∎

We now bound $\Delta_{\text{IN}}(M_i)$ for the matching M_i returned by the algorithm. Instead of bounding the maximum internal degree of the matching itself, we simply bound the maximum degree of the subgraph $G[X_i]$ where the matching M_i is chosen from.

[1]For each vertex v, first sample a number k_v from the binomial distribution of $\deg(v)$ and p_i (using its closed-form formula); then, sample k_v neighbors of v uniformly at random from $N(v)$.

CLAIM 3.3. *With high probability, for every $i \in [\Gamma]$, max-degree of $G[X_i]$ is $O(n^{3\gamma} \cdot \log(n)/p_i)$.*

Proof. The claim trivially holds for $i = 1$ since $p_1 = n^{3\gamma}/n$ and $G[X_1]$ can only have maximum degree $n < n \log(n) = n^{3\gamma} \log(n)/p_1$. We focus on the $i > 1$ case in the following.

Fix any vertex v in $G[X_{i-1}]$. Consider the step wherein v is being processed by the greedy algorithm. First, suppose that the degree of v to vertices in $X_{i-1} \setminus V(M_{i-1})$ at this point is at most $100 \ln(n)/p_{i-1}$. In this case, even if v remains unmatched, its degree in $G[X_i]$ will be $O(\log(n)/p_{i-1}) = O(n^{3\gamma} \cdot \log(n)/p_i)$ as desired.

On the other hand, consider the case where degree of v to vertices of $X_{i-1} \setminus V(M_{i-1})$ is at least $100 \ln(n)/p_{i-1}$ when we start processing v. Then, the probability that none of these neighbors are sampled in $N_i(v)$ is at most

$$(1 - p_{i-1})^{100 \log(n)/p_{i-1}} \leqslant e^{-100 \ln n} = n^{-100};$$

thus, with high probability, at least one of these vertices is sampled in $N_i(v)$. Conditioned on this event, v will surely get matched (if it was not already matched) by the greedy algorithm. Taking a union bound over all vertices now ensures that with high probability, every vertex in X_{i-1} that remains unmatched by M_{i-1} has degree $O(n^{3\gamma} \cdot \log(n)/p_i)$ to other unmatched vertices in X_{i-1}, and hence in the graph $G[X_i]$. This concludes the proof. ∎

We are now ready to conclude the proof of Lemma 3.1.

Proof. [Proof of Lemma 3.1] By Claim 3.1, we know Algorithm 1 terminates in some iteration $i \in [\Gamma]$ and returns a matching M_i of size at least $\gamma \cdot \delta n$ by the termination condition. By Claim 3.2, this takes $O(m \cdot p_i)$ time with high probability.

Finally, since $V(M_i)$ is a subset of X_i, we obtain that $\Delta_{\text{IN}}(M_i)$ is at most the maximum degree of $G[X_i]$ which itself is at most $O(n^{3\gamma} \cdot \log(n)/p_i)$ with high probability by Claim 3.3. Thus, the runtime of the algorithm is $O(m \cdot n^{3\gamma} \cdot \log(n)/\Delta_{\text{IN}}(M_i))$ with high probability as desired. ∎

REMARK 3.1. One can also prove an alternate form of Lemma 3.1 where the runtime of Algorithm 1 is $O(m \cdot n^{3\gamma}/\bar{d})$ but $\bar{d}$ is now the *average* internal degree of the returned matching instead of the maximum, and the failure probability is now exponentially small. This means a weaker guarantee in terms of average degree instead of maximum degree, but a stronger guarantee on the probability of success of the algorithm.

This is done by proving that the subgraph $G[X_i]$ in Algorithm 1 only has $O(n^{1+3\gamma}/p_i)$ edges by a union bound over all possible subgraphs of $G[X_{i-1}]$ (this corresponds to replacing the "residual sparsity guarantee" of [ACG+15, Kon18, AOSS19] in the above argument by the "filtering" technique of [LMSV11] instead).

4 A Key Intermediate Dynamic Problem

In order to prove our main result, we focus on solving the following intermediate problem. Similar versions of this problem also appear in recent work including [BKS23, Liu24, BG24].

PROBLEM 4.1. *The problem is parameterized by integers $n, m, q \geqslant 1$ and reals $\gamma, \delta, \alpha \in (0, 1)$. We have a fully dynamic n-vertex graph $G = (V, E)$ that starts empty, i.e., has $E = \emptyset$, and throughout, never has more than m edges, nor receives more than $\text{poly}(n)$ updates in total.*

Updates: *The updates to G happen in* **chunks** *$C_1, C_2, \ldots$, each consisting of exactly $\alpha \cdot n$ edge insertions or deletions in G.*

Queries: *After each chunk, there will be at most q queries, coming one at a time and in an adaptive manner (based on the answer to all prior queries including the ones in this chunk). Each query is a set $U \subseteq V$ of vertices with the promise that $\mu(G[U]) \geqslant \delta n$; the algorithm should respond with a matching of size at least $\gamma \cdot \delta n$ from $\mu(G[U])$.*

For ease of reference, we list the parameters of this problem and their definitions:

n : *number of vertices in the graph*;

m : *maximum number of edges at any point present in the graph*;

q : *number of adaptive queries made after each chunk*;

γ : *approximation ratio of the returned matching for each query*;

δ : *a lower bound on the fraction of vertices matched in the subgraph of G for the query*;

α : *a parameter for determining the size of each chunk as a function of n.*

For technical reasons, we allow additional updates, called **empty updates** *to also appear in the chunks but these "updates" do not change any edge of the graph, although will be counted toward the number of updates in their chunks*[2].

We will design a family of recursive algorithms for solving Problem 4.1 in this section, starting with the base case, which also acts as a good warm-up for the key ideas of the algorithm.

4.1 Base Case The proof of the following lemma follows a similar approach as used in [BG24] albeit with several modifications to take into account the dependence on the (overall) sparsity of the input graph and to match the requirements of Problem 4.1. This lemma effectively gives an algorithm with $\approx \sqrt{(m/n \cdot \mathsf{ORS}(n, \Theta(n)))}$ update time for solving Problem 4.1.

LEMMA 4.1. *There is an algorithm (Algorithm 2) for Problem 4.1 that with high probability takes*

$$O\Big(q \cdot \sqrt{\frac{m \cdot n^{6\gamma} \cdot \mathsf{ORS}(n, \gamma \cdot \delta n/2)}{\alpha \cdot n}}\Big),$$

amortized time over the updates to maintain the answer to all given queries in each chunk. The algorithm works as long as $\gamma < 1/12$ *and* $\alpha \geqslant \gamma \cdot \delta$.

The algorithm in Lemma 4.1 processes the chunks in **batches**. Each batch B processes t chunks of updates to G for some t to be fixed later (t is going to be $\approx \sqrt{(m/n \cdot \mathsf{ORS}(n, \Theta(n)))}$). Whenever a batch starts, the algorithm maintains threes graphs (see also Figure 1 for an illustration):

- G_{old} which starts as the graph G at the beginning of the batch; no further insertions will be added to G_{old} during the processing of this batch and if an edge already in G_{old} is deleted in an update, the algorithm removes the edge from G_{old} (i.e., G_{old} is a decremental graph);

- G_{batch} which starts as an empty graph and receives all subsequent insertion of edges to G during the updates of this batch and will be updated based on their deletions also;

- G_{match} which starts as an empty graph and is updated by the algorithm by moving certain matchings from G_{old} to G_{match} instead. Once an edge is moved to G_{match}, if it gets deleted, the algorithm deletes the edge from G_{match} (and subsequent insertions are processed in G_{batch}; in other words, insertions to G_{match} only come from moving edges from G_{old} to G_{match}).

Given a query $U \subseteq V$, the algorithm starts by examining the edges of G_{batch} to see if it can already find a large matching in $G_{\text{batch}}[U]$. This is done by running the greedy matching algorithm (in Fact 2.1) over the edges of $G_{\text{batch}}[U]$ to obtain a matching M_{batch}. If M_{batch} is large enough, it will be returned as the answer to the query. Otherwise, the algorithm runs the greedy matching algorithm on $G_{\text{match}}[U]$ to obtain a matching M_{match}. Again, if this matching is large enough, it will be returned as the answer to the query. Finally, if neither of these cases happen, then the algorithm runs Algorithm 1 on G_{old} with the subgraph U to obtain a matching M_{old}; we can guarantee this matching is large enough given $G[U]$ is promised to have a large matching. The algorithm then moves all edges of M_{old} from G_{old} to G_{match} and returns M_{old} as the answer to the query.

A formal specification of the algorithm is as follows.

[2]This is used for simplifying the exposition when solving this problem recursively; these empty updates will still be counted when computing amortized runtime of these recursive algorithms.

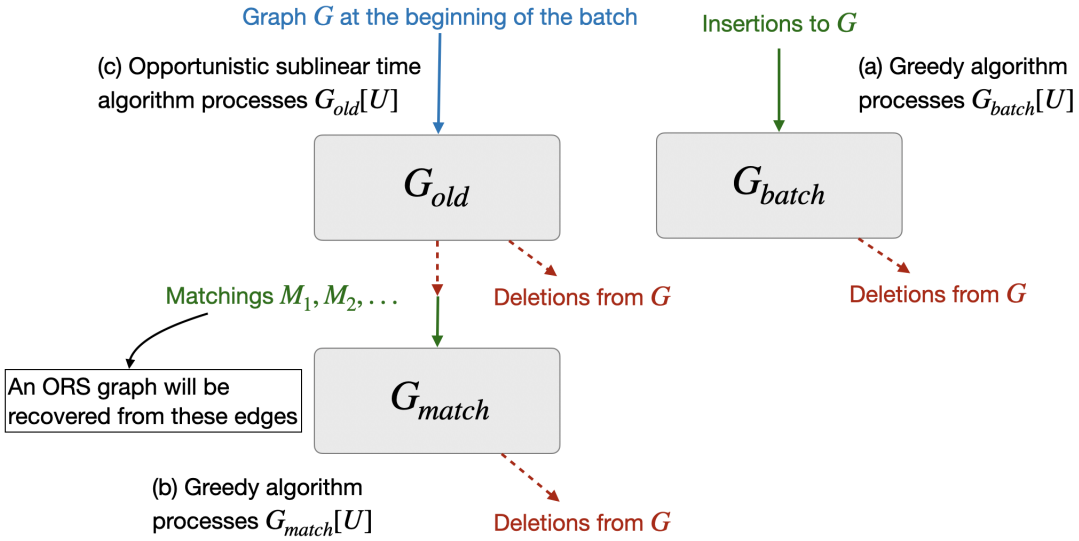

Figure 1: An illustration of the three graphs $G_{\text{old}}, G_{\text{batch}}, G_{\text{match}}$ in Algorithm 2, their role, and how they are being processed. Notice that G_{old} is a decremental graph, while $G_{\text{batch}}, G_{\text{match}}$ are fully dynamic. The analysis of the algorithm forms an ORS from the edges of the matchings $M_1, M_2, \ldots$, moved from G_{old} to G_{match} – this ORS is a *subgraph* of the *static* graph G at the beginning of the batch and does not contain any edges inserted in this batch.

Algorithm 2 Algorithm of Lemma 4.1

Process the updates in batches B of t chunks $C_1, \ldots, C_t$
for Each Batch **do**
$G_{\text{old}} = G$, $G_{\text{batch}} = \emptyset$, and $G_{\text{match}} = \emptyset$ (on vertices V)
for Updates in each chunk **do**
For an edge insertion $e = (u, v)$, add the edge to G_{batch}
For an edge deletion $e = (u, v)$, remove the edge e from each of the graphs G_{old}, G_{batch}, or G_{match} that it belongs to currently
for Query $U \subseteq V$ **do**
Go over all edges of G_{batch} and run the greedy matching algorithm on $G_{\text{batch}}[U]$ to obtain a matching M_{batch}. If $|M_{\text{batch}}| \geqslant \gamma \cdot \delta n$, return M_{batch}, otherwise continue.
Go over all edges of G_{match} and run the greedy matching algorithm on $G_{\text{match}}[U]$ to obtain a matching M_{match}. If $|M_{\text{match}}| \geqslant \gamma \cdot \delta n$, return M_{match}, otherwise continue.
Run Algorithm 1 on G_{old}, the set U, and parameter 2γ with the guarantee that $\mu(G_{\text{old}}[U]) \geqslant \delta n/2$ (which we establish in Claim 4.1) to obtain a matching M_{old}. Move M_{old} from G_{old} to G_{match} and return M_{old}.

We start by establishing the correctness of Algorithm 2.

CLAIM 4.1. *With high probability, the answer to each query U in Algorithm 2 is a valid answer according to Problem 4.1.*

Proof. Notice that $G_{\text{old}}, G_{\text{batch}}, G_{\text{match}}$ at any point partition the current graph G. If either of M_{batch} or M_{match} is of size at least $\gamma \cdot \delta n$, the output is correct. Otherwise, by the guarantee of the greedy algorithm, we know that both $\mu(G_{\text{batch}}[U]), \mu(G_{\text{match}}[U])$ are at most $2\gamma \cdot \delta n$. Thus,

$$\mu(G_{\text{old}}[U]) \geqslant \mu(G[U]) - \mu(G_{\text{batch}}[U]) - \mu(G_{\text{match}}[U]) \geqslant \delta n - 4\gamma \cdot \delta n \geqslant \delta n/2,$$

by the choice of $\gamma < 1/12$. This implies that the requirement of Lemma 3.1 for $G_{\text{old}}[U]$ is satisfied (including

having $2\gamma < 1/6$) and thus, its output, with high probability, is of size $2\gamma \cdot \delta n/2 = \gamma \cdot \delta n$. Thus, the returned matching in this case is also of the proper size, concluding the proof. ∎

The main part is to analyze the running time of Algorithm 2. The following claim is the key to relating the runtime of this algorithm to the density of ORS graphs.

CLAIM 4.2. *Let $M_1, M_2, \ldots, M_\rho$ be the matchings computed from G_{old} in Line (10) of Algorithm 2 and added to G_{match} at the time of their computation (i.e., here, we ignore the deletions that have happened subsequently, namely, some edges of M_i might have been deleted from G_{old} when we are inserting M_{i+1}, but we still keep those edges in the definition of M_i). These matchings are edge-disjoint and for every $i \in [\rho]$, maximum degree of M_i among the matchings $M_i, \ldots, M_\rho$ is at most $\Delta_{\text{IN}}(M_i)$ in the graph G_{old} at the time M_i was computed.*

Proof. The edge-disjoint part follows from the fact that each M_i is chosen from G_{old} and at that point, edges of $M_1, \ldots, M_{i-1}$ are already removed from G_{old} (and cannot be inserted back to G_{old}).

Fix any $i \in [\rho]$ and matching M_i. Since G_{old} is a decremental graph, all edges in $M_{i+1}, \ldots, M_\rho$ belong to G_{old} at the time of computation of M_i. Thus, these edges will be counted toward $\Delta_{\text{IN}}(M_i)$ in G_{old} at the time of computation of M_i. As such, the maximum degree of M_i among the matchings $M_i, \ldots, M_\rho$ is at most $\Delta_{\text{IN}}(M_i)$ as desired. ∎

We can now bound the runtime of the algorithm.

CLAIM 4.3. *With high probability, when running Algorithm 2 on a single batch of t chunks:*

1. *the total time spent for maintaining the graphs and bookkeeping is $O(t \cdot q \cdot \alpha \cdot n)$ time;*
2. *the total time spent computing M_{batch} in Line (8) is $O(t \cdot q \cdot t \cdot \alpha \cdot n)$ time;*
3. *the total time spent computing M_{match} in Line (9) is $O(t \cdot q \cdot t \cdot q \cdot \alpha \cdot n)$ time;*
4. *the total time spent computing M_{old} in Line (10) is $O(m \cdot n^{6\gamma} \cdot \log^2(n) \cdot \mathsf{ORS}(n, \gamma \cdot \delta n/2))$ time.*

Proof. There are t chunks in a batch, each involving αn updates and q queries. Processing the updates can be done in $O(1)$ expected time for each update by maintaining the three graphs $G_{\text{old}}, G_{\text{batch}}, G_{\text{match}}$ as explained in Section 2.1. Thus, with high probability, these steps take $O(t \cdot q \cdot \alpha \cdot n)$ time in total. Finally, given $\alpha \geqslant \gamma \cdot \delta$ in Lemma 4.1, moving each choice of M_{old} from G_{old} to G_{match} takes $O(\alpha \cdot n)$ time per each query (at most), and thus $O(t \cdot q \cdot \alpha \cdot n)$ in total.

For computing M_{batch} for each query, the algorithm iterates over edges of G_{batch} and takes linear time in the size of the entire G_{batch} to run the greedy matching algorithm (on $G_{\text{batch}}[U]$). Given that G_{batch} can only have $\leqslant t \cdot \alpha \cdot n$ edges at any point (before a new batch is restarted), the runtime for each query is $O(t \cdot \alpha \cdot n)$ time. Given there are $t \cdot q$ queries in total, this part takes $O(t \cdot q \cdot t \cdot \alpha \cdot n)$ time.

Similarly, computing M_{match} for each query is done by iterating over all edges of G_{match} and taking linear time on those. The edges in G_{match} come from inserting a matching of size $\gamma \cdot \delta n \leqslant \alpha n$ (by the assumption in the statement of Lemma 4.1) after a query (possibly) and since there are most $t \cdot q$ queries, there can be at most $O(t \cdot q \cdot \alpha \cdot n)$ edges in G_{match}. Thus, similar (but not identical) to the previous case, this step takes $O(t \cdot q \cdot t \cdot q \cdot \alpha \cdot n)$ time (this is a factor q larger).

We now get to the main part of bounding the runtime of computing M_{old}. Let $M_1, M_2, \ldots, M_\rho$ be the matchings computed as M_{old} throughout this entire batch. For each $i \in [\rho]$, let $\Delta_{\text{IN}}(M_i)$ denote the maximum degree of M_i in G_{old} at the time M_i was computed; additionally, let d_i denote the maximum degree of M_i among the matchings $M_i, \ldots, M_\rho$. By Claim 4.2, we have $\Delta_{\text{IN}}(M_i) \geqslant d_i$. Moreover, by Lemma 3.1, the runtime for computing M_i in Algorithm 1 with high probability is $O(m \cdot n^{6\gamma} \cdot \log(n)/\Delta_{\text{IN}}(M_i))$. Putting these together with Claim 4.2 in Lemma 2.1 (for parameter $\eta = 1/2$, and since size of each matching is $\gamma \cdot \delta n$ by Claim 4.2), we have that the total time spent computing $M_1, \ldots, M_\rho$ is

$$\sum_{i=1}^{\rho} O(m \cdot n^{6\gamma} \cdot \log(n) \cdot \frac{1}{\Delta_{\text{IN}}(M_i)}) \leqslant O(m \cdot n^{6\gamma} \cdot \log(n)) \cdot \sum_{i=1}^{\rho} \frac{1}{d_i}$$
$$\leqslant O(m \cdot n^{6\gamma} \cdot \log^2(n) \cdot \mathsf{ORS}(n, \gamma \cdot \delta n/2)).$$

This concludes the proof. ∎

Proof. [Proof of Lemma 4.1] The correctness of the algorithm follows from Claim 4.1 and a union bound over $\text{poly}(n)$ intermediate graphs created in Problem 4.1 (by the assumption on number of updates).

Furthermore, the amortized runtime per each of $t \cdot \alpha \cdot n$ updates during a batch, by Claim 4.3 is

$$O(t \cdot q^2) + O(\frac{1}{t \cdot \alpha \cdot n} \cdot m \cdot n^{6\gamma} \cdot \log^2(n) \cdot \mathsf{ORS}(n, \gamma \cdot \delta n/2)).$$

We can now balance these terms by setting

$$t := \left(\frac{m \cdot n^{6\gamma} \cdot \mathsf{ORS}(n, \gamma \cdot \delta n/2)}{\alpha \cdot n \cdot q^2}\right)^{1/2},$$

which leads to the desired update time of

$$O(q) \cdot \left(\frac{m \cdot n^{6\gamma} \cdot \mathsf{ORS}(n, \gamma \cdot \delta n/2)}{\alpha \cdot n}\right)^{1/2}.$$

Note however that it is possible the *entire* number of updates to Problem 4.1 is less than t, namely, the algorithm does not receive even one full batch of updates. In that case, we cannot amortize the runtime as above given the fewer number of updates.

Nevertheless, since by the definition of Problem 4.1, the graph starts as an empty graph, in this case both $G_{\text{old}} = \emptyset$ and $G_{\text{match}} = \emptyset$ for the single batch processed by the algorithm. Thus, the entire runtime of the algorithm will be based on the first two items of Claim 4.3 and thus is still upper bounded as above. ∎

4.2 The Recursive Step We now design a family of recursive algorithms $\{\mathbb{A}\}_{k=1}^{\infty}$ with progressively better update times for Problem 4.1 using Algorithm 2 as the base case of this family (i.e., $\mathbb{A}_1$). Roughly speaking, the algorithm $\mathbb{A}_k$ in this family achieves an update time $\approx (m/n)^{1/(k+1)} \cdot \mathsf{ORS}(n, \Theta_k(n))^{1-1/(k+1)}$.

LEMMA 4.2. *There exists an absolute constant $c \geqslant 1$ such that the following holds. For any $k \geqslant 1$, there is an algorithm $\mathbb{A}_k(n, m, q, \gamma, \delta, \alpha)$ (Algorithm 3) for Problem 4.1 that with high probability takes*

$$O\Big((2q)^{k-1} \cdot \left(\frac{m}{n}\right)^{1/k+1} \cdot \mathsf{ORS}(n, \gamma \cdot \delta n/2)^{1-1/(k+1)} \cdot n^{6\gamma} \cdot (\log(n)/\delta)^c\Big),$$

amortized time over the updates to maintain the answer to all given queries. The algorithm works as long as $\gamma < (1/12)^k$ and $\alpha \geqslant \gamma \cdot \delta$.

We prove Lemma 4.2 by induction. The base case is handled by Lemma 4.1 for $\mathbb{A}_1$. Now, suppose the lemma is true for some $k \geqslant 1$ and we prove it for $k+1$. The algorithm $\mathbb{A}_{k+1}$ follows the same approach of Algorithm 2 in processing the graph in batches of t chunks of size $\alpha_{k+1} \cdot n_{k+1}$ for some t to be determined later (it is going to be $\approx (m/n)^{k+1/(k+2)} \cdot \mathsf{ORS}(n, \Theta(n))^{1/(k+2)}$). In each batch, the algorithm also partitions the graph into three subgraphs $G_{\text{old}}, G_{\text{batch}}, G_{\text{match}}$ with very similar definitions as in the past. The main difference however is that both graphs G_{batch} and G_{match} are now handled by running algorithm $\mathbb{A}_k$ over them (instead of the greedy approach of Algorithm 2).

We first specify the parameters used for running $\mathbb{A}_k$ on G_{batch} and G_{match}:

$$\begin{aligned} &n_k := n && m_k := t \cdot q \cdot \alpha \cdot n && q_k := q \\ &\gamma_k := 12 \cdot \gamma && \delta_k := \delta/12 && \alpha_k := \alpha. \end{aligned} \tag{4.3}$$

We will run $\mathbb{A}_k(n_k, m_k, q_k, \gamma_k, \delta_k, \alpha_k)$ on G_{batch} and G_{match}. By the assumption in Lemma 4.2, we have $\gamma < (1/12)^{k+1}$ and thus $\gamma_k = 12 \cdot \gamma < (1/12)^k$; hence we satisfy the condition for invoking the induction hypothesis of $\mathbb{A}_k$. Similarly, we have $\alpha = \alpha_k$ and $\gamma_k \cdot \delta_k = \gamma \cdot \delta$ and so $\alpha_k \geqslant \gamma_k \cdot \delta_k$ also holds. Finally, at the beginning of each batch, $G_{\text{batch}}, G_{\text{match}}$ are both empty graphs and thus satisfy the promise of Problem 4.1. As such, we can indeed apply the induction hypothesis to $\mathbb{A}_k$ in the following (the only remaining part we need to explicitly account for is to make sure that for each query U, we are guaranteed that $\mu(G_{\text{batch}}[U])$ is at least

$\delta_k \cdot n$, before calling $\mathbb{A}_k$ on G_{batch} to be consistent with the definition of Problem 4.1 (similarly for G_{match}); we use Proposition 2.2 for ensuring this guarantee). This is formalized in Algorithm 3.

The following algorithm follows the same strategy of Algorithm 2 modulo applying $\mathbb{A}_k$ to G_{batch} and G_{match} instead of running the greedy algorithm over them (see also Figure 2).

Algorithm 3 Algorithm $\mathbb{A}_{k+1}(n, m, q, \gamma, \delta, \alpha)$ of Lemma 4.2

Process the updates in batches B of t chunks $C_1, \ldots, C_t$

for each batch **do**

Let $G_{\text{old}} = G$, $G_{\text{batch}} = \emptyset$, and $G_{\text{match}} = \emptyset$ (on vertices V). Start two copies of $\mathbb{A}_k(n_k, m_k, q_k, \gamma_k, \delta_k, \alpha_k)$ on G_{batch} and G_{match} separately with the parameters in Equation (4.3).

for Updates in each chunk (these graphs may also be updated based on queries) **do**

For an edge insertion $e = (u, v)$, add the edge to G_{batch}.

For an edge deletion $e = (u, v)$, remove the edge e from any of the graphs G_{old}, G_{batch}, or G_{match} that it belongs to currently.

for Each query $U \subseteq V$ after a chunk is updated **do**

Run Proposition 2.2 on $G_{\text{batch}}[U]$ with parameter $\varepsilon = (\delta_k/2)$ to obtain an estimate $\tilde{\mu} := \tilde{\mu}(G_{\text{batch}}[U])$ of $\mu(G_{\text{batch}}[U])$. If $\tilde{\mu} \geqslant \delta_k \cdot n$, pass the query U to $\mathbb{A}_k$ on G_{batch} and return its output matching M_{batch} as the answer; otherwise, continue.

Run Proposition 2.2 on $G_{\text{match}}[U]$ with parameter $\varepsilon = (\delta_k/2)$ to obtain an estimate $\tilde{\mu} := \tilde{\mu}(G_{\text{match}}[U])$ of $\mu(G_{\text{match}}[U])$. If $\tilde{\mu} \geqslant \delta_k \cdot n$, pass the query U to $\mathbb{A}_k$ on G_{match} and return its output matching M_{match} as the answer; otherwise, continue.

Run Algorithm 1 on G_{old}, the set U, and parameter 2γ with the guarantee that $\mu(G_{\text{old}}[U]) \geqslant \delta n/2$ (which we establish in Claim 4.5) to obtain a matching M_{old}. Remove M_{old} from G_{old} and insert it into G_{match}.

A remark about the updates in Algorithm 3 is in order. Firstly, when processing an arriving chunk, to update G_{batch} or G_{match}, we create two separate chunks of size $\alpha_k \cdot n_k$ based on these updates for $\mathbb{A}_k$ on G_{batch} and G_{match} separately (appending them with empty updates if needed, to have length exactly $\alpha_k \cdot n_k$). We then pass these chunks to each algorithm as their updates. The updates to G_{old} are done directly. Moreover, in Line (10), we insert M_{old} of size $\gamma \cdot \delta n \leqslant \alpha_k \cdot n_k$ (by the guarantee of Lemma 4.2 and choice of parameters in Equation (4.3)) as a single chunk (possibly with empty updates) to $\mathbb{A}_k$ running on G_{match} (there will be no queries after these chunks for $\mathbb{A}_k$).

We first ensure that the subroutine calls in Algorithm 3 are all valid.

Claim 4.4. *With high probability, when running Algorithm 3 on a single batch of t chunks:*

1. *G_{batch} starts empty, at any point it has at most m_k edges, and in total it receives t chunks of size $\alpha_k \cdot n_k$ each; moreover, each query U to $\mathbb{A}_k$ on G_{batch} satisfies $\mu(G_{\text{batch}}[U]) \geqslant \delta_k \cdot n$;*
2. *G_{match} starts empty, at any point it has at most m_k edges, and in total it receives at most $t \cdot (q+1)$ chunks of size $\alpha_k \cdot n_k$; moreover, each query U to $\mathbb{A}_k$ on G_{match} satisfies $\mu(G_{\text{match}}[U]) \geqslant \delta_k \cdot n$;*
3. *$G_{\text{old}}, G_{\text{batch}}, G_{\text{match}}$ at any point partition the edges of G.*

Proof. We prove each part as follows:

1. At the beginning of the batch, we have $G_{\text{batch}} = \emptyset$ and for each chunk as input to Algorithm 3, $\mathbb{A}_k$ also receives a chunk of size $\alpha_k \cdot n_k = \alpha \cdot n$ as an update (possibly with empty updates; recall that G_{batch} only processes insertions in the batch and deletions of the edges inserted during the batch). Since there are t chunks in each batch of Algorithm 3, there will be at most $t \cdot \alpha \cdot n$ edge insertions to G_{batch} which is equal to $m_k/q \leqslant m_k$ by Equation (4.3). Moreover, running Proposition 2.2 with high probability, returns $\tilde{\mu}(G_{\text{batch}}[U]) \leqslant \mu(G_{\text{batch}}[U])$ and thus when Algorithm 3 decides to query $\mathbb{A}_k$ on G_{batch}, we have $\mu(G_{\text{batch}}[U]) \geqslant \delta_k \cdot n$.
2. At the beginning of the batch $G_{\text{batch}} = \emptyset$ and for each chunk as input to Algorithm 3, $\mathbb{A}_k$ also receives a chunk of size $\alpha_k \cdot n_k = \alpha \cdot n$ as an update, which can only include deletions and empty updates. Moreover,

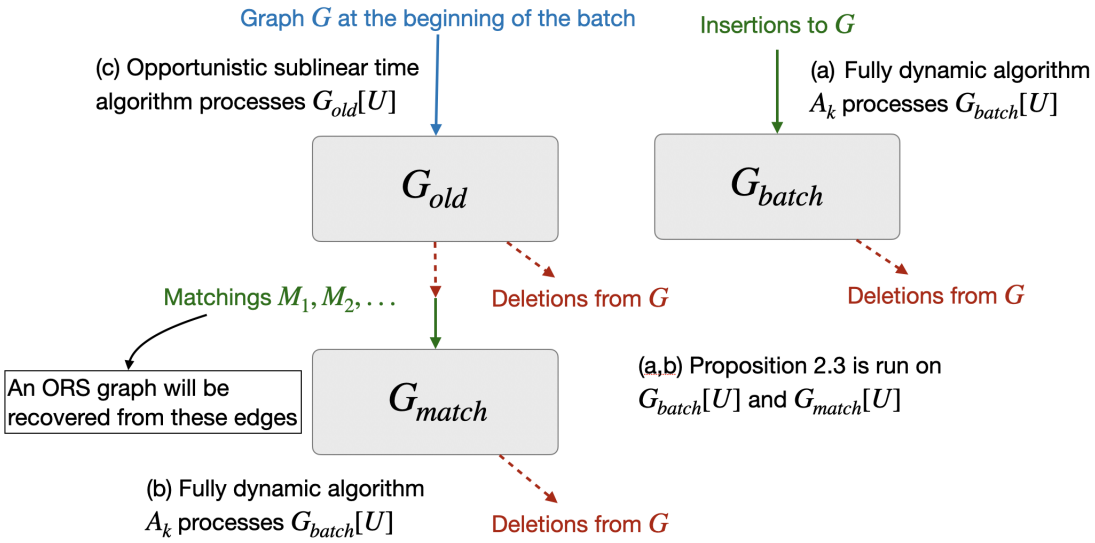

Figure 2: An illustration of the three graphs $G_{\text{old}}, G_{\text{batch}}, G_{\text{match}}$ in Algorithm 3 for $\mathbb{A}_{k+1}$, their role, and how they are being processed. Notice that G_{old} is a decremental graph, while $G_{\text{batch}}, G_{\text{match}}$ are fully dynamic. The main difference with Algorithm 2 is that G_{batch} and G_{match} are now being handled recursively with $\mathbb{A}_k$ (steps (a) and (b) also now involve running Proposition 2.2 to check if applying $\mathbb{A}_k$ is valid). This algorithm also form an ORS from the edges of the matchings $M_1, M_2, \ldots$, moved from G_{old} to G_{match}.

each time the algorithm reaches Line (10), it will be inserting edges of a matching M_{old} as updates in chunks of size $\alpha_k \cdot n_k$. Given that M_{old} is of size $\gamma \cdot \delta n \leqslant \alpha n$ (by the technical assumption on γ, δ, α in Problem 4.1), this translates into having one more chunk here as well. Thus, for each chunk and each of its q queries, we may insert another chunk of size $\alpha_k \cdot n$ into G_{match}, implying that the total number of inserted chunks is $t \cdot (q+1)$. Moreover, running Proposition 2.2 with high probability, returns $\tilde{\mu}(G_{\text{match}}[U]) \leqslant \mu(G_{\text{match}}[U])$ and thus when Algorithm 3 decides to query $\mathbb{A}_k$ on G_{match}, we have $\mu(G_{\text{match}}[U]) \geqslant \delta_k \cdot n$.

3. This step simply follows from the above (on validity of updating edges of G_{batch} and G_{match}) and since we only remove an edge from G_{old} if it is deleted or if it is moved to G_{match}.

□

We can now establish the correctness of Algorithm 3.

CLAIM 4.5. *With high probability, the answer to each query U in Algorithm 3 is a valid answer according to Problem 4.1.*

Proof. Claim 4.4 ensures that each query to recursive calls on $\mathbb{A}_k$ on G_{batch} and G_{match} returns a valid answer (given all promises required by Problem 4.1 and the induction hypothesis of Lemma 4.2 for running this algorithm are satisfied). Thus, it remains to consider the case when Algorithm 3 reaches Line (10) to answer the query U.

Recall that we have the assumption $\mu(G[U]) \geqslant \delta n$. At the same time, each call to Proposition 2.2 in Lines (8) and (9) guarantees that, respectively,

$$\tilde{\mu}(G_{\text{batch}}[U]) \geqslant \frac{1}{2} \cdot \mu(G_{\text{batch}}[U]) - \frac{1}{2} \cdot \delta_k \cdot n,$$
$$\tilde{\mu}(G_{\text{match}}[U]) \geqslant \frac{1}{2} \cdot \mu(G_{\text{match}}[U]) - \frac{1}{2} \cdot \delta_k \cdot n.$$

Thus, if the algorithm has reached Line (10), we know that

$$\mu(G_{\text{batch}}[U]) \leqslant 2\tilde{\mu}(G_{\text{batch}}[U]) + \delta_k \cdot n \leqslant 3 \cdot \delta_k \cdot n,$$
$$\mu(G_{\text{match}}[U]) \leqslant 2\tilde{\mu}(G_{\text{match}}[U]) + \delta_k \cdot n \leqslant 3 \cdot \delta_k \cdot n.$$

We further have that $G_{\text{old}}, G_{\text{batch}}, G_{\text{match}}$ at any point partition G by Claim 4.4. Thus,

$$\mu(G_{\text{old}}[U]) \geqslant \mu(G[U]) - \mu(G_{\text{batch}}[U]) - \mu(G_{\text{match}}[U]) \geqslant \delta n - 6 \cdot \delta_k n \geqslant \delta n/2,$$

by the choice of $\delta_k = \delta/12$ in Equation (4.3). This implies that $G_{\text{old}}[U]$ satisfies the guarantee of Lemma 3.1 for Algorithm 1 with parameters $(\delta/2)$ and (2γ). As such, this algorithm, with high probability, returns a matching M_{old} of size $\gamma \cdot \delta n$ from $G_{\text{old}}[U]$, concluding the proof. ∎

The following claim is a direct analogue of Claim 4.2. Its proof is verbatim as before and hence is omitted.

CLAIM 4.6. *Let $M_1, M_2, \ldots, M_\rho$ be the matchings computed from G_{old} in Line (10) of Algorithm 3 and added to G_{match} at the time of their computation (i.e., here, we ignore the deletions that have happened subsequently, namely, some edges of M_i might have been deleted from G_{old} when we are inserting M_{i+1}, but we still keep those edges in the definition of M_i). These matchings are edge-disjoint and for every $i \in [\rho]$, maximum degree of M_i among the matchings $M_i, \ldots, M_\rho$ is at most $\Delta_{\text{IN}}(M_i)$ in the graph G_{old} at the time M_i was computed.*

The last part is then to bound the runtime of the algorithm.

CLAIM 4.7. *With high probability, when running Algorithm 3 on a single batch of t chunks:*

1. *the total time spent for maintaining the graphs and bookkeeping is*
$$O(t \cdot \alpha \cdot n);$$
2. *the total time spent for running Proposition 2.2 in Lines (8) and (9) is*
$$O(t \cdot q \cdot n \cdot \text{poly}(\log(n)/\delta));$$
3. *the total time spent computing M_{batch} in Line (8) is*
$$O\Big(t \cdot \alpha \cdot n \cdot (2q)^{k-1} \cdot \Big(t \cdot q \cdot \alpha\Big)^{1/(k+1)} \cdot \mathsf{ORS}(n, \gamma \cdot \delta n/2)^{1-1/(k+1)} \cdot n^{6\gamma} \cdot \log^2(n)\Big);$$
4. *the total time spent computing M_{match} in Line (9) is at most*
$$O\Big(t \cdot (q+1) \cdot \alpha \cdot n \cdot (2q)^{k-1}\Big(t \cdot q \cdot \alpha\Big)^{1/(k+1)} \cdot \mathsf{ORS}(n, \gamma \cdot \delta n/2)^{1-1/(k+1)} \cdot n^{6\gamma} \cdot \log^2(n)\Big);$$
5. *the total time spent computing M_{old} in Line (10) is at most*
$$O\Big(m \cdot \mathsf{ORS}(n, \gamma \cdot \delta n/2) \cdot n^{6\gamma} \cdot \log^2(n)\Big).$$

Proof. The proof just follows the same argument as in Claim 4.3.

Specifically, the first part follows immediately, and the second part is by Proposition 2.2. For parts three and four, plugging the choice of $m_k = t \cdot q \cdot \alpha \cdot n$ when applying the induction hypothesis of Lemma 4.2 for $\mathbb{A}_k$ on G_{batch} and G_{match}, implies the bounds.

Finally, the last part holds by Claim 4.6 and Lemma 2.1 exactly as in Claim 4.3 as here also, each matching M_{old} is of size $\gamma \cdot \delta n$ and is chosen from G_{old} which is a decremental graph throughout the batch. ∎

Proof. [Proof of Lemma 4.2] The correctness of the algorithm follows from Claim 4.5 and a union bound over $\text{poly}(n)$ intermediate graphs created in Problem 4.1 (by the assumption on number of updates).

Furthermore, the runtime per each of $t \cdot \alpha \cdot n$ updates during a batch, by Claim 4.7, is at most

$$\begin{aligned}
&O(t \cdot \alpha \cdot n) \\
&+ O(t \cdot q \cdot n \cdot \text{poly}(\log(n)/\delta)) \\
&+ O\Big(t \cdot \alpha \cdot n \cdot (2q)^{k-1} \cdot \Big(t \cdot q \cdot \alpha\Big)^{1/(k+1)} \cdot \mathsf{ORS}(n, \gamma \cdot \delta n/2)^{1-1/(k+1)} \cdot n^{6\gamma} \cdot \log^2(n)\Big) \\
&+ O\Big(t \cdot (q+1) \cdot \alpha \cdot n \cdot (2q)^{k-1}\Big(t \cdot q \cdot \alpha\Big)^{1/(k+1)} \cdot \mathsf{ORS}(n, \gamma \cdot \delta n/2)^{1-1/(k+1)} \cdot n^{6\gamma} \cdot \log^2(n)\Big) \\
&+ O\Big(m \cdot \mathsf{ORS}(n, \gamma \cdot \delta n/2) \cdot n^{6\gamma} \cdot \log^2(n)\Big) \\
&= O\Big(t \cdot (2q) \cdot \alpha \cdot n \cdot (2q)^{k-1}\Big(t \cdot (2q) \cdot \alpha\Big)^{1/(k+1)} \cdot \mathsf{ORS}(n, \gamma \cdot \delta n/2)^{1-1/(k+1)} \cdot n^{6\gamma} \cdot (\log(n)/\delta)^c\Big) \\
&+ O\Big(m \cdot \mathsf{ORS}(n, \gamma \cdot \delta n/2) \cdot n^{6\gamma} \cdot (\log(n)/\delta)^c\Big),
\end{aligned}$$

where in the equality, we used several loose upper bounds (to simplify the subsequent calculations) and use c as the absolute constant which is equal to the exponent of the poly-term in Proposition 2.2 (we also take $c > 2$ to subsume the $\log^2(n)$ term of prior equations).

We can now balance these terms by setting

$$t := \Big(\frac{m}{n}\Big)^{k+1/(k+2)} \cdot \mathsf{ORS}(n, \gamma \cdot \delta n/2)^{1/(k+2)} \cdot \frac{1}{(2q)^{\frac{(k-1)\cdot(k+1)+(k+2)}{k+2}} \cdot \alpha},$$

which leads to the amortized update time of

$$\begin{aligned}
&O\Big(m \cdot \mathsf{ORS}(n, \gamma \cdot \delta n/2) \cdot n^{6\gamma} \cdot (\log(n)/\delta)^c \cdot \frac{1}{t \cdot \alpha \cdot n}\Big) \\
&\quad = O\Big(\Big(\frac{m}{n}\Big)^{1/(k+2)} \cdot \mathsf{ORS}(n, \gamma \cdot \delta n/2)^{1-1/(k+2)} \cdot (2q)^k \cdot n^{6\gamma} \cdot (\log(n)/\delta)^c\Big),
\end{aligned}$$

where we used $(k-1)\cdot(k+1) + (k+2) = k^2 + k + 1 \leqslant k \cdot (k+2)$ for $k \geqslant 1$.

Similar to the proof of Lemma 4.1, we should also handle the case wherein the total number of chunks given to the algorithm does not even reach a single batch. As before, in this case, given the promise that the graph G starts empty, the only graph that is non-empty will be G_{batch}, and thus the amortized runtime of the algorithm is the same as $\mathbb{A}_k$ on G_{batch} (with the given parameters, in particular $m_k = t \cdot q \cdot \alpha \cdot n$). Thus, the amortized runtime of Algorithm 3 will be at most (by Claim 4.7 for bookkeeping, running Proposition 2.2[3] and running $\mathbb{A}_k$ on G_{batch}),

$$\begin{aligned}
&O\Big((2q)^{k-1} \cdot \Big(t \cdot q \cdot \alpha\Big)^{1/(k+1)} \cdot \mathsf{ORS}(n, \gamma \cdot \delta n/2)^{1-1/(k+1)} \cdot n^{6\gamma} \cdot (\log(n)/\delta)^c\Big) \\
&\quad = O\Big(\Big(\frac{m}{n}\Big)^{1/(k+2)} \cdot \mathsf{ORS}(n, \gamma \cdot \delta n/2)^{1-1/(k+2)} \cdot (2q)^{k-1} \cdot n^{6\gamma} \cdot (\log(n)/\delta)^c\Big),
\end{aligned}$$

by the choice of t (using the same exact calculation and the above step).

This proves the induction step of Lemma 4.2 and concludes the proof. ∎

5 A Fully Dynamic Algorithm for Maximum Matching

The following theorem, which is the main contribution of our work, formalizes Result 1.

THEOREM 5.1. *Let $\varepsilon \in (0, 1/100)$ be a given parameter and $k \geqslant 1$ be any integer. Let $\gamma = (1/20)^k$ and $f(\gamma, \varepsilon/4)$ and $g(\gamma, \varepsilon/4)$ be as defined in Proposition 2.1.*

[3] In fact, for the first batch, we do not even need to run Proposition 2.2, given that we know $\mu(G[U]) \geqslant \delta n$ (by the promise of Problem 4.1 and since we know $G_{\text{old}} = G_{\text{match}} = \emptyset$. We ignore this extra optimization step since it does not affect the overall runtime of the algorithm.

There exists an algorithm for maintaining a $(1-\varepsilon)$-approximation to maximum matching in a fully dynamic n-vertex graph that starts empty with amortized update time of

$$O\Big(n^{1/k+1}\cdot \mathsf{ORS}(n\,,\,\frac{1}{15^k}\cdot f(\gamma,\Theta(\varepsilon)^2)\cdot n)^{1-1/(k+1)}\cdot n^{15/(20)^k}\Big).$$

The guarantees of this algorithm hold with high probability even against an adaptive adversary.

Proof. [Proof of Theorem 5.1] The proof is a combination of the standard tools listed in Section 2.2 to reduce the problem to Problem 4.1 and then applying Lemma 4.2 for solving this problem.

We start by obtaining an algorithm for an *additive* $\varepsilon\cdot n$ approximation to maximum matching. Define the following parameters for solving Problem 4.1 via the algorithm $\mathbb{A}_k(n_k,m_k,q_k,\gamma_k,\delta_k,\alpha_k)$:

$$\begin{aligned} &n_k := n && m_k := \binom{n}{2} && q_k := g(\gamma,\varepsilon/4)\\ &\gamma_k := (1/15)^k && \delta_k := f(\gamma,\varepsilon/4) && \alpha_k := \varepsilon^2. \end{aligned} \tag{5.4}$$

Suppose we have computed a $(\varepsilon/4)\cdot n$ additive approximate matching at some time. Then, for the next $\alpha_k\cdot n=\varepsilon^2\cdot n$ this remains at least a $(\varepsilon/2)\cdot n$ approximation (even if all updates delete edges of this matching). At this point, we should recompute another $(\varepsilon/4)\cdot n$ additive approximate matching. By Proposition 2.1, at this point, we need to answer $q_k := g(\gamma,\varepsilon/4)$ queries $U\subseteq V$ with $\mu(G[U])\geqslant \delta_k n = f(\gamma,\varepsilon/4)\cdot n$ and returning a matching of size $\mu(G[U])\geqslant \gamma_k\cdot\delta_k\cdot n$ satisfies the requirement of answering the queries in Proposition 2.1. However, we do need to ensure that $\mu(G[U])\geqslant\delta_k\cdot n$ which can be done by running Proposition 2.2 first. At this point, the problem we need to solve to implement Proposition 2.1 is exactly Problem 4.1 with the parameters of Equation (5.4).

Running $\mathbb{A}_k$ for solving Problem 4.1 by Lemma 4.2 is going to have an amortized update time of

$$\begin{aligned} &O\Big((2q_k)^{k-1}\cdot\Big(\frac{m_k}{n_k}\Big)^{1/k+1}\cdot \mathsf{ORS}(n_k,\gamma_k\cdot\delta_k\cdot n/2)^{1-1/(k+1)}\cdot n^{6\gamma_k}\cdot(\log(n)/\delta_k)^c\Big)\\ &= O\Big(n^{1/k+1}\cdot\mathsf{ORS}(n\,,\,\frac{1}{20^k}\cdot f(\gamma,\varepsilon/4)\cdot n)^{1-1/(k+1)}\cdot n^{10/(20)^k}\Big), \end{aligned}$$

with high probability; here, we used that $(2g(\gamma,\varepsilon/4))^{k-1}\cdot n^{6/(20)^k}\cdot(\log(n)/\delta_k)^c = O(n^{10/(20)^k})$ given the values $f(\gamma,\varepsilon), g(\gamma,\varepsilon), \delta_k = O_{k,\varepsilon}(1)$. Also, the runtime of $O(n\cdot\mathrm{poly}(\log(n)/\delta_k))$ for running Proposition 2.2 amortized over the $\varepsilon^2\cdot n$ updates is asymptotically upper bounded by the above and can be neglected. All in all, this implies an algorithm with amortized update time of

$$O\Big(n^{1/k+1}\cdot\mathsf{ORS}(n\,,\,\frac{1}{20^k}\cdot f(\gamma,\varepsilon/4)\cdot n)^{1-1/(k+1)}\cdot n^{10/(20)^k}\Big),$$

for maintaining an additive εn approximate matching in a dynamic graph, with high probability.

Finally, we apply Proposition 2.3 to turn this into a multiplicative $(1-\varepsilon)$-approximation guarantee. This effectively requires re-parameterizing ε with $\Theta(\varepsilon^2)$ in the above bounds (and bounding $n^{10/(20)^k}\cdot\mathrm{poly}(\log(n)/\varepsilon) = O(n^{15/(20)^k})$ by the range of parameters). This implies an algorithm with amortized update time promised in the theorem statement for maintaining a (multiplicative) $(1-\varepsilon)$-approximation to maximum matching in a fully dynamic graph, with high probability.

We shall remark that in the arguments above—in particular, to satisfy the guarantee promised in Problem 4.1—we need to assume that the total number of updates is $\mathrm{poly}(n)$ to be able to apply union bound in conjunction with our high probability guarantees. This however can be easily fixed using a standard trick[4] as we explain next.

After every, say, n^{10} updates to the underlying graph G, we entirely terminate the current run of the algorithm and erase all the data structures. Then, we start a new run of the algorithm on an initially empty graph H and insert the current edges in G to this new graph H using at most $O(n^2)$ insertions. After this step, the graph H

[4]In fact, it appears that many of existing dynamic matching algorithms make this assumption implicitly, e.g., in [BKS23, Liu24, BG24], although some are also more explicit about this, e.g. [BK22].

becomes the same as G and we continue with processing the upcoming updates the current data structures we have. This does not change the asymptotic runtime of the algorithm, but now ensures that even if the algorithm needs to process super-polynomial number of updates, after *each* update, with high probability, the output is correct and the amortized runtime of the algorithm is as desired[5]. This concludes the proof. ∎

5.1 Removing the Assumption on the Prior Knowledge of ORS In the description of our algorithms throughout this paper, we assumed that the algorithm is aware of the value of $\mathsf{ORS}(n, c \cdot n)$ (for a proper choice of c in the algorithm) to find the right balancing point for size of batches. However, as we explain next, this knowledge is actually not necessary, which is a desirable feature given our current state of (lack of) understanding of $\mathsf{ORS}(n, c \cdot n)$.

We again focus on the case of polynomially many updates. The algorithm starts with *guessing* that $\mathsf{ORS}(n, c \cdot n)$ is some $\beta = n^{\Omega(1/\log\log n)}$ (the current best lower bound in Equation (1.1)), runs the algorithm of Theorem 5.1 with β as the value of ORS, and continues as long as the runtime does not exceed the bounds dictated by the current guess β. Now suppose during some run of the algorithm, the runtime exceeds the current update time bound. Then there are two possibilities: either the high probability event of Theorem 5.1 has failed or the guess β of $\mathsf{ORS}(n, c \cdot n)$ falls shorts of the true value. The first case happens with a negligible probability which we can ignore so let us focus on the second case.

The guarantees given in Lemma 4.2 hold for each fixed batch of updates (in other words, we amortize the runtime over a single batch and not beyond that). Suppose we have a batch with longer than expected runtime. We consider all the matchings moved from G_{old} to G_{match} throughout this batch and apply Lemma 2.1 to them in an algorithmic fashion—which takes linear time in the size of the graph—to explicitly construct an ORS graph. If we succeed in creating an ORS graph with strictly more matchings than the current estimate β, we terminate the current algorithm, increase the value of β by a factor of 4, and restart the process from the beginning of the last batch. Since for any $k \geq 1$, the target runtime of the algorithm $\mathbb{A}_k$ is proportional to $\mathsf{ORS}(n, c\,n)^{\eta}$ for some $\eta \geq 1/2$, running the algorithm again with β increased by a factor of 4, results in a geometrically increasing sequence of runtimes. This makes the final runtime only a constant factor larger than if the algorithm had been run with the correct value of $\mathsf{ORS}(n, c \cdot n)$ from the very beginning. We also note that the value of β only monotonically increases over the execution of the algorithm because at every occurrence, we recover a certificate of a new lower bound on the value of $\mathsf{ORS}(n, c \cdot n)$. So the revision of parameter β occurs only $O(\log n)$ times over the *entire* execution of the algorithm.

In summary, we are able to recover the bounds of Theorem 5.1 without a prior knowledge of the value of $\mathsf{ORS}(n, c \cdot n)$ as also advertised in Result 1.

Acknowledgement

We would like to thank Soheil Behnezhad and Alma Ghafari for helpful discussions on their results in [BG24], and Aaron Bernstein, Sayan Bhattacharya, and Thatchaphol Saranurak for their shared discussions and insights on this problem.

Part of this work was conducted while the first named author was visiting the Simons Institute for the Theory of Computing as part of the Sublinear Algorithms program.

References

[ABKL23] Sepehr Assadi, Soheil Behnezhad, Sanjeev Khanna, and Huan Li. On regularity lemma and barriers in streaming and dynamic matching. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 131–144. ACM, 2023.

[ABR24] Amir Azarmehr, Soheil Behnezhad, and Mohammad Roghani. Fully dynamic matching: $(2-\sqrt{2})$-approximation in polylog update time. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 3040–3061. SIAM, 2024.

[5] We shall emphasize that this does not mean on such a long sequence, the guarantees are satisfied *throughout* (as we simply cannot do a union bound over so many events). However, even an adaptive adversary cannot make sure that a *fixed* update returns a wrong answer or take longer than guaranteed except with negligible probability.

[ACG+15] Kook Jin Ahn, Graham Cormode, Sudipto Guha, Andrew McGregor, and Anthony Wirth. Correlation clustering in data streams. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 2237–2246. JMLR.org, 2015.

[ACK19] Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Sublinear algorithms for $(\Delta + 1)$ vertex coloring. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 767–786. SIAM, 2019.

[AG11] Kook Jin Ahn and Sudipto Guha. Linear programming in the semi-streaming model with application to the maximum matching problem. In Luca Aceto, Monika Henzinger, and Jirí Sgall, editors, *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part II*, volume 6756 of *Lecture Notes in Computer Science*, pages 526–538. Springer, 2011.

[AKL16] Sepehr Assadi, Sanjeev Khanna, and Yang Li. The stochastic matching problem with (very) few queries. In Vincent Conitzer, Dirk Bergemann, and Yiling Chen, editors, *Proceedings of the 2016 ACM Conference on Economics and Computation, EC '16, Maastricht, The Netherlands, July 24-28, 2016*, pages 43–60. ACM, 2016.

[AKLY16] Sepehr Assadi, Sanjeev Khanna, Yang Li, and Grigory Yaroslavtsev. Maximum matchings in dynamic graph streams and the simultaneous communication model. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1345–1364. SIAM, 2016.

[ALT21] Sepehr Assadi, S. Cliff Liu, and Robert E. Tarjan. An auction algorithm for bipartite matching in streaming and massively parallel computation models. In Hung Viet Le and Valerie King, editors, *4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021*, pages 165–171. SIAM, 2021.

[AMS12] Noga Alon, Ankur Moitra, and Benny Sudakov. Nearly complete graphs decomposable into large induced matchings and their applications. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 1079–1090, 2012.

[AOSS19] Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear in n update time. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1919–1936. SIAM, 2019.

[AS23] Sepehr Assadi and Janani Sundaresan. Hidden permutations to the rescue: Multi-pass streaming lower bounds for approximate matchings. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 909–932. IEEE, 2023.

[Beh21] Soheil Behnezhad. Time-optimal sublinear algorithms for matching and vertex cover. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 873–884. IEEE, 2021.

[Beh23] Soheil Behnezhad. Dynamic algorithms for maximum matching size. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 129–162. SIAM, 2023.

[BG24] Soheil Behnezhad and Alma Ghafari. Fully dynamic matching and Ordered Ruzsa-Szemerédi graphs. *arXiv preprint arXiv:2404.06069. To appear in FOCS 2024*, 2024.

[BGS18] Surender Baswana, Manoj Gupta, and Sandeep Sen. Fully dynamic maximal matching in $O(\log n)$ update time (corrected version). *SIAM J. Comput.*, 47(3):617–650, 2018.

[BK22] Soheil Behnezhad and Sanjeev Khanna. New trade-offs for fully dynamic matching via hierarchical EDCS. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 3529–3566. SIAM, 2022.

[BKS23] Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. Dynamic $(1 + \varepsilon)$-approximate matching size in truly sublinear update time. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 1563–1588. IEEE, 2023.

[BKSW23] Sayan Bhattacharya, Peter Kiss, Thatchaphol Saranurak, and David Wajc. Dynamic matching with better-than-2 approximation in polylogarithmic update time. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 100–128. SIAM, 2023.

[BS15] Aaron Bernstein and Cliff Stein. Fully dynamic matching in bipartite graphs. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, volume 9134 of *Lecture Notes in Computer Science*, pages 167–179. Springer, 2015.

[BS16] Aaron Bernstein and Cliff Stein. Faster fully dynamic matchings with small approximation ratios. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms,*

SODA 2016, Arlington, VA, USA, January 10-12, 2016, pages 692–711. SIAM, 2016.

[CCE+16] Rajesh Chitnis, Graham Cormode, Hossein Esfandiari, MohammadTaghi Hajiaghayi, Andrew McGregor, Morteza Monemizadeh, and Sofya Vorotnikova. Kernelization via sampling with applications to finding matchings and related problems in dynamic graph streams. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1326–1344. SIAM, 2016.

[FHS17] Jacob Fox, Hao Huang, and Benny Sudakov. On graphs decomposable into induced matchings of linear sizes. *Bulletin of the London Mathematical Society*, 49(1):45–57, 2017.

[FLN+02] Eldar Fischer, Eric Lehman, Ilan Newman, Sofya Raskhodnikova, Ronitt Rubinfeld, and Alex Samorodnitsky. Monotonicity testing over general poset domains. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 474–483, 2002.

[Fox11] Jacob Fox. A new proof of the graph removal lemma. *Annals of Mathematics*, pages 561–579, 2011.

[GKK12] Ashish Goel, Michael Kapralov, and Sanjeev Khanna. On the communication and streaming complexity of maximum bipartite matching. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 468–485. SIAM, 2012.

[GP13] Manoj Gupta and Richard Peng. Fully dynamic $(1+\varepsilon)$-approximate matchings. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 548–557. IEEE Computer Society, 2013.

[HKNS15] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 21–30. ACM, 2015.

[Kis22] Peter Kiss. Deterministic dynamic matching in worst-case update time. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPIcs*, pages 94:1–94:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[Kon18] Christian Konrad. A simple augmentation method for matchings with applications to streaming algorithms. In Igor Potapov, Paul G. Spirakis, and James Worrell, editors, *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018, August 27-31, 2018, Liverpool, UK*, volume 117 of *LIPIcs*, pages 74:1–74:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[Liu24] Yang P. Liu. On approximate fully-dynamic matching and online matrix-vector multiplication. *CoRR*, abs/2403.02582. To appear in FOCS 2024, 2024.

[LMSV11] Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, and Sergei Vassilvitskii. Filtering: a method for solving graph problems in mapreduce. In Rajmohan Rajaraman and Friedhelm Meyer auf der Heide, editors, *SPAA 2011: Proceedings of the 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures, San Jose, CA, USA, June 4-6, 2011 (Co-located with FCRC 2011)*, pages 85–94. ACM, 2011.

[McG05] Andrew McGregor. Finding graph matchings in data streams. In Chandra Chekuri, Klaus Jansen, José D. P. Rolim, and Luca Trevisan, editors, *Approximation, Randomization and Combinatorial Optimization, Algorithms and Techniques, 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2005 and 9th InternationalWorkshop on Randomization and Computation, RANDOM 2005, Berkeley, CA, USA, August 22-24, 2005, Proceedings*, volume 3624 of *Lecture Notes in Computer Science*, pages 170–181. Springer, 2005.

[OR10] Krzysztof Onak and Ronitt Rubinfeld. Maintaining a large matching and a small vertex cover. In Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 457–464. ACM, 2010.

[RS78] Imre Z Ruzsa and Endre Szemerédi. Triple systems with no six points carrying three triangles. *Combinatorics (Keszthely, 1976), Coll. Math. Soc. J. Bolyai*, 18:939–945, 1978.

[Sol16] Shay Solomon. Fully dynamic maximal matching in constant update time. In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 325–334. IEEE Computer Society, 2016.

[Tir18] Sumedh Tirodkar. Deterministic algorithms for maximum matching on general graphs in the semi-streaming model. In Sumit Ganguly and Paritosh K. Pandya, editors, *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2018, December 11-13, 2018, Ahmedabad, India*, volume 122 of *LIPIcs*, pages 39:1–39:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.