

Fully Dynamic Matching and Ordered Ruzsa-Szemerédi Graphs

Soheil Behnezhad
Northeastern University

Alma Ghafari
Northeastern University

Abstract

We study the fully dynamic maximum matching problem. In this problem, the goal is to efficiently maintain an approximate maximum matching of a graph that is subject to edge insertions and deletions. Our focus is particularly on algorithms that maintain the edges of a $(1 - \varepsilon)$ -approximate maximum matching for an arbitrarily small constant $\varepsilon > 0$. Until recently, the fastest known algorithm for this problem required $\Theta(n)$ time per update where n is the number of vertices. This bound was slightly improved to $n/(\log^* n)^{\Omega(1)}$ by Assadi, Behnezhad, Khanna, and Li [STOC'23] and very recently to $n/2^{\Omega(\sqrt{\log n})}$ by Liu [ArXiv'24]. Whether this can be improved to $n^{1-\Omega(1)}$ remains a major open problem.

In this paper, we present a new algorithm that maintains a $(1 - \varepsilon)$ -approximate maximum matching. The update-time of our algorithm is parametrized based on the density of a certain class of graphs that we call Ordered Ruzsa-Szemerédi (ORS) graphs, a generalization of the well-known Ruzsa-Szemerédi graphs. While determining the density of ORS (or RS) remains a hard problem in combinatorics, we prove that if the existing constructions of ORS graphs are optimal, then our algorithm runs in $n^{1/2+O(\varepsilon)}$ time for any fixed $\varepsilon > 0$ which would be significantly faster than existing near-linear in n time algorithms.

Contents

1	Introduction	1
1.1	Our Contributions	1
1.2	Perspective: ORS vs RS	3
2	Our Techniques	3
3	Preliminaries	5
4	Dynamic Approximate Matching	6
4.1	Random Sampling Algorithm	7
4.2	The Overall Algorithm via Certificates and Caching Them	9
4.3	Running Time Analysis	12
5	Bounding Density of (Ordered) Ruzsa-Szemerédi Graphs	16
5.1	Linear Matchings	16
5.2	When Matchings are (Very) Large	21

1 Introduction

We study *dynamic* algorithms for the *maximum matching* problem, a cornerstone of combinatorial optimization. Given a graph $G = (V, E)$ a matching $M \subseteq E$ is a collection of vertex-disjoint edges. A maximum matching is a matching of largest possible size in G . We study the maximum matching problem in fully dynamic graphs. In this problem, the input graph G changes over time via a sequence of edge insertions and deletions. The goal is to maintain an (approximate) maximum matching of G at all times, without spending too much time after each update.

Background on Dynamic Matching: The dynamic matching problem has received a lot of attention over the last two decades [33, 4, 5, 32, 27, 18, 11, 12, 17, 16, 35, 21, 1, 13, 9, 10, 37, 14, 15, 34, 29, 26, 8, 20, 7, 19]. There is a relatively simple algorithm that maintains a $(1 - \varepsilon)$ -approximate maximum matching, for any fixed $\varepsilon > 0$, in just $O(n)$ time per update (see e.g. [27]). The update-time can be significantly improved if we worsen the approximation. For instance, a $1/2$ -approximation can be maintained in $\text{poly log } n$ time per update [36, 4, 9], or an (almost) $2/3$ -approximation can be maintained in $O(\sqrt{n})$ time [12]. Nonetheless, when it comes to algorithms with approximation ratio better than $2/3$, the update-time stays close to n . A slightly sublinear algorithm was proposed by Assadi, Behnezhad, Khanna, and Li [3] which runs in $n/(\log^* n)^{\Omega(1)}$ time per update and maintains a $(1 - o(1))$ -approximation. In a very recent paper, Liu [30] improved this to $n/2^{\Omega(\sqrt{\log n})}$ via a nice connection to algorithms for the online matrix-vector multiplication (OMv) problem. Despite this progress, the following remains a major open problem:

Open Problem 1. *Is it possible to maintain a $(1 - \varepsilon)$ -approximate maximum matching in a fully dynamic graph, for any fixed $\varepsilon > 0$, in $n^{1-\Omega(1)}$ update-time?*

We note that there is an orthogonal line of work on fully dynamic algorithms that instead of maintaining the edges of the matching, maintain only its size [7, 20, 19]. For this easier version of the problem, Bhattacharya, Kiss, and Saranurak [19] positively resolved the open problem above. However, their algorithm crucially relies on only estimating the size and does not work for the problem of maintaining the edges of the matching. We refer interested readers to [7] where the difference between the two versions of the problem is mentioned.

1.1 Our Contributions

Contribution 1: Dynamic Matching. In this paper, we make progress towards [Open Problem 1](#) by presenting a new algorithm whose update time depends on the density of a certain class of graphs that we call Ordered Ruzsa-Szemerédi (ORS) graphs, a generalization of the well-known Ruzsa-Szemerédi (RS) graphs.

Let us start by defining RS graphs.

Definition 1 (Ruzsa-Szemerédi Graphs). *An n -vertex graph $G = (V, E)$ is an $\text{RS}_n(r, t)$ graph if its edge-set E can be decomposed into t edge-disjoint induced matchings each of size r . We use $\text{RS}_n(r)$ to denote the maximum t for which $\text{RS}_n(r, t)$ graphs exists.*

Instead of each matching being an induced matching in the whole graph, the edges of an ORS graph should be decomposed into an ordered list of matchings such that each matching is induced only with respect to the previous matchings in the ordering. The following formalizes this.

Definition 2 (Ordered Ruzsa-Szemerédi Graphs). *An n -vertex graph $G = (V, E)$ is an $\text{ORS}_n(r, t)$ graph if its edge-set E can be decomposed into an ordered list of t edge-disjoint matchings M_1, \dots, M_t each of size r such that for every $i \in [t]$, matching M_i is an induced matching in $M_1 \cup \dots \cup M_i$. We use $\text{ORS}_n(r)$ to denote the maximum t for which $\text{ORS}_n(r, t)$ graphs exists.*

Note that every $\text{RS}_n(r, t)$ graph is an $\text{ORS}_n(r, t)$ graph but the reverse is not necessarily true.

Our main result can now be stated as follows:

Theorem 1. *Let $\varepsilon > 0$ be fixed. There is a fully dynamic algorithm that maintains the edges of a $(1 - \varepsilon)$ -approximate maximum matching in $O\left(\sqrt{n^{1+\varepsilon} \cdot \text{ORS}_n(\Theta(\varepsilon^2 n))} \cdot \text{poly}(\log n)\right)$ amortized update-time. The algorithm is randomized but works against adaptive adversaries.*

To understand the update-time in [Theorem 1](#), we need to understand the density of ORS graphs for linear size matchings. Let $0 < c < 1/5$ be a constant. Since $\text{ORS}_n(cn) \geq \text{RS}_n(cn)$ as every RS graph is also an ORS graph with the same parameters, it is natural to first look into the more well-studied case of RS graphs. In other words, how dense can RS graphs with linear size matchings be? We note that this question has been of interest to various communities from property testing [\[22\]](#) to streaming algorithms [\[25, 3, 2\]](#) to additive combinatorics [\[24\]](#). Despite this, the value of $\text{RS}_n(cn)$ remains widely unknown.

The best lower bound on $\text{RS}_n(cn)$ —i.e., the densest known construction—is that of [\[22\]](#) from more than two decades ago which shows $\text{RS}_n(cn) \geq n^{\Omega_c(1/\log \log n)} = n^{o(1)}$. This is indeed the densest known construction of ORS graphs we are aware of too. If this turns out to be the right bound, [Theorem 1](#) implies a $(1 - \varepsilon)$ -approximation in $\tilde{O}(n^{1/2+\varepsilon})$ time, an almost quadratic improvement over prior near-linear in n algorithms of [\[3, 30\]](#). In fact, we note that so long as $\text{ORS}_n(cn)$ is moderately smaller than n (say $\text{ORS}_n(\Theta(\varepsilon^2 n)) \ll n^{1-2\varepsilon}$) [Theorem 1](#) implies a truly sublinear in n update-time algorithm, positively resolving [Open Problem 1](#).

Finally, we note that there is a long body of work on proving conditional lower bounds for dynamic problems. For instance, the OMv conjecture can be used to prove that maintaining an exact maximum matching requires near-linear in n update-time [\[28\]](#). Adapting these lower bounds to the $(1 - \varepsilon)$ -approximate maximum matching problem has remained open since then. Our [Theorem 1](#) implies that proving such lower bounds either requires a strong lower bound of near-linear on $\text{ORS}_n(\varepsilon n)$, or requires a conjecture that implies this.

Contribution 2: Better Upper Bounds for ORS and RS: Unfortunately there is a huge gap between existing lower and upper bounds for $\text{RS}_n(cn)$ (and as a result also for $\text{ORS}_n(cn)$). The best known upper bound on $\text{RS}_n(cn)$ for linear size matchings follows from the improved triangle-removal lemma of Fox [\[23\]](#) which implies $\text{RS}_n(cn) \leq n/\log^{(\ell)} n$ for $\ell = O(\log(1/c))$ where $\log^{(x)}$ is the iterated log function. We note that this result is implicit in [\[23\]](#) and was mentioned in the paper of [\[24\]](#). To our knowledge, this upper bound does not carry over to ORS graphs (we briefly discuss this at the beginning of [Section 5](#)). Our second result is a similar upper bound for ORS albeit with a worse dependence on constant c .

Theorem 2. *For any $c > 0$, it holds that $\text{ORS}_n(cn) = O(n/\log^{(\ell)} n)$ for some $\ell = \text{poly}(1/c)$.*

Since every RS graph is also an ORS graph with the same parameters, [Theorem 2](#) immediately implies the same upper bound for $\text{RS}_n(cn)$. Note that this implication is not a new result, but the

proof is very different from the abovementioned upper bound.

1.2 Perspective: ORS vs RS

Summarizing the above-mentioned bounds, we have

$$n^{\Omega_c(1/\log \log n)} \stackrel{[22]}{\leq} \text{RS}_n(cn) \leq \text{ORS}_n(cn) \stackrel{\text{Theorem 2}}{\leq} O(n/\log^{\text{poly}(1/c)} n).$$

While the value of $\text{RS}_n(cn)$ remains widely unknown, one might argue that $\text{RS}_n(cn) = n^{o(1)}$ is a plausible outcome, given that the construction of [22] has resisted any improvements for over two decades despite significant interest. But should we believe that $\text{ORS}_n(cn)$ is also small in this case? Unfortunately the authors could not prove any formal relation between the densities of RS and ORS graphs beyond the upper bound of Theorem 2. In particular, we believe the following is an extremely interesting question for future work:

Open Problem 2. *Does it hold that $\text{ORS}_n(cn) = \text{poly}(\text{RS}_n(O(cn)))$?*

In the event that the answer to Open Problem 2 is positive and $\text{RS}_n(cn) = n^{o(1)}$ for any fixed $c > 0$, we also get that $\text{ORS}_n(cn) = n^{o(1)}$. Therefore Theorem 1 would imply an $n^{1/2+O(\varepsilon)}$ time algorithm in this case.

In the event that the answer to Open Problem 2 is negative, one might wonder whether we can improve Theorem 1 by parametrizing it based on RS instead of ORS. Put differently, suppose that $\text{ORS}_n(cn) = n^{1-o(1)}$ and $\text{RS}_n(cn) = n^{o(1)}$. Can we somehow utilize the sparsity of RS graphs (instead of ORS graphs) in this case to improve existing dynamic matching algorithms? We start Section 2 by providing an input construction which informally shows ORS is the right parameter for Theorem 1 even if RS is much sparser.

2 Our Techniques

Before describing the intuition behind our algorithm of Theorem 1, let us start with a sequence of updates that, in a sense, explains why existence of dense ORS graphs would make it challenging to maintain a $(1 - \varepsilon)$ -approximate maximum matching in a fully dynamic setting. We then proceed to show that this input construction is essentially the only instance that we do not know how to solve fast.

An input construction based on ORS graphs: Consider a fully dynamic input graph $G = (V, E)$ that is composed of two types of vertices: the *ORS vertices* $V_{\text{ORS}} \subseteq V$ which is a subset of n vertices, and the *singleton vertices* V_S which is a subset of $(1 - 2\varepsilon)n$ vertices. We start by inserting an ORS graph in the induced subgraph $G[V_{\text{ORS}}]$. Namely, take an $\text{ORS}_n(\varepsilon n, t)$ graph on n vertices. We make the induced subgraph $G[V_{\text{ORS}}]$ isomorphic to this ORS graph by inserting its edges one by one to $G[V_{\text{ORS}}]$. Let M_1, \dots, M_t be the ordered induced matchings of $G[V_{\text{ORS}}]$ as defined in Definition 2. Then the sequence of updates is as follows:

- For $i = t$ to 1:
 - Delete all existing edges of V_S .

- If $i \neq t$, delete the edges of M_{i+1} .
- Let M_i be the *current* induced matching in $G[V_{ORS}]$.
- Insert a perfect matching between the $(1 - 2\varepsilon)n$ vertices of V_{ORS} left unmatched by M_i and the $(1 - 2\varepsilon)n$ vertices in V_S .

Take the graph after the iteration i of the for loop. Note that there is a perfect matching in G : match all singleton vertices to the V_{ORS} vertices not matched by M_i , and match the rest of the vertices in V_{ORS} through M_i . Importantly, since all matchings M_{i+1}, \dots, M_t have already been deleted from the graph, M_i must be an induced matching of the remaining graph G (the other matchings M_1, \dots, M_{i-1} cannot have any edge with both endpoints matched by M_i due to [Definition 2](#)). Because of this, it can be confirmed that any $(1 - \varepsilon/2)$ -approximate maximum matching of G must include at least half of the edges of M_i . The naive algorithm for finding an edge of M_i for some vertex v would scan the neighbors of v , which could take $\Omega(t)$ time per vertex (as this is the degrees in the ORS graph) and thus nt time in total after every iteration of the loop consisting of n updates. Hence, the amortized update-time of this algorithm must be at least $\Omega(nt/n) = \Omega(\text{ORS}_n(\varepsilon n))$.

The input construction above implies that to maintain a $(1 - \varepsilon)$ -approximation of maximum matching, either we have to find a way to identify induced matchings of an ORS graph fast (without scanning the neighbors of each vertex) or have to parameterize our algorithm's update-time by t , the density of ORS graphs.

Overview of our algorithm for [Theorem 1](#): Let us for this informal overview of our algorithm assume that the maximum matching size is at least $\Omega(n)$. Having this assumption allows us to find the matching once, do nothing for the next εn updates, and then repeat without hurting the size of the approximate matching that we find by more than a $1 + O(\varepsilon)$ factor.

As it is standard by now, to find a $(1 - \varepsilon)$ -approximate matching it suffices to design an algorithm that given a subset $U \subseteq V$, finds a constant approximate maximum matching in $G[U]$. If this algorithm runs in T time, we can find a $(1 - \varepsilon)$ -approximate maximum matching of the whole graph also in $O_\varepsilon(T)$ time. Amortized over εn updates, this runs in $O(T/n)$ total time for constant $\varepsilon > 0$. However, just like the challenging example discussed above, in case the maximum matching in the induced subgraph $G[U]$ is an induced matching, we do not know how to find a constant fraction of its edges without spending $\Omega(n^2)$ time. However, if we manage to bound the total number of such hard subsets U for which we spend a lot of time, then we can bound the update-time of our algorithm. Intuitively, we would like to guarantee that if our algorithm takes $\Omega(n^2)$ time to solve an instance $G[U]$, then the maximum matching in $G[U]$ must be an induced matching of the graph G , and charge these heavy computations to ORS which provides an upper bound on the number of edge-disjoint such induced matchings. However, there are two main problems: (1) it may be that the maximum matching in $G[U]$ is not an induced matching of the graph, yet it is sparse enough that it is hard to find; (2) even if $G[U]$ forms an induced matching, we have to ensure that its edges do not belong to previous induced matchings that we have charged, as ORS only bounds the number of *edge-disjoint* ordered induced matchings.

For the first problem discussed above, we present an algorithm that runs in (essentially) $O(n^2/d)$ time to find the maximum matching in $G[U]$. Here d is a parameter that depends on the structure of $G[U]$ that measures how easy it is to find an approximate maximum matching of $G[U]$. Intuitively, if the average degree within $G[U]$ is d , we can random sample pairs of vertices to add to the matching. If each vertex is adjacent to d others, we only need $O(n/d)$ samples to match it and the algorithm

runs in $O(n^2/d)$ time. Of course, this can take up to $\Omega(n^2)$ time if $G[U]$ is sparse – e.g. when it is an induced matching. Then instead of charging a matching in $G[U]$ that is an induced matching, we charge this matching of average degree at most d inside. Let M_1, \dots, M_t be the matchings that we charge and let d_i be the average degree of the i -th matching and suppose that these matchings are edge-disjoint. We show in our update-time analysis that $\sum_{i=1}^t 1/d_i$ can be at most $\text{ORS}(\varepsilon^2 n)$, and therefore the total time spent by our algorithm can be upper bounded by $n^2 \text{ORS}(\varepsilon^2 n)$.

For the second problem, or in other words, to ensure that the matchings that we charge are edge-disjoint, we maintain a set S and add all edges of any matching that we charge to this set. Thereafter, before solving $G[U]$, we first go over the edges stored in this set and see whether they can be used to find a large matching in $G[U]$. If they do, we do not run the random sampling algorithm discussed above. If not, the matching that we find must be edge-disjoint. We have to be careful that we do not make S too dense though as we spend linear time in the size of S . Our final algorithm resets S after a certain number of updates.

Overview of our upper bound in Theorem 2: To obtain our upper bound of Theorem 2, we partition the matchings into two subsets, \mathcal{M} and \mathcal{M}' , based on their order in the sequence. A key insight is the following: take a vertex v and suppose that it is matched by some matching M in \mathcal{M}' . If we remove all neighbors of vertex v in \mathcal{M} , then it can be proved that no vertex of matching M is removed because otherwise there must be an edge from \mathcal{M} that matches two vertices of M , violating the inducedness property. Intuitively, this shows that if vertices have large degrees in \mathcal{M} , we can remove a relatively large number of vertices without hurting the matchings that include this vertex. To derive the upper bound, we carefully select a set of pivots based on the degrees in \mathcal{M} , and remove the neighbors of these pivots. We show that this reduces the number of vertices significantly enough, and keeps the size of a small (but sufficiently many) of the matchings unchanged. If the initial number of matchings is so large, we show that we can iteratively applying this procedure. Because the size of matchings do not change but the number of vertices drops, we get that the process should eventually stop. This implies the upper bound on the number of matchings in the starting graph.

3 Preliminaries

A *fully dynamic* graph $G = (V, E)$ is a graph defined on a fixed vertex set V that is subject to edge insertions and deletions. We assume that each edge update is issued by an *adaptive adversary* that can see our algorithm's previous outputs and can accordingly decide on the next update. We say an algorithm has amortized update-time U if the total time spent after T updates is $U \cdot T$ for some sufficiently large $T = \text{poly}(n)$.

Tools from prior work: Here we list some of the tools we use from prior work in our result.

The following proposition, implied by the streaming algorithm of McGregor [31] (see also [19] for its dynamic adaptation), shows that to find a $(1 - \varepsilon)$ -approximate matching, it suffices to solve a certain induced matching problem a constant number of times.

Proposition 3 ([31]). *Let $G = (V, E)$ be any (possibly non-bipartite) n -vertex graph. Suppose that for some parameters $\varepsilon \in (0, 1)$ we have an algorithm \mathcal{A} that provided any vertex subset $U \subseteq V$ with $\mu(G[U]) \geq \varepsilon n$, finds a matching of size at least $\varepsilon \mu(G[U])$ in $G[U]$. Then there is an algorithm that finds a matching of size at least $\mu(G) - O(\varepsilon n)$ in G by making $\tilde{O}_\varepsilon(1)$ adaptive calls to \mathcal{A} where*

preparing the subset U for each call can be done in $\tilde{O}_\varepsilon(n)$ time.

We also use the following sublinear-time algorithm of Behnezhad [6] for estimating the size of maximum matching. We note that even though we use this algorithm, we maintain the edges of the maximum matching explicitly.

Proposition 4 ([6]). *Let $G = (V, E)$ be any (possibly non-bipartite) n -vertex graph. For any $\varepsilon > 0$, there is an algorithm that makes $\tilde{O}(n \text{poly}(1/\varepsilon))$ adjacency matrix queries to G and provides an estimate $\tilde{\mu}$ of the size of maximum matching $\mu(G)$ in G such that with probability $1 - 1/\text{poly}(n)$, it holds that*

$$0.5\mu(G) - \varepsilon n \leq \tilde{\mu} \leq \mu(G).$$

Finally, we use the following vertex sparsification idea which was adapted to the dynamic setting in the work of Kiss [29].

Definition 5. *Let V be a vertex set and denote $k \leq |V|/2$ be a parameter. A set $\mathcal{P} = \{P_1, \dots, P_\ell\}$ of ℓ partitionings of V is a (ℓ, k, ε) -vertex sparsifier if the following hold:*

1. *Each partitioning P_i partitions V into $O(k/\varepsilon^2)$ subsets.*
2. *For each subset $S \subseteq V$ of size $2k$, there is at least one partitioning P_i such that there are at least $(1 - \varepsilon)|S|$ subsets in P_i that have exactly one vertex of S .*

Proposition 6 ([29]). *Let \mathcal{P} include $\ell = \lceil 512 \log n / \varepsilon^2 \rceil$ random partitionings P_1, \dots, P_ℓ of V , where each P_i partitions V into at least $8k/\varepsilon$ subsets randomly. Then, with probability $1 - 1/\text{poly}(n)$, \mathcal{P} is a (ℓ, k, ε) -vertex sparsifier of V .*

The notion of graph contraction here refers to the process of simplifying the graph based on the partitionings provided by the vertex sparsifiers. Each set of partitions effectively represents a contracted version of the graph, where the vertices within the same subset can be considered as a single entity for the purpose of matching. This contraction not only reduces the overall size of the graph but also retains the essential structural properties that influence the size and composition of maximum matchings.

Definition 7. *Denote M be a given matching with $2r$ vertices in graph G . Let us assume the d_v for a vertex v in the matching is the number of neighbors of v in $V(M)$. we define average degree of M as $\sum_{v \in V(M)} d_v / 2r$.*

4 Dynamic Approximate Matching

We present our dynamic matching algorithm in this section and prove [Theorem 1](#).

Let us start with an overview of how our proposed algorithm is built. We randomly sample edges from graph G to find a matching with ε -approximation, which is described in [Algorithm 1](#). We propose that if we need many samples to find this matching then the average degree of this matching is small in the remaining edges (what we mean by the remaining edges is that, when we output a matching, the adversary can remove its edges from the graph, while searching for a new matching we should discard the edges found before), resulting in the matching to be induced in the remaining graph. In this case, we charge the cost of the algorithm to ORS, with characteristics similar to the matching we found. Otherwise, we will succeed in finding a matching, relatively fast.

Note that, in this model, we can argue that if the matching size in the graph is μ then the matching size will not change after $\varepsilon^2 n$ updates. We will find a matching once in $\varepsilon^2 n$ edge updates to optimize amortized running time. This motivates us to run [Algorithm 1](#) to find a matching after $\varepsilon^2 n$ updates. To keep track of the large matchings we already found we introduce [Algorithm 2](#). This algorithm simplifies our problem where the matching size does not change. If we maintain this algorithm for different matching sizes, knowing the approximate matching we will know which maintained algorithm to look at to find the ε -approximate matching.

To boost our approximation to $(1 - \varepsilon)$ we use [Proposition 3](#) to output the subsets of U that we need to search for the matchings of size $\varepsilon\mu$. We use the subsets in [Algorithm 1](#) which is a ε -approximation algorithm.

Note that we provide a multiplicative approximation maximum matching algorithm for the dynamic model, which leads to the fact that we need to handle the cases where the matching size is not of order n . This adds some layers to the intuition we stated above.

1. If the matching size is n' we output a matching after $\varepsilon^2 n'$ updates. This means we need to handle matching with similar sizes together, we will have $\log n$ partition to keep track of the matchings outputted by [Algorithm 1](#) of almost the same size simultaneously.
2. To handle the graph with small matchings of $O(n')$ we need to contract the graph to one with n' vertices. We build a contracted graph by creating n' buckets of $O(n/n')$ vertices, the vertices are distributed randomly in buckets. Since we want to preserve the edges of the maximal matching we need to randomly maintain $O(\log n)$ such contractions for $\log n$ values of n' .
3. The fact that the edges of the matching found in the earlier steps of [Algorithm 2](#), could be removed by the adversary leads us to a definition of the degree of matching that only takes into account the matchings that we will be finding in the next rounds (we do not care about the edges removed from the graph in the dynamic steps). This concept leads us to define the set of matchings named ORS, where every matching M is induced in the set of matchings we observe after M .

4.1 Random Sampling Algorithm

Here we introduce a sampling algorithm that given a graph G , with a matching of size μ can output a matching of size $\varepsilon\mu$. We bound the algorithm running time with the average degree of the outputted matching. The way the algorithm works is that for each vertex we sample b vertices and check if the edge is in the graph using the adjacency matrix of the graph. We add the edges to the matching if none of the endpoints have been used previously. Note that by starting with a small b for the budget, we ensure to find the edges from dense parts of the graph which is easier to find. As we go further in the algorithm we make the budget larger to find edges in more sparse parts of the graph. Now, we state the random sampling algorithm and prove its efficiency.

Lemma 8. *Given a graph $G = (V, E)$, vertex subset $U \subseteq V$, and parameters $\varepsilon \in (0, 1)$, such that maximal matching of $G[U]$ is $\mu \geq \varepsilon n$, there is an algorithm that finds a matching $M \subseteq G[U]$ of size at least $\varepsilon\mu$. The algorithm runs in time $O(n^{2+\varepsilon}/d)$ where $d = |E \cap V^2(M)|/O(n)$.*

Proof. Let us consider [Algorithm 1](#), first we prove that this algorithm will provide a matching M_c of size at least $\varepsilon\mu/2$ by sampling vertices with budget $b \in [2^{\varepsilon c \cdot \log n}, 2^{\varepsilon(c+1) \cdot \log n}]$. Let us assume there

Algorithm 1: RandomSamplingAlgorithm($G[U], \varepsilon, \mu$)

Parameters: $\varepsilon \in (0, 1]$.

```

1  $M \leftarrow \emptyset$ .
2  $M_1, \dots, M_{1/(2\varepsilon)} \leftarrow \emptyset$ .
3  $S \leftarrow \emptyset$ 
4  $b \leftarrow 1$ .
5  $c = 0$ .
6 while  $|M| < \varepsilon\mu$  do
7   if  $\log b > \varepsilon c \cdot \log n$  then
8      $M_{c+1} \leftarrow M$ .
9      $M \leftarrow \emptyset$ .
10     $c \leftarrow c + 1$ .
11  for vertex  $v \in U \setminus V(M)$  in random order do
12     $S \leftarrow$  Sample  $b$  vertices in  $U$ .
13    if  $\exists u \in S$  such that  $u \in U$  and  $u \notin V(M)$  and  $u$  is a neighbor of  $v$  then
14       $M \leftarrow (v, u) \cup M$ .
15   $b \leftarrow 2b$ 
16 return  $M$ 

```

are no such intervals. This means that for any integer $c \in [0, 1/(2\varepsilon)]$, running the algorithm for b in the interval $[2^{\varepsilon c \cdot \log n}, 2^{\varepsilon(c+1) \cdot \log n}]$ outputs a matching with less than $\varepsilon\mu$ edges. Note that for each edge e in $M_1 \cup \dots \cup M_{1/(2\varepsilon)}$, at most 2 edges in the maximum matching of $G[U]$ have common vertices with e . Summing over all the $1/(2\varepsilon)$ intervals we have:

$$2 \sum_{c=1}^{1/(2\varepsilon)} |M_i| < 2 \cdot \frac{1}{2\varepsilon} \cdot \varepsilon\mu/2 = \mu/2.$$

The fact that we could not find a matching of size $\mu/2$ is a contradiction with the lemma statement ensuring that the size of maximum matching is at least $\mu/2$. Note that by exhausting the algorithm with budget of n , we will go through all the edges and output a maximal matching. Note that any maximal matching in this graph should be of size at least $\mu/2$. Therefore, we have at least the size of one of the matchings in $\{M_1, \dots, M_{1/(2\varepsilon)}\}$ is $\varepsilon\mu/2$.

Now, we prove the statement of the algorithm. We assume that this algorithm terminates at b such that $2^{\varepsilon(c-1) \cdot \log n} \leq b < 2^{\varepsilon c \cdot \log n}$, for a constant δ , we spend $n[b + b/2 + \dots + 1] < \delta n b \leq \delta n \cdot 2^{\varepsilon c \log n} < \delta n^{1+\varepsilon c}$ time to find the matching M_c such that $|M_c| \geq \varepsilon\mu/2$. Note that if it takes at most $n^{2+\varepsilon}/d$ time to find the matching of size $\varepsilon\mu/2$, then we have $n^{2+\varepsilon}/d < \delta n^{1+\varepsilon c}$ therefore, $d > \frac{n^{1+\varepsilon-\varepsilon c}}{\delta} \geq \frac{n^{1+\varepsilon}}{\delta}$.

For budget b , we sample vertices for all the vertices, we argue that for the vertices that are not in M , and after sampling b vertices we could not find their neighbor then the degree of vertices out of the matching is less than $O(n/b)$. Assuming that the degree of vertex v is $d_v = c \cdot \frac{n}{b}$, with probability $(1 - \frac{d_v}{n})^b = (1 - c \cdot \frac{1}{b})^b \approx (1/e)^c$ we sampled a neighbor. Therefore in this case with a constant probability we already matched v . This implies that if we find the first edge of this

matching at a budget of $2b$, we get that the degree of any vertex out of the matching we found until this point is less than $c \cdot \frac{n}{b}$ for any constant c . Picking c small enough, the total sum of the degree of the matching is at most $\varepsilon\mu c \cdot \frac{n}{b} < \varepsilon\mu n^{(1-\varepsilon(c-1))} = O(nd)$. \square

4.2 The Overall Algorithm via Certificates and Caching Them

In the next step, we aim to use [Algorithm 1](#) efficiently to output an approximate matching in the dynamic model. First, we prove that given the matching size is equal to $O(n')$, after $\varepsilon^2 n'$ updates the matching size does not change substantially. This means we can output a matching after each $\varepsilon n'$ update and get an ε -approximate matching.

Observation 9. *Given a graph $G = (V, E)$ with n vertices, if the maximum matching size is $O(\mu)$ and we provide an ε -approximate matching M' , after $\varepsilon^2 \mu$ updates in the dynamic model, M' or one of its subsets remains an ε -approximate matching for the updated graph.*

Proof. We assume that in a graph G the maximal matching size is $c\mu$ where c is a constant, after $\varepsilon^2 \mu$ updates in the graph the size of the maximal matching is between $(c - \varepsilon^2)\mu, (c + \varepsilon^2)\mu$. This implies that if we have an algorithm that outputs an $\varepsilon c\mu$ matching, after $\varepsilon^2 \mu$ updates, the ε approximate matching is preserved. This is because we lose at most ε^2 fraction of the outputted matching. \square

Note that to use [Algorithm 1](#) we need to know about the existence of a matching μ , the algorithm described in [Proposition 4](#), given a graph with n vertices, outputs if there is matching of size $O(n)$. Using this algorithm alone will not help us with the cases where the matching size is not of order n . This motivates us to find an algorithm that reduces the number of vertices on the graph while maintaining the maximum matching size. [Proposition 6](#) will maintain $O(\log n)$ contraction on the graph for an integer $k \in [1, \log n]$, such that for any matching M with high probability, there is a contraction preserving a $1 - \varepsilon$ approximation of M . After running the reduction on the graph, we find the contraction that preserves the maximum matching. Let us assume that G' is the contracted graph, we run [Algorithm 1](#) on G' to output the ε approximation matching.

Note that throughout the changes in the dynamic model, both the matching size and the contracted graph that preserves the maximum matching, change. An edge insertion in G is a weight addition to an edge in graph G' , and edge deletion is a weight subtraction in G' . In the following algorithms, when we find a matching in G' , we do not care about the weight of edges, an edge in G' exists if its weight is more than 0, otherwise, This edge does not exist in G' .

First, we solve a simplified version of the problem where the matching size does not change throughout the updates, and a single contracted graph G' preserves matching edges in the dynamic model. We prove that there exists an efficient algorithm maintaining a $(1 - \varepsilon)$ -approximation matching in the dynamic model.

We introduce [Algorithm 2](#). We assume that the updates in this algorithm are in the case that the matching size in $O(n')$, and G' preserves the maximum matching of the graph. This algorithm works in phases, each phase represents the number of updates we want to use the matchings we found in the last updates.

First, we prove that this algorithm obtains a ε -approximation matching.

Lemma 10. *Given a contracted graph G' with n' vertices that the maximum matching size in G' is $O(n')$ in t updates, [Algorithm 2](#) maintains an $(1 - \varepsilon)$ -approximate maximum matching for G' in the dynamic model.*

Algorithm 2: Algorithm for Dynamic Model in Contracted Graphs

Parameters: $\varepsilon \in (0, 1]$, t the threshold of the algorithm.

```

1  $G' \leftarrow$  the contracted graph.
2  $S_1 \leftarrow \emptyset$ .
3  $S_2 \leftarrow \emptyset$ .
4 while edge updates  $< t$  do
5   if There are  $\varepsilon^2 n'$  new updates then
6      $S_1 \leftarrow S_1 \cup \text{Insertions}$ .
7      $M \leftarrow \emptyset$ .
8     while Proposition 3 outputs a subset  $U$  that has a matching of size at least  $\mu = \varepsilon n'$ 
9       do
10         $M' \leftarrow$  a maximal matching found by a greedy algorithm on edges of  $S_1 \cup S_2$  in
11         $U$ . For each edge in this set, we check if it exists in the dynamic graph.
12        if  $|M'| > \varepsilon \mu$  then
13           $M \leftarrow M'$ .
14        else
15           $M_U \leftarrow \text{RandomSamplingAlgorithm}(G[U \setminus V(M')], \frac{\varepsilon}{1-2\varepsilon}, (1-2\varepsilon)\mu)$ .
16           $M \leftarrow M \cup M_U$ .
17         $S_2 \leftarrow S_2 \cup M_U$ .
18      Output  $M$ .
```

Proof. Note that we proved in [Observation 9](#) that if the maximal matching size is of order n' , then we do not need to output a matching in $\varepsilon^2 n'$ updates.

By [Algorithm 2](#), set S_1 contains edge insertions, and set S_2 contains all the matchings we found earlier in the algorithm.

Using [Proposition 3](#), we are given constant subsets of vertices that contain matchings of size at least εn , we can run [Algorithm 1](#) on these subsets to output a ε -approximate matching in given subset U , if there are no large enough activated matchings in set $S_2 \cup S_1$. Given that we aim to find edge-disjoint matchings, we need to find a maximal matching in $S_2 \cup S_1$ and remove its endpoints from U .

After finding a matching M' in $S_2 \cup S_1$ of size $\varepsilon \mu$, we output this matching; otherwise, we deactivate all the endpoints in M' . Note that we are certificated by [Proposition 3](#) to have a matching of size μ in U . Since for maximal matching M' in $S_1 \cup S_2$ we have $|M'| < \varepsilon \mu$, then the maximum matching size in $S_1 \cup S_2$ is at most of size $2\varepsilon \mu$. This certifies the existence of a matching of size $(1 - 2\varepsilon)\mu$ in the rest of U . Now, we run [Algorithm 1](#) on graph U with endpoints of M' removed with parameter $\varepsilon/(1 - 2\varepsilon)$. This implies that we get a matching of size $\varepsilon \mu$. Since we deactivate vertices of S_2 we only add edge-disjoint matchings to set S_2 .

Note that any vertex in G' represents a bucket of vertices of G . By maintaining the number of vertices between two buckets in G , we can observe whether there is an edge between two vertices in G' . This is how we can certify if a matching M' in S_2 is in G' .

We proved that [Algorithm 2](#) outputs a ε -approximate matching M_U in each subset of U pre-

sented by [Proposition 3](#). Note that given M_U for each subset U , [Proposition 3](#) finds a $(1 - \varepsilon)$ -approximation matching in G' . This is true by the assumption that the maximum matching size of G' is $O(n')$. \square

Now that we introduced an algorithm for each contraction, we explore how to build and maintain each contraction. To partition updates to the contraction graphs that we maintain we need to run [Proposition 4](#) to approximate the matching size, and then use [Proposition 6](#) to figure out the contracted graph preserving the maximum matching, which we will do after each $\varepsilon^2 n'$ updates, given that the last contracted graph had n' vertices. We introduce [Algorithm 3](#) that given updates calculates the matching size and finds a suitable contraction for the graph at each state. For each update, [Algorithm 3](#) calls [Algorithm 2](#) on all the contractions, however, it only outputs the matching build by the matching preserving contraction.

We will run $O(\log^2 n)$ parallel versions of [Algorithm 2](#) each for a maintained contracted graphs that we maintain. More formally we introduce [Algorithm 3](#):

Algorithm 3: Main

Parameters: $\varepsilon \in (0, 1]$.

- 1 $k \leftarrow \log n$.
- 2 $n_1, n_2, \dots, n_k \leftarrow 2, 4, \dots, 2^k$.
- 3 $t \leftarrow$ threshold of algorithm [Algorithm 2](#).
- 4 $(C) \leftarrow$ set of maintained $O(\log n)$ contracted graphs for each $n_c \in \{n_1, \dots, n_k\}$ using [Proposition 6](#).
- 5 $\mathcal{P} \leftarrow$ set of maintained parallel edges between two contracted sets of vertices after each update.
- 6 Run a new phase [Algorithm 2](#) for each contraction after t updates.
- 7 For each contraction c with n_c vertices, run [Proposition 4](#) on c after $\varepsilon^2 n_c$ updates to see whether c preserves the maximum matching in G .
- 8 Find the contraction c preserving the largest matching of G and use the output of [Algorithm 2](#) for the corresponding contraction, to output an $(1 - \varepsilon)$ -approximate matching.

Now, we prove that this algorithm outputs a $(1 - \varepsilon)$ -approximation matching.

Lemma 11. *Given graph G with n vertices, [Algorithm 3](#) maintains an $(1 - \varepsilon)$ -approximate maximum matching for G in the dynamic model.*

Proof. In this algorithm, we maintain $O(\log^2 n)$ contraction of graph G , where G is the updated graph in the current batch of t updates. If the maximum matching size of G is μ , by [Proposition 6](#), there exists a contraction c with n_c vertices that preserves the maximum matching with $1 - \delta$ factors and $\mu = O(n_c)$. Denote δ a parameter sufficiently smaller than ε . We call c a preserving contraction of G . This means that by [Lemma 10](#), [Algorithm 2](#) outputs a $(1 - \delta)$ -approximation matching for G' , that is $(1 - \varepsilon)$ -approximation matching for G . Note that, by running [Proposition 4](#) on the maintained reduction, we find the preserving contraction of G . This concludes the $(1 - \varepsilon)$ -approximate. \square

4.3 Running Time Analysis

Now, we explore the time per update for [Algorithm 2](#).

Lemma 12. *If we set the threshold of [Algorithm 2](#) equal to $t = n^{1+\varepsilon/2} \sqrt{n \log n \cdot \text{ORS}_n(\varepsilon\mu)}$, then the time spent on single contraction G' in both [Algorithm 2](#) is $O(\sqrt{n^{1+\varepsilon} \log n \cdot \text{ORS}_n(\varepsilon\mu)})$ per each update, where μ is the maximum matching size in G' .*

Proof. Fix a contracted graph $G' = (V', E')$ with n vertices. We explore the algorithm running time for this graph.

First, let us determine the time the algorithm spends finding the size of the certificate matching, μ , after $\varepsilon^2 n$ updates. Note that we run [Proposition 4](#) after each $\varepsilon^2 n$ vertex update, this algorithm outputs μ , which is a constant approximation of the maximum matching size of the graph. Moreover, this algorithm certifies the existence of a matching of size μ in the graph, the running time for which is $\tilde{O}(n \text{ poly}(1/\varepsilon))$.

By [Proposition 3](#), after each $\varepsilon^2 n$ update we spend $\tilde{O}_\varepsilon(n)$ on finding the appropriate subsets of vertices such as U , including matching of size at least εn .

Now, we calculate the time [Algorithm 2](#) takes going through the set S_1 and S_2 , after $\varepsilon^2 n$ updates to find a matching M' . Note that, after $\varepsilon^2 n$ updates we go over the set $S_1 \cup S_2$ to find an active matching. We add an active edge to the matching if none of the endpoints was added before it. The algorithm outputs a maximal matching this way. The total running time per $\varepsilon^2 n$ updates is $|S_1| + |S_2|$.

Finally, we explore the running time of [Algorithm 1](#). Let us assume that at the end of a phase of [Algorithm 2](#) we have $S_2 = \{M_1 \cup \dots \cup M_l\}$. We prove in [Lemma 10](#) that these matchings are edge-disjoint. Note that the total edge-disjoint matchings in G' can be more than l matching, M_1, \dots, M_l are all the matchings we find during the t edge updates. We assume that the total edge-disjoint matchings in G' is l_t . We label these matchings as $M_1, \dots, M_l, M_{l+1}, \dots, M_{l_t}$. Let us assume that for $i \in \{1, \dots, l\}$ we denote d_i the average degree of matching M_i in the edge set of $\{M_i \cup \dots \cup M_{l_t}\}$. Denoting by M' a maximal matching in $S_1 \cup S_2$ found in [Algorithm 2](#), we prove that if [Algorithm 1](#) in graph $G' \setminus V(M')$, takes $O(n^{2+\varepsilon}/d)$ to output matching M_i then we have $O(d) = d_i$. Note that by [Lemma 8](#) we know that the average degree of matching M_i in $G' \setminus (S_1 \cup S_2)$ is $O(d)$. Note that, for matching M_1, \dots, M_{l-1} , either their edges are removed from the graph or we deactivate the vertices that made a large matching. This implies that no edges from M_1, \dots, M_{l-1} remain in the graph. If we assumed that an edge e was still in the graph with both endpoints not deactivated, we could add this edge to M' , this is in contradiction with the fact that M' is a maximal matching. This implies that $d_i = O(d)$. This results in the conclusion that the running time of [Algorithm 1](#) is less than $n^{2+\varepsilon}/d_i$.

This definition of matching average degree motivates us to define ORS graphs. Recall that the ORS graph is a set of matching M_1, \dots, M_l that each matching is induced in the set of matching with higher indices. We then upper bound the running time of our algorithm by the size of ORS graphs according to [Lemma 13](#).

Now, let us compute the total running time per update. Note that since we add a matching of size $\varepsilon\mu$ to S_2 after $\varepsilon^2 n$ updates, $|S_2| < |S_1|$. Let us assume that we add M_1, \dots, M_l to S_2 by running [Algorithm 1](#). The total running time per update is:

$$\frac{\frac{|S_1|}{\varepsilon^2 n} \cdot (|S_1| + |S_2| + \tilde{O}(n)) + n^{2+\varepsilon} \cdot \sum_{i=1}^l 1/d_i}{t}.$$

Using [Lemma 13](#) we have the running time as:

$$\frac{\frac{|S_1|}{\varepsilon^2 n} \cdot (|S_1| + |S_2| + \tilde{O}(n)) + n^{2+\varepsilon} \cdot O(\text{ORS}_n(\varepsilon\mu)) \log n}{t}.$$

Now, if we set the threshold as $t = n^{1+\varepsilon/2} \sqrt{n \log n \cdot \text{ORS}_n(\varepsilon\mu)}$ we have:

$$|S_2|, |S_1| < t = n^{1+\varepsilon/2} \sqrt{n \log n \cdot \text{ORS}_n(\varepsilon\mu)},$$

This is because we could add a matching of size $\varepsilon\mu$ to S_2 after $\varepsilon^2 n$ updates, and the total insertions are less than the total updates. The total running time per update is:

$$O(\sqrt{n^{1+\varepsilon} \log n \cdot \text{ORS}_n(\varepsilon\mu)}). \quad \square$$

Lemma 13. *Given a set of matching M_1, \dots, M_t with distinct edges, of size at least r in graph G , for any index $i \in [1, t]$ let us define d_i as the average degree of $V(M_i)$ in $M_1 \cup \dots \cup M_i$. For any $\alpha \in (0, 1)$, we have $\sum_{i=1}^t 1/d_i = O(\text{ORS}_n(r(1-\alpha)) \cdot \log n)$.*

Proof. Note that the maximum average degree of a matching M is $n-1$, and the minimum is 1. Therefore, if we partition the input matchings into $\log n = k$ groups, the average degree of any matching in each partition is at most within 2 factors of the minimum average degree.

We first prove that if we have t' matchings $M'_1, \dots, M'_{t'}$ a subset of $\{M_1, \dots, M_t\}$, and their average degree as $d'_1, \dots, d'_{t'}$ that for any $i \in [1, t']$, $1/d'_i$ is in a 2 factor of $\min_{i=1}^{t'} 1/d'_i$. For any $\alpha \in (0, 1)$, we have $\sum_{i=1}^{t'} 1/d'_i = O(\text{ORS}_n(r(1-\alpha)))$.

To prove this, we first sample some of the matchings and the edges that are not induced in proper subsets of matchings. We prove there are plenty number of matchings remaining that are induced within their corresponding set of matchings.

Let us fix a small variable $\delta < \varepsilon, \delta, \varepsilon \in (0, 1)$. We set $H = M'_1 \cup \dots \cup M'_{t'}$.

- Step 1. Let us define x_e for any edge $e = (u, v)$ appearing in matching M'_{α_j} as follows

$$\sum_{i \in \{j, j+1, \dots, t'\}; u, v \in V^2(M_{\alpha_i})} \frac{1}{d'_{\alpha_i}}.$$

We define $E_1 = \{e | x_e > 1/\varepsilon\}$ We remove any matching from H with more than $2\varepsilon r$ edges in E_1 . Let us call the remaining set of matchings H_1 .

- Step 2. Let us pick a matching $M'_i \in H$ with probability $\frac{\delta}{2d'_i}$. Let us assume that at this step, we pick the following k matchings $M'_{\alpha_1}, \dots, M'_{\alpha_k}$. we define this subset of the graph as H_2 .
- Step 3. Now, we prune any matching M'_i chosen in H_1 and has less than $(1-\delta)cn$ edges induced in $(M'_1 \cup \dots \cup M'_i) \cap (H_2)$. Let us call the remaining graph H_3 .

To see that the claim is true, first, we need to prove that in Step 1 we remove at most ε fraction of the edges in H . Second, we prove that H_2 is large enough.

Let us assume that after Step 1 we remove more than $\varepsilon rt'$ of the edges in H from the total rt' edges. Therefore, we have

$$\sum_{e \in E_1} x_e > rt'.$$

To compute $\sum_{e \in H} x_e$, we notice how much a matching M'_{α_i} contributes to the total sum. Note that since the average degree of M'_{α_i} in $\{M'_{\alpha_1}, \dots, M'_{\alpha_i}\}$ is at most d'_{α_i} therefore, the total contribution of matching M'_{α_i} is $r \cdot d'_{\alpha_i} \cdot \frac{1}{d'_{\alpha_i}}$. In conclusion, the total sum of x_e for any edge $e \in H$ is at most

$$\sum_{i \in \{1, \dots, t'\}} d'_{\alpha_i} \cdot r / d'_{\alpha_i} = rt',$$

which is a contradiction suggesting that we remove at most $\varepsilon rt'$ edges.

Now that we only removed at most ε fraction of the edges, note that H_1 contains at least half of the matchings in H . This is because if more than $|H|/2$ matchings had more than $2\varepsilon r$ edges removed, then the total number of edges in E_1 would be more than $\varepsilon rt'$ which is a contradiction.

We aim to prove that H_3 contains $\delta|H_1|/2$ matchings. Let us fix a matching $M \in H_1$. Note that for any remaining edge e in M , we have $x_e < 1/\varepsilon$. Therefore, the probability that e belongs to another matching is less than $\delta/2\varepsilon$ since we picked each matching M'_i with probability $\delta/2d'_i$. This implies that the expected number of edges in $E(M) \cap E(H_3)$ is more than $(1 - 2\varepsilon)(1 - \delta/2\varepsilon)r$. Now, by applying Markov's inequality to the random variable $r(1 - 2\varepsilon) - |E(M) \cap E(H_3)|$, here we assume that the maximum number of edges is $r(1 - 2\varepsilon)$, and we get the following.

$$\Pr[r(1 - 2\varepsilon) - |E(M) \cap E(H_3)| \geq (1 - 2\varepsilon)(\delta/\varepsilon)r | M \in H_2] \leq \frac{\mathbf{E}[r(1 - 2\varepsilon) - |E(M) \cap E(H_3)| | M \in H_2]}{(1 - 2\varepsilon)(\delta/\varepsilon)r}.$$

Since $\mathbf{E}[r(1 - 2\varepsilon) - |E(M) \cap E(H_3)| | M \in H_2] \leq ((1 - 2\varepsilon)\delta/2\varepsilon)r$ we have

$$\Pr[r(1 - 2\varepsilon) - |E(M) \cap E(H_3)| \geq (1 - 2\varepsilon)(\delta/\varepsilon)r | M \in H_2] \leq \frac{(\delta/2\varepsilon)r}{(\delta/\varepsilon)r} = 1/2.$$

Therefore, we have

$$\Pr[r(1 - 2\varepsilon) - |E(M) \cap E(H_3)| \leq (1 - 2\varepsilon)(\delta/\varepsilon)r | M \in H_2] > 1/2,$$

which means

$$\Pr[|E(M) \cap E(H_3)| \geq (1 - 2\varepsilon)(1 - \delta/\varepsilon)r | M \in H_2] > 1/2.$$

Multiplying with the probability that $M \in H_2$, we get that the probability of matching to have more than $(1 - 2\varepsilon)(1 - \delta/\varepsilon)r$ edges and be selected is more than $\delta/(4\varepsilon d'_i)$.

Now, summing over all matchings in H_1 we get

$$\mathbf{E}[|H_3|] \geq (\delta/(4\varepsilon)) \sum_{i, M'_i \in H_1} 1/d'_i.$$

Since H_3 is an ORS graph we get

$$(4\varepsilon)\text{ORS}_n((1-2\varepsilon)(1-\delta/\varepsilon)r)/\delta \geq \sum_{i, M'_i \in H_1} 1/d'_i.$$

Now, using that for any $j \in \{1, \dots, t'\}$ we have $1/d'_j \leq 2 \min_{i=1}^{t'} 1/d'_i$, for any $i, j \in \{1, \dots, t'\}$ we get $1/d'_i \leq 2/d'_j$. Therefore,

$$\sum_{i, M'_i \in H_1} 1/d'_i \geq \left(\sum_{j, M'_j \in H \setminus H_1} 1/d'_j \right) / 2.$$

Now, we give an upper bound for $\sum_{i, M'_i \in H} 1/d'_i = \sum_{i, M'_i \in H_1} 1/d'_i + \sum_{j, M'_j \in H \setminus H_1} 1/d'_j$. We have the following.

$$(12\varepsilon)\text{ORS}_n((1-2\varepsilon)(1-\delta/\varepsilon)r)/\delta \geq 3 \sum_{i, M'_i \in H_1} 1/d'_i \geq \sum_{i, M'_i \in H_1} 1/d'_i + \sum_{j, M'_j \in H \setminus H_1} 1/d'_j.$$

Setting $\delta = \varepsilon/2, \varepsilon = 4\alpha/13$ we have:

$$\sum_{i, M'_i \in H} 1/d'_i = O(\text{ORS}_n((1-\alpha)r)).$$

Note that we showed an upper bound for the sum of the inverse average degree of each matching in one partition. Summing up the inverse average degree for all the partitions we get:

$$\sum_{i=1}^t 1/d_i = O(\text{ORS}_n((1-\alpha)r) \cdot \log n). \quad \square$$

Lemma 14. *If we set the threshold of [Algorithm 3](#) equal to $t = n^{1+\varepsilon/2} \sqrt{n \log n \cdot \text{ORS}_n(\Theta(\varepsilon^2 n))}$, then the total time per update is $O(\text{poly log } n) \cdot \sqrt{n^{1+\varepsilon} \log n \cdot \text{ORS}_n(\Theta(\varepsilon^2 n))}$.*

Proof. We run the $O(\log^2 n)$ parallel version of [Algorithm 2](#), and maintaining the contracted graphs takes $\text{poly log } n$ time. We will maintain the number of parallel edges in G' between any two contracted vertex in $\text{poly log } n$ time. Denote the set of contractions as \mathcal{C} in one phase of the algorithm.

Note that if the size of a contraction c is n_c , and the maximum matching in G is of order n_c we run the algorithm for this contraction. Using [Proposition 6](#) there is a reduction that preserves that matching in G with a factor of $1 - \varepsilon$. Now, integrating over the contractions in \mathcal{C} we find the corresponding contraction by running [Proposition 4](#). If the maximum matching in contraction c is M_c , this algorithm certifies the existence of a matching of size $|M_c|/2 < \mu \leq |M_c|$ in at least one contraction with n_c vertices. On the other hand, [Proposition 3](#) outputs subsets with a matching of size at least εn_c . This means that the ε -approximation matching found on each subset determined by [Proposition 3](#) is at least of size $\varepsilon^2 n$.

Now, since we run [Algorithm 2](#) on all contractions, nevertheless they contain the maximum matching or not, assuming n_c is the number of vertices in c , we have the total running time as:

$$\sum_{c \in \mathcal{C}} \sqrt{n_c^{1+\varepsilon} \log n_c \cdot \text{ORS}_{n_c}(\Theta(\varepsilon^2 n_c))} \leq O(\text{poly log } n) \cdot \sqrt{n^{1+\varepsilon} \log n \cdot \text{ORS}_n(\Theta(\varepsilon^2 n))}. \quad \square$$

Now, using [Lemma 11](#) and [Lemma 14](#) the proof of [Theorem 1](#) is complete.

5 Bounding Density of (Ordered) Ruzsa-Szemerédi Graphs

In this section we prove upper bounds on the density of ORS graphs. Our main result of this section is a proof of [Theorem 2](#) which we present in [Section 5.1](#). We then show that a much stronger upper bound can be proved if matchings cover more than half of vertices in [Section 5.2](#).

Before presenting our proof, let us first briefly discuss how the triangle removal lemma is useful for upper bounding density of RS graphs and why it does not seem to help for upper bounding ORS. The triangle removal lemma states that so long as the number of triangles in a graph are $o(n^3)$, then we can make the graph triangle free by removing $o(n^2)$ of its edges. Suppose we have a bipartite RS graph with induced matchings M_1, \dots, M_k . Add k vertices v_1, \dots, v_k to this graph and connect each v_i to all the vertices matched by M_i . Now, crucially, because each M_i is an induced matching, each v_i is part of exactly $|M_i|$ triangles, one for each edge of M_i . As such, the total number of triangles can be upper bounded by $O(kn) = O(n^2)$ which is well within the regime that one can apply triangle removal lemma. However, because the matchings in ORS are not induced matchings of the whole graph, the number of triangles cannot simply be upper bounded by $O(n^2)$ after adding the auxiliary vertices. In fact, our upper bound completely deviates from this approach and does not rely on the triangle removal lemma.

5.1 Linear Matchings

The following lemma, which is one of our main tools in proving [Theorem 2](#), shows that so long as vertex degrees are larger than a threshold, we can reduce the number of vertices rather significantly without hurting the size of a relatively large number of induced matchings. Our proof of [Lemma 15](#) builds on the techniques developed for upper bounding RS graphs with matchings of size very close to $n/4$ in [\[24\]](#) that we extend to ORS graphs with balanced degrees.

Lemma 15. *Let G be an $\text{ORS}_n(r, t)$ graph with $r = cn$ with even t . Let M_1, \dots, M_t be the t matchings in G as defined in [Definition 2](#). Let us partition these matchings into two subsets*

$$\mathcal{M} = \{M_1, \dots, M_{t/2}\} \quad \text{and} \quad \mathcal{M}' = \{M_{t/2+1}, \dots, M_t\}.$$

Suppose that for every vertex v , it holds that $\deg_{\mathcal{M}'}(v) \geq \delta ct$ for some $\delta > 0$. Then there exists an $\text{ORS}_{n'}(cn, t')$ graph H on $n' < (1 - \delta c^2/2)n$ vertices where $t' \geq \delta c^2 t / (8 \cdot 2^x)$ and $x = 4n/ct$.

Let us start with an observation that is extremely helpful in proving [Lemma 15](#).

Observation 16. *Take a matching $M' \in \mathcal{M}'$ and take any vertex v matched by M' . Let $N_{\mathcal{M}}(v)$ be the set of neighbors of v in \mathcal{M} . That is, $u \in N_{\mathcal{M}}(v)$ iff there is $M \in \mathcal{M}$ such that $(v, u) \in M$. Then no vertex in $N_{\mathcal{M}}(v)$ can be matched in M' .*

Proof. Suppose there is $u \in N_{\mathcal{M}}(v)$ that is matched by M' . Let $(u, w) \in M'$ be the matching edge involving u . Also take the edge $(v, y) \in M'$ that involves v (which exists by definition of v). Note that since the matchings M_1, \dots, M_t are edge-disjoint by [Definition 2](#), v and u cannot be matched together in M' (as v and u must be matched in some other matching in \mathcal{M} by definition of $N_{\mathcal{M}}(v)$). Now since $(u, w), (v, y) \in M'$ but (v, u) belongs to some matching in \mathcal{M} that comes before

M' in the ordering, matching M' cannot be an induced matching among its previous matchings, contradicting [Definition 2](#) for ORS graphs. \square

We are now ready to prove [Lemma 15](#).

Proof of Lemma 15. We explain the proof in a few steps.

Step 1: The pivots. We iteratively take a vertex of highest remaining degree in \mathcal{M} , and remove its neighbors in \mathcal{M} from the graph. More formally, take the graph $G_0 = (V_0, E_0)$ where $V_0 = V(G)$ and E_0 is the union of the matchings in \mathcal{M} . At step i , we choose v_i in V_{i-1} with the maximum degree in G_{i-1} . Let us define N_i as the neighbors of v_i in G_{i-1} . We then remove v_i and its neighbors in G_{i-1} to define $G_i = (V_i, E_i)$. Namely, $V_i = V_{i-1} \setminus (\{v_i\} \cup N_i)$ and E_i includes all edges in E_{i-1} except those that have at least one endpoint that does not belong to V_i .

Let k be the maximum integer such that $|N_k| > n/x$. We define $P = \{v_1, \dots, v_k\}$ to be the set of *pivots*. Note that since N_1, \dots, N_k are disjoint sets, we get that $n \geq \sum_{i=1}^k |N_i| \geq kn/x$ which implies that

$$|P| \leq x. \quad (1)$$

Moreover, note that when removing v_i , we remove $|N_i|$ vertices from the graph and each of those vertices has remaining degree at most $|N_i|$ (as we choose v_i to be the vertex of highest remaining degree). In other words, we remove at most $|N_i|^2$ from the graph after removing v_i and its remaining neighbors. On the other hand, after removing v_1, \dots, v_k and their neighbors, the maximum degree in the graph is at most n/x (by definition of k), and so there are at most n^2/x edges in the graph. This implies that:

$$|E_0| - \sum_{i=1}^k |N_i|^2 \leq n^2/x.$$

Given that E_0 includes $t/2$ edge disjoint matchings of size cn , we get that $|E_0| \geq tcn/2$. Plugged into the inequality above, this implies that

$$\sum_{i=1}^k |N_i|^2 \geq |E_0| - n^2/x \geq tcn/2 - n^2/x. \quad (2)$$

Step 2: Vertex reduction. Take a matching $M \in \mathcal{M}'$. Let S_M be the set of pivots matched by M , i.e., $S_M = P \cap V(M)$. Let us define $Y_M := V \setminus \cup_{v_i \in S} N_i$. From [Observation 16](#), we get that all edges of M must have both endpoints still in Y_M . Suppose for now that $M \in \mathcal{M}'$ is chosen uniformly at random. We first argue that the expected size of $|Y_M|$ is relatively small (despite it preserving all edges of M completely). We have

$$\begin{aligned} \mathbf{E}_{M \sim \mathcal{M}'}[|Y_M|] &= n - \sum_{i=1}^k \Pr[v_i \in V(M)] \cdot |N_i| \\ &= n - \sum_{i=1}^k \frac{\deg_{\mathcal{M}'}(v_i)}{|\mathcal{M}'|} \cdot |N_i| \\ &\quad \text{(As } M \text{ is chosen uniformly from } \mathcal{M}' \text{ and } v_i \text{ is matched in } \deg_{\mathcal{M}'}(v_i) \text{ of matchings in } \mathcal{M}'). \\ &= n - \sum_{i=1}^k \frac{2 \deg_{\mathcal{M}'}(v_i)}{t} \cdot |N_i| \quad \text{(Since } |\mathcal{M}'| = t/2.) \end{aligned}$$

$$\begin{aligned}
&\leq n - \sum_{i=1}^k \frac{2\delta ct}{t} \cdot |N_i| && \text{(Since } \deg_{\mathcal{M}'}(v_i) \geq \delta ct \text{ as assumed in Lemma 15.)} \\
&\leq n - \sum_{i=1}^k \frac{4\delta c}{t} \cdot |N_i|^2. && \text{(Since } |N_i| \leq t/2 \text{ as } \mathcal{M} \text{ is the union of } t/2 \text{ matchings.)} \\
&\leq n - \frac{4\delta c}{t} \cdot (tcn/2 - n^2/x) && \text{(By (2).)} \\
&= n - 2\delta c^2 n + \frac{4\delta cn^2}{tx} \\
&\leq n - 2\delta c^2 n + \frac{4\delta cn^2}{(4n/cx)x} && \text{(Since } t = 4n/cx \text{ by definition of } x \text{ in Lemma 15.)} \\
&= n - 2\delta c^2 n + \delta c^2 n \\
&= (1 - \delta c^2)n. && (3)
\end{aligned}$$

Instead of the expected value of Y_M , we need some (weak) concentration. Take $\varepsilon = \delta c^2/2$ and note from (3) that $(1 + \varepsilon) \mathbf{E}[|Y_M|] < (1 + \varepsilon)(1 - \delta c^2)n < (1 - \delta c^2/2)n$. We have

$$\begin{aligned}
\Pr_{M \sim \mathcal{M}'} \left[|Y_M| < (1 + \varepsilon) \mathbf{E}[|Y_M|] < (1 - \delta c^2/2)n \right] &= 1 - \Pr_{M \sim \mathcal{M}'} \left[|Y_M| > (1 + \varepsilon) \mathbf{E}[|Y_M|] \right] \\
&\geq 1 - \frac{1}{1 + \varepsilon} && \text{(By Markov's inequality.)} \\
&\geq \varepsilon/2 = \delta c^2/4. && (4)
\end{aligned}$$

We call a matching $M \in \mathcal{M}'$ a good matching if $|Y_M| < (1 - \delta c^2/2)n$. From (4) we get that at least $(\delta c^2/4)|\mathcal{M}'| = \delta c^2 t/8$ matchings are good.

Step 3: Grouping good matchings. Now, we partition the good matchings in \mathcal{M}' into $2^{|P|} \leq 2^x$ classes depending on which subset of the pivots they match. Since there are at least $\delta c^2 t/8$ good matchings, at least $\frac{\delta c^2 t}{8 \cdot 2^x}$ of them must belong to the same class C . Let $Y = Y_M$ for some $M \in C$. Note also that $Y = Y_{M'}$ for any other matching $M' \in C$ as well since they all match the same subset of pivots. Moreover, we have $|Y| < (1 - \delta c^2/2)n$ since M is good, and all matchings in C only match vertices in Y as discussed earlier (as a consequence of Observation 16). This implies that the graph H on vertex set Y and edge-set C (i.e., including all edges of all matchings in C) is an $\text{ORS}_{n'}(cn, t')$ graph for $n' < (1 - \delta c^2/2)n$ and $t' \geq \delta c^2 t/(8 \cdot 2^x)$ as desired. \square

The next lemma follows by applying Lemma 15 after carefully pruning vertices of low degree in \mathcal{M}' . Intuitively, Lemma 17 significantly reduces the number of vertices and shows that a relatively large number of matchings will have a lot of edges in the remaining graph.

Lemma 17. *Let G be an $\text{ORS}_n(cn, t)$ graph for even t . Then for some $\kappa \geq c^3/160$, there exist $\text{ORS}_{n'}(c'n', t')$ graphs such that $n' \leq n$, $c' \geq (1 + \kappa)c$ and $t' \geq t/2^{O(n/ct)}$.*

Proof. Let \mathcal{M} and \mathcal{M}' be defined for G as in Lemma 15. We would like to apply Lemma 15 iteratively to prove Lemma 17. However, Lemma 15 requires a lower bound on the degrees in \mathcal{M}' which might not necessarily hold for our graph G . Therefore to be able to use it, we first have to prune vertices of small degrees.

Let G'_0 be the subgraph of G only including the edges of the matchings in \mathcal{M}' . Let $\delta = 1/4$. We iteratively remove vertices of low degree. Formally, in step i , if there is a vertex v_i with degree less than $2\delta ct$ in G'_{i-1} , we define the graph G'_i of the next step to be graph obtained after removing v_i and its edges from G'_{i-1} . Let G'_k be the final graph that does not have any vertex of degree smaller than $2\delta ct$.

Suppose that $k = \alpha n$; that is, we remove αn vertices in the process above. We consider two cases depending on the value of α .

Case 1: $\alpha \geq \delta c^3/10$. Since we remove at most δct edges in every step, the total number of removed edges is at most $\delta \alpha ct n$ over the $k = \alpha n$ steps of constructing G'_k . Say a matching $M \in \mathcal{M}'$ is *damaged* if a total of at least $3\delta \alpha cn$ of its edges have been removed. In other words, M is damaged if less than $|M| - 3\delta \alpha cn = (1 - 3\delta \alpha)cn$ of its edges belong to G'_k . Since the total number of edges removed is at most $\delta \alpha ct n$ and each damaged matching has at least $3\delta \alpha cn$ of its edges removed, the total number of damaged matchings can be upper bounded by $\delta \alpha ct n / 3\delta \alpha cn = t/3$. Since $|\mathcal{M}'| = t/2$, at least $t/6$ matchings in \mathcal{M}' are not damaged, and have size at least $(1 - 3\delta \alpha)cn$. These matchings themselves form an $\text{ORS}_{n'}(c'n', t/6)$ graph with parameters

$$n' = (1 - \alpha)n \leq (1 - \delta c^3/10)n \stackrel{(\delta=1/4)}{=} (1 - c^3/40)n,$$

and

$$c'n' \geq (1 - 3\delta \alpha)cn = (1 - 3\delta \alpha)c \frac{n'}{1 - \alpha} \stackrel{(\delta=1/4)}{=} \left(1 - \frac{3}{4}\alpha\right) c \frac{n'}{1 - \alpha} \geq \left(1 + \frac{1}{4}\alpha\right) cn' \geq (1 + c^3/160)cn',$$

which implies that $c' \geq (1 + c^3/160)c$. Taking $\kappa = c^3/160$ proves the lemma in this case.

Case 2 – $\alpha < \delta c^3/10$: Let G_k be the graph G after removing vertices v_1, \dots, v_k (the difference between G_k and G'_k is that G'_k only includes the edges of \mathcal{M}' but G_k includes both edges of \mathcal{M}' and \mathcal{M} that do not have any endpoint removed). Since we remove a total of αn vertices from G to obtain G_k , each matching in \mathcal{M}' and \mathcal{M} loses at most αn edges. Therefore, G_k is an $\text{ORS}_{n''}(c''n'', t'')$ graph with parameters

$$n'' = (1 - \alpha)n, \quad c'' \geq \frac{cn - \alpha n}{n''} = \frac{(c - \alpha)n}{(1 - \alpha)n} \geq c - \alpha \geq (1 - \delta c^2/10)c, \quad t'' = t. \quad (5)$$

Additionally, by the construction of G'_k , all vertices v in G_k satisfy

$$\deg_{\mathcal{M}'}(v) \geq 2\delta ct \geq \delta c''t''$$

(Since $t'' = t$ and $c'' \leq \frac{cn}{n''} = \frac{cn}{(1 - \alpha)n} \leq 2c$ where the last inequality follows from $\alpha < 1/2$.)

This now satisfies the requirements of [Lemma 15](#). Applying it on graph G_k , we obtain an $\text{ORS}_{n'}(c'n', t')$ graph where the number of vertices satisfies

$$n' < (1 - \delta c''^2/2)n'', \quad (6)$$

the number of matchings satisfies

$$t' \geq \delta c''^2 t'' / (8 \cdot 2^{4n''/c''t''}) = \delta c \cdot t / 2^{O(n/ct)} = t / 2^{O(n/ct)},$$

and finally since [Lemma 15](#) does not change the size of matchings, we get

$$c'n' = c''n''$$

$$\geq \frac{c''n'}{(1 - \delta c''^2/2)} \quad (\text{By (6).})$$

$$\geq (1 + \delta c''^2/2)c''n'$$

$$\geq \left(1 + \delta \left((1 - \delta c^2/10)c\right)^2/2\right) \left((1 - \delta c^2/10)c\right)n',$$

(Since the RHS is minimized when c'' is minimized, thus we can replace c'' with its LB from (5).)

$$\geq \left(1 + 0.4\delta c^2\right) \left((1 - \delta c^2/10)c\right)n' \quad (\text{Since } \delta((1 - \delta c^2/10)c)^2/2 \geq \delta(0.9c)^2/2 > 0.4\delta c^2)$$

$$> (1 + 0.07c^2)cn'.$$

Dividing both sides of the inequality by n' implies $c' \geq (1 + 0.07c^2)c$. Taking $\kappa = 0.07c^2 \geq c^3/160$ implies the lemma. \square

We are now ready to prove [Theorem 2](#).

Proof of Theorem 2. We prove that for some $d = \text{poly}(1/c)$, there does not exist any $\text{ORS}_n(cn, t)$ graph G with $t \geq n/\log_b^{(d)} n$ where $b = 2^{\beta/c}$ for some large enough constant $\beta \geq 1$. Note that the base of the iterated log is different from the statement of the theorem, but since $\log_b^{(x)} z = \log^{(\Theta(x \log^* b))} z$, this implies the theorem as well by letting $\ell = \Theta(d \cdot \log^* b) = \text{poly}(1/c)$.

Suppose for the sake of contradiction that there exists an $\text{ORS}_n(cn, t)$ graph G with $t \geq n/\log_b^{(d)} n$. Let κ be as in [Lemma 17](#). We iteratively apply [Lemma 17](#) for $k = \log_2(1/c)/\kappa = \text{poly}(c)$ steps to obtain a sequence of graphs $G_0, G_1, G_2, G_3, \dots, G_k$ where $G_0 = G$ is the original graph, G_1 is obtained by applying [Lemma 17](#) on G_0 , G_2 is obtained by applying [Lemma 17](#) on G_1 , and so on so forth.

Let us now analyze the ORS properties of graph G_k , starting with the parameter c_k . Since every application of [Lemma 17](#) multiplies the parameter c_i by a factor of at least $(1 + \kappa)$, we have

$$\begin{aligned} c_k &\geq (1 + \kappa)^k c \\ &= (1 + \kappa)^{\log_2(1/c)/k} c \\ &\geq 2^{\log_2(1/c)} c \quad (\text{Since } (1 + \kappa)^{1/\kappa} \geq 2 \text{ for all } 0 < \kappa \leq 1) \\ &\geq 1. \end{aligned}$$

This implies that in graph G_k , there must be t_k edge-disjoint matchings of size $c_k n_k \geq n_k$, but each matching in a graph on n_k vertices can have size at most $n_k/2$. In other words, we must have $t_k = 0$. We will obtain a contradiction by proving that $t_k \geq 1$.

We prove by induction that for every $i \in [d]$,

$$t_i \geq \frac{n}{\log_b^{(d-i)} n}.$$

For the base case $i = 0$ this holds as assumed at the beginning of the proof. By choosing appropriately large β , we know by [Lemma 17](#) that,

$$\begin{aligned} t_i &\geq t_{i-1}/2^{0.5\beta(n_{i-1}/c_{i-1}t_{i-1})} \\ &\geq t_{i-1}/2^{0.5\beta(n/ct_{i-1})} \quad (\text{Since } n_{i-1} \leq n \text{ and } c_{i-1} \geq c \text{ as we iteratively apply Lemma 17.}) \\ &= t_{i-1}/b^{0.5(n/t_{i-1})} \quad (\text{Since we defined } b = 2^{\beta/c}.) \end{aligned}$$

$$\begin{aligned}
&\geq \frac{n}{\log_b^{(d-i+1)} n} \cdot b^{-0.5 \left(n / \frac{n}{\log_b^{(d-i+1)} n} \right)} && \text{(By the induction hypothesis } t_{i-1} \geq \frac{n}{\log_b^{(d-i+1)} n} \text{.)} \\
&= \frac{n}{\log_b^{(d-i+1)} n} \cdot b^{-0.5 \log_b^{(d-i+1)} n} \\
&= \frac{n}{\log_b^{(d-i+1)} n \cdot \sqrt{\log_b^{(d-i)} n}} \\
&\geq \frac{n}{\log_b^{(d-i)} n},
\end{aligned}$$

concluding the proof.

Now let $d > k = \text{poly}(1/c)$. From the above, we get that $t_k \geq n/(\log_b^{(d-k)}) \geq 1$, which as discussed is a contradiction. \square

5.2 When Matchings are (Very) Large

The goal of this section is to prove that the number of matching of an ORS graph with $r > n/4$ is bounded by a constant value. More precisely we prove [Theorem 3](#). An equivalent of this theorem was already known for RS graphs (see [\[24\]](#)), and we show that almost the same proof carries over to ORS graphs as well.

Theorem 3. *For any $c > 1/4 + \varepsilon$, $\text{ORS}_n(cn, t) \leq 1/\varepsilon + 1$.*

First, we argue that the number of common vertices of any two matching in ORS is at most r , using this we can prove that a constant value bounds the number of matching since the number of subsets of length $2r$ with at most r common vertices between any two sets.

Lemma 18. *Suppose G is an $\text{ORS}(r, t)$ graph on n vertices and let M_1, \dots, M_t be the corresponding ordered list of matchings as in [Definition 2](#). Then for any $i \neq j$ in $[t]$, we have $|V(M_i) \cap V(M_j)| \leq r$.*

Proof. Let us assume w.l.o.g. that $i < j$ and suppose that $|V(M_i) \cap V(M_j)| \geq r + 1$. By the pigeonhole principle, this means that there must be an edge in M_i whose both endpoints are matched in M_j , contradicting the fact from [Definition 2](#) that M_j is an induced matching in $M_1 \cup \dots \cup M_j$. \square

Now, we use a similar approach as [\[24\]](#) to prove that $t < 1/\varepsilon$. This theorem only uses the fact that the number of common vertices of two matchings is at most r .

Proof of Theorem 3. Using [Lemma 18](#) we have that the distance between sets of vertices of any two matchings is at most r , this implies a constant upper bound for the size of ORS using bounds from coding theory. To improve the upper bound we set $v_i \in \{0, 1\}^n$, a vector that represents vertices in $V(M_i)$. Let us use a_i to denote the number of vectors v_j with 0 in their i -th coordinate and let b_i be the number of vectors with 1 in their i -th coordinate. If we set v_0 a vector of all zeros. Note that we have $a_i + b_i = t + 1$. Now, by double counting we have:

$$2r \binom{t+1}{2} \leq \sum_{0 \leq i < j \leq t} \text{dist}(v_i, v_j) = \sum_{i=1}^n a_i b_i. \quad (7)$$

By $\text{dist}(v_i, v_j)$ we mean the Hamming distance which satisfies the following inequality:

$$\text{dist}(v_i, v_j) = |V_i| + |V_j| - 2|V_i \cap V_j| \geq 2r.$$

Note that in (7), $a_i b_i$ maximizes when we have $a_i = b_i = (t+1)/2$ for odd t and $a_i = (t+2)/2, b_i = t/2$ for even t . This concludes that $t < 1/\varepsilon + 1$ \square

Acknowledgements

The first author thanks Sepehr Assadi, Sanjeev Khanna, Huan Li, Ray Li, Mohammad Roghani, and Aviad Rubinfeld for helpful discussions about RS graphs and their applications in dynamic graphs over the years.

References

- [1] Moab Arar, Shiri Chechik, Sarel Cohen, Cliff Stein, and David Wajc. Dynamic Matching: Reducing Integral Algorithms to Approximately-Maximal Fractional Algorithms. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 7:1–7:16, 2018.
- [2] Sepehr Assadi. A two-pass (conditional) lower bound for semi-streaming maximum matching. 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 708–742. SIAM, 2022.
- [3] 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.
- [4] Surender Baswana, Manoj Gupta, and Sandeep Sen. Fully Dynamic Maximal Matching in $O(\log n)$ Update Time. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 383–392. IEEE Computer Society, 2011.
- [5] 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.
- [6] 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.
- [7] 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.

- [8] Soheil Behnezhad and Sanjeev Khanna. New Trade-Offs for Fully Dynamic Matching via Hierarchical EDCS. In *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.
- [9] Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Cliff Stein, and Madhu Sudan. Fully Dynamic Maximal Independent Set with Polylogarithmic Update Time. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 382–405. IEEE Computer Society, 2019.
- [10] Soheil Behnezhad, Jakub Lacki, and Vahab S. Mirrokni. Fully Dynamic Matching: Beating 2-Approximation in Δ^ϵ Update Time. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2492–2508. SIAM, 2020.
- [11] Aaron Bernstein and Cliff Stein. Fully Dynamic Matching in Bipartite Graphs. In *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.
- [12] Aaron Bernstein and Cliff Stein. Faster Fully Dynamic Matchings with Small Approximation Ratios. In *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.
- [13] Aaron Bernstein, Sebastian Forster, and Monika Henzinger. A Deamortization Approach for Dynamic Spanner and Dynamic Maximal Matching. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1899–1918, 2019.
- [14] Aaron Bernstein, Aditi Dudeja, and Zachary Langley. A Framework for Dynamic Matching in Weighted Graphs. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2021, to appear, 2021*.
- [15] Sayan Bhattacharya and Peter Kiss. Deterministic Rounding of Dynamic Fractional Matchings. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, pages 27:1–27:14, 2021.
- [16] Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. New Deterministic Approximation Algorithms for Fully Dynamic Matching. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 398–411. ACM, 2016.
- [17] Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. Fully Dynamic Approximate Maximum Matching and Minimum Vertex Cover in $O(\log^3 n)$ Worst Case Update Time. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 470–489. SIAM, 2017.
- [18] Sayan Bhattacharya, Monika Henzinger, and Giuseppe F. Italiano. Deterministic Fully Dynamic Data Structures for Vertex Cover and Matching. *SIAM J. Comput.*, 47(3):859–887, 2018.

- [19] Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. Dynamic $(1 + \epsilon)$ -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.
- [20] 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.
- [21] Moses Charikar and Shay Solomon. Fully Dynamic Almost-Maximal Matching: Breaking the Polynomial Worst-Case Time Barrier. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 33:1–33:14, 2018.
- [22] Eldar Fischer, Eric Lehman, Ilan Newman, Sofya Raskhodnikova, Ronitt Rubinfeld, and Alex Samorodnitsky. Monotonicity testing over general poset domains. In John H. Reif, editor, *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 474–483. ACM, 2002.
- [23] Jacob Fox. A new proof of the graph removal lemma. *Annals of Mathematics*, pages 561–579, 2011.
- [24] Jacob Fox, Hao Huang, and Benny Sudakov. On graphs decomposable into induced matchings of linear sizes, 2015.
- [25] 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.
- [26] Fabrizio Grandoni, Chris Schwiegelshohn, Shay Solomon, and Amitai Uzzad. Maintaining an EDCS in General Graphs: Simpler, Density-Sensitive and with Worst-Case Time Bounds. In *5th Symposium on Simplicity in Algorithms, SOSA@SODA 2022, Virtual Conference, January 10-11, 2022*, pages 12–23. SIAM, 2022.
- [27] Manoj Gupta and Richard Peng. Fully Dynamic $(1 + \epsilon)$ -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.
- [28] 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.
- [29] Peter Kiss. Deterministic Dynamic Matching in Worst-Case Update Time. In *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.

- [30] Yang P. Liu. On approximate fully-dynamic matching and online matrix-vector multiplication. *CoRR*, abs/2403.02582, 2024.
- [31] 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 International Workshop 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.
- [32] Ofer Neiman and Shay Solomon. Simple deterministic algorithms for fully dynamic maximal matching. In *Symposium on Theory of Computing Conference, STOC’13, Palo Alto, CA, USA, June 1-4, 2013*, pages 745–754, 2013.
- [33] Krzysztof Onak and Ronitt Rubinfeld. Maintaining a large matching and a small vertex cover. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 457–464. ACM, 2010.
- [34] Mohammad Roghani, Amin Saberi, and David Wajc. Beating the Folklore Algorithm for Dynamic Matching. In *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPIcs*, pages 111:1–111:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
- [35] Shay Solomon. Fully Dynamic Maximal Matching in Constant Update Time. In *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.
- [36] Shay Solomon. Fully dynamic maximal matching in constant update time. In *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.
- [37] David Wajc. Rounding Dynamic Matchings Against an Adaptive Adversary. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 194–207. ACM, 2020.