An Asymptotically Optimal Algorithm for Maximum Matching in Dynamic Streams

Sepehr Assadi /envel▷/home
Department of Computer Science, Rutgers University, Piscataway, NJ, USA

Vihan Shah /envel▷
Department of Computer Science, Rutgers University, Piscataway, NJ, USA

Abstract

We present an algorithm for the maximum matching problem in dynamic (insertion-deletions) streams with asymptotically optimal space: for any \( n \)-vertex graph, our algorithm with high probability outputs an \( \alpha \)-approximate matching in a single pass using \( O(n^2/\alpha^3) \) bits of space.

A long line of work on the dynamic streaming matching problem has reduced the gap between space upper and lower bounds first to \( n^{o(1)} \) factors [Assadi-Khanna-Li-Yaroslavtsev; SODA 2016] and subsequently to polylog(\( n \)) factors [Dark-Konrad; CCC 2020]. Our upper bound now matches the Dark-Konrad lower bound up to \( O(1) \) factors, thus completing this research direction.

Our approach consists of two main steps: we first (provably) identify a family of graphs, similar to the instances used in prior work to establish the lower bounds for this problem, as the only “hard” instances to focus on. These graphs include an induced subgraph which is both sparse and contains a large matching. We then design a dynamic streaming algorithm for this family of graphs which is more efficient than prior work. The key to this efficiency is a novel sketching method, which bypasses the typical loss of polylog \( (n) \)-factors in space compared to standard \( L_0 \)-sampling primitives, and can be of independent interest in designing optimal algorithms for other streaming problems.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Approximation algorithms analysis; Theory of computation → Sketching and sampling

Keywords and phrases Graph streaming algorithms, Sketching, Maximum matching

Funding Sepehr Assadi: Research supported in part by a NSF CAREER Grant CCF-2047061, and a gift from Google Research.
Vihan Shah: Research supported in part by a NSF CAREER Grant CCF-2047061.

Acknowledgements We thank the anonymous reviewers of ITCS 2022 for their many insightful suggestions that helped with improving the presentation of this paper.

1 Introduction

We study the maximum matching problem in the dynamic streaming setting. In this problem, the edges of an input graph \( G = (V, E) \) are presented to the algorithm as a sequence of both edge insertions and deletions. The goal is to recover an approximate maximum matching of \( G \) at the end of the stream using a limited space smaller than the input size, namely, \( o(n^2) \) space where \( n \) is the number of vertices. The (dynamic) graph streaming model is highly motivated by applications to processing massive graphs and has been studied extensively in recent years; see, e.g. [1, 2, 28, 31, 10, 36, 7, 6, 24, 38, 5, 8, 29, 17, 32] and references therein.

* A full version of the paper with the same title appears on arXiv.
A brief note on the history of dynamic streaming matching is in order. Initiated by a breakthrough of [1], for most graph problems studied in insertion-only streams, researchers were able to subsequently obtain algorithms with similar guarantees in dynamic streams as well; this includes connectivity [1], cut sparsifiers [2], spectral sparsifiers [28], densest subgraph [36], subgraph counting [2], $(\Delta + 1)$-vertex coloring [5], among many others. This placed the maximum matching problem in a rather unique position in the literature: while there is a straightforward 2-approximation algorithm for this problem in insertion-only streams using only $O(n \log n)$ space [20], no non-trivial dynamic streaming algorithms were developed for this problem even in $o(n^2)$ space, despite significant attention; see, e.g. [9, 15, 14].

This problem was addressed in a series of (independent and concurrent) work [31, 7, 13]. In particular, [7] proved that $\alpha$-approximation for matching in dynamic streams requires $(n^{2-o(1)}/\alpha^3)$ space and designed an $\alpha$-approximation algorithm with $O(n^2/\alpha^3 \cdot \text{polylog}(n))$ space ([31] gave a slightly weaker lower and upper bounds for this problem and [13] obtained an algorithm with similar performance as [7]). The work of [7] thus brought the gap between space upper and lower bounds on this problem down to an $n^{o(1)}$ factor. The lower bound of [7] relied on a remarkable characterization of dynamic streaming algorithms due to [33, 3] that allows for transforming linear sketching lower bounds to dynamic streams. However, this characterization requires making strong requirements from the streaming algorithms (such as processing doubly exponentially long streams); see [25] for a detailed discussion on this topic. More recently, [17] bypassed this characterization step entirely and along the way, improved the lower bound for this problem to $\Omega(n^2/\alpha^3)$ space directly in dynamic streams. This constitutes the state-of-the-art for the dynamic streaming matching problem.

In parallel to this line of work on the matching problem that focused on determining the “high order terms” in the space complexity (namely, up to $n^{o(1)}$ or $\text{polylog}(n)$ factors), there has also been substantial work on determining the “lower order terms” on space complexity of other dynamic graph streaming problems [42, 30, 38, 43]. For instance, [38], building on [30], proved that any dynamic streaming algorithm for connectivity requires $\Omega(n \cdot \log^3(n))$ space which matches the algorithm of [1] up to constant factors. This quest for obtaining asymptotically optimal bounds is common in the streaming literature beyond graph streams such as in frequency moment estimation [26, 27, 34, 4, 11], empirical entropy [21, 12, 22], numerical linear algebra [16], compressed sensing [39, 40], and sampling [30].

This state of affairs is the motivation of our work: Can we determine the space complexity of the maximum matching problem down to its lower order terms in dynamic streams? We resolve this question in the affirmative by presenting an improved algorithm for this problem.

**Main Result (Formalized in Theorem 14).** There is a dynamic streaming algorithm that with high probability outputs an $\alpha$-approximation to maximum matching using $O(n^2/\alpha^3)$ space for any $\alpha \ll n^{1/2}$.

Let us right away note that the condition of $\alpha \ll n^{1/2}$ in our main result is not arbitrary\(^1\): for $\alpha > n^{1/2}$, we have $n^2/\alpha^3 < n/\alpha$, while one needs $\Omega((n/\alpha) \cdot \log n)$ space simply to store an $\alpha$-approximate matching! As a result, our algorithm now matches the lower bound of [17] up to constant factors in almost the entirety of its meaningful regime for parameter $\alpha$, thus completely resolving the space complexity of the maximum matching problem in dynamic streams. We now discuss further aspects of our work.

\(^1\) This condition is in fact used in all prior lower bounds in [31, 7, 17] as well as algorithms [31, 13] with the exception of algorithm of [7].
Beyond $L_0$-samplers. The key technique\(^2\) in dynamic graph streams is the use of $L_0$-samplers that allow for sampling an edge from a sequence of edges that undergo insertions and deletions (see Section 2.2).

Previously-best algorithms of [7, 13] for dynamic streaming matching sample $O(n^2/\alpha^3)$ edges from the input graph (from a carefully-designed non-uniform distribution) and show that this sample contains an $\alpha$-approximate matching. For the sampling, they need to use $L_0$-samplers that will bring in an additional $\text{poly}\ log(n)$ factor overhead in the space. At the same time, a careful examination of the lower bound of [17] shows that one needs to recover $\Omega(n^2/\alpha^3)$ edges from the graph (not only bits, assuming one only communicates edges). On top of this, the lower bound of [38] for the connectivity problem is based on showing that recovering $(n-1)$ edges of a spanning forest in the input, essentially require paying the cost of $(n-1) L_0$-samplers as well, leading to their $\Omega(n \cdot \log^3(n))$ lower bound. Putting all this together, it is natural to conjecture that one also needs $\Omega(n^2/\alpha^3 \cdot \text{poly}\ log(n))$ space for the matching problem\(^3\).

Our algorithm in this paper is still based on finding $\Theta(n^2/\alpha^3)$ edges from the input graph. It turns out however that one can do this more efficiently than using the same number of $L_0$-samplers. In particular, we show a way of recovering these edges with only $O(1)$ bit overhead per edge on average. This is achieved using a novel sketching primitive in this paper (Section 3). On a high level, this sketch allows us to recover sparse induced subgraphs of the input graph, specified to the algorithm only at the end stream, in a more efficient manner than recovering them one edge at a time via $L_0$-samplers\(^4\). We believe this idea can be useful for obtaining asymptotically optimal algorithms for other dynamic streaming problems as well.

Classifying input graphs. Another key idea in our paper is a way of roughly classifying input graphs into “easy” and “hard” instances. Informally speaking, the easy instances are the ones that one can recover a large matching from them by sampling $\ll n^2/\alpha^3$ edges (again, in a non-uniform way). Such a graph can then be handled in $O(n^2/\alpha^3)$ space even if we use $L_0$-samplers for our sampling given, we now need much fewer number of samples than before. One of our two main lemmas (Lemma 16) gives one characterization of these graphs: essentially, any “hard” graph, i.e., a one not solvable by the above approach, includes a subgraph on $n-o(n/\alpha)$ vertices with only $\approx n$ edges and a matching of size $\approx n-o(n/\alpha)$ (they essentially have an induced matching of size $\approx n-o(n/\alpha)$). A reader familiar with [31, 7, 17] may notice that this family of graphs is precisely the one used in all prior lower bounds for the dynamic streaming matching problem.

Our next main lemma (Lemma 17) then gives an algorithm for solving the hard graphs. The idea behind the algorithm is as follows. Let us use $S$ to denote the vertices in the induced sparse subgraph of the input and let $T$ be the remaining vertices (we will be able to recover an approximate version of this partitioning at the end of the stream). If we are able to recover edges inside $S$, we will be done as there is a large matching in $S$ and it does not have too many extra edges. The problem is that we will not know this set until the end of the stream and by that point we should have collected all the required

\(^2\) We are only aware of a single work [28] in dynamic graph streams that does not use $L_0$-samplers.

\(^3\) This was in fact the authors’ conjecture at the beginning of this project.

\(^4\) Let us note that our sketch cannot do magic: The problem of finding sparse induced subgraphs is at the core of the lower bound approaches for dynamic streaming matching in [7, 17], thus there is no hope of solving it “efficiently”. Our sketch shows that one can recover these graphs without paying any extra cost over the lower bounds of these work.
information. This is where our main sketching tool mentioned earlier comes into place. Informally, the sketch allows us to, for any vertex \( v \in S \), recover the neighbors \( N(v) \) of \( v \) using roughly \( |N(v) \cap T| + (|N(v) - T| \cdot \text{poly log } (n)) \) bits (as opposed to \( |N(v)| \cdot \text{polylog } (n) \) bits via \( L_0 \)-samplers). As the total number of edges outside \( T \) is quite small \( \approx n \) in total, this is huge saving for us that allows for obtaining our desired \( O(n^2/\alpha^3) \) bit upper bounds.

We shall remark that in this discussion, we have been imprecise to give a rough intuition of our approach; the actual details turn out to be considerably more challenging as described in Section 5 and Section 6.

“Shaving” log-factors? Finally, we emphasize that our improvement over prior work in [7, 13] at no place is obtained via “shaving log-factors”. Indeed, there is a considerable gap of \( n \Omega(1) \) factor between the parameters that our easy-graph algorithms and hard-graph algorithms can still handle within \( O(n^2/\alpha^3) \) space. This in turn allowed us to be quite cavalier with the parameters (e.g., using \( \log n \)-factors or \( n \Omega(1) \)-factors where constant or \( \text{poly log } (n) \) sufficed) and still recover an optimal space bound.

2 Preliminaries

Notation. For a graph \( G = (V, E) \), we write \( \text{vec}(E) \) to denote the \( \binom{n}{2} \)-dimensional vector where \( \text{vec}(E)_i \) denotes the multiplicity of the edge \( e_i \) in \( G \). We use \( \text{deg}(v) \) and \( N(v) \) for each vertex \( v \in V \) to denote the degree and neighborhood of \( v \), respectively. For a subset \( F \) of edges in \( E \), we use \( V(F) \) to denote the vertices incident on \( F \); similarly, for a set \( U \) of vertices in \( V \), \( E(U) \) denotes the edges incident on \( U \).

Throughout, we will use the term “with high probability” to mean with probability at least \( 1 - 1/n^c \) for some large constant \( c > 0 \). The constant \( c \) can be made arbitrarily large by only increasing the space of our algorithms with a constant factor and thus within the same asymptotic bounds. Moreover, for our purpose, this probability is large enough that one can always do a union bound over at most \( \text{poly}(n) \) different events that we consider in this paper; we do not necessarily mention this each time.

2.1 Dynamic (Graph) Streams and Linear Sketches

The dynamic streaming model is defined formally as follows.

**Definition 1 (Dynamic (graph) streams).** A dynamic stream \( \sigma = (\sigma_1, \ldots, \sigma_N) \) defines a vector \( x \in \mathbb{R}^m \). Each entry of the stream is a tuple \( \sigma_i = (j_i, \Delta_i) \) for \( j_i \in [m] \) and \( \Delta_i \in \{-1, +1\} \). The vector \( x \) is defined as:

\[
  \text{for all } j \in [m]: \quad x_j = \sum_{\sigma_i: j_i = j} \Delta_i.
\]

A dynamic graph stream is a dynamic stream wherein \( m = \binom{n}{2} \) and \( x = \text{vec}(E) \) for the graph \( G = (V, E) \) with \( V = [n] \). Each update to \( \text{vec}(E) \) corresponds to inserting or deleting the specified edge from the graph.

A dynamic streaming algorithm makes a single pass over updates to \( x \) and uses a limited memory, measured in number of bits, and output an answer to the given problem at the end of the stream.

Similar to virtually all other dynamic streaming algorithms, our algorithms will also be based on linear sketches, defined as follows.
Definition 2 (Linear sketch). Let \( \Pi \) be a problem defined over vectors \( x \in \mathbb{R}^m \) (e.g., return the \( \ell_2 \)-norm of \( x \)). A linear sketch for \( \Pi \) is an algorithm defined by the following pair:
- **sketching matrix:** A matrix \( \Phi \in \text{poly}(m)^{s \times m} \) that can be random and implicit;
- **recovery algorithm:** An algorithm that given the sketching matrix \( \Phi \) and the vector \( \Phi \cdot x \), returns a solution to \( \Pi(x) \).

We refer to the vector \( \Phi \cdot x \) as a sketch of \( x \), and to the number of bits needed to store \( \Phi \) (implicitly) and \( \Phi \cdot x \) as the size of the linear sketch.

The linear sketch for an input \( x \) then consists of sampling a sketching matrix \( \Phi \) (independent of \( x \)), computing the sketch \( \Phi \cdot x \), and running the recovery algorithm on the sketch to solve the problem.

(We note that the computations can be on the set of integers, reals, or on finite fields.)

For our purpose in this paper, we typically focus on graph problems for the choice of \( \Pi \) in Definition 2 and then \( x = \text{vec}(E) \) where \( E \) is the edge-set of the input graph. The following proposition is well-known and is omitted here.

Proposition 3. Let \( \Pi \) be a problem defined over vectors \( x \in \mathbb{R}^m \). Suppose there exists a linear sketch of size \( s(n) \) for \( \Pi \) with probability of success \( p(n) \). Then, there is also a streaming algorithm for solving \( \Pi \) on dynamic streams defining \( x \) with probability of success \( p(n) \) using \( O(s(n) + \log m) \) bits of space.

Given Proposition 3, in the rest of the paper, we simply focus on designing linear sketches for our dynamic streaming problems.

2.2 Standard Sketching Toolkit

We will also use \( L_0 \)-samplers, a powerful tool used by most dynamic graph streaming algorithms, in our paper. The goal of \( L_0 \)-samplers is to solve the following basic problem.

Problem 1 (\( L_0 \)-Sampling). Given a vector \( x \in \mathbb{R}^m \) specified in a dynamic stream, sample \( x_i \) uniformly at random from the support of \( x \) at the end of the stream.

We will typically use \( L_0 \)-samplers by applying them to different pre-specified subsets of edges of the underlying graph to sample a uniform edge from those subsets.

Proposition 4 ([23, 30]). There is a linear sketch, called \( L_0 \)-Sampler, for Problem 1 with size
\[
s_{L0} = s_{L0}(m, \delta_F, \delta_E) = O(\log m \cdot (\log m \cdot \log(1/\delta_F) + \log(1/\delta_E)))
\]
bits, that outputs \text{FAIL} with probability at most \( \delta_F \) and outputs a wrong answer with probability at most \( \delta_E \).

Another standard tool we use is sparse-recovery (from both sketching and compressed sensing literature). The goal of sparse recovery is to solve the following problem.

Problem 2 (Sparse Recovery). Given an integer \( k \geq 1 \) and a vector \( x \in \mathbb{R}^m \) specified in a dynamic stream with the promise that \( \|x\|_0 \leq k \), recover all of \( x \) at the end of the stream.

We use the following standard result on sparse recovery over finite fields.

Proposition 5 (c.f. [18]). Let \( q \) be any prime number and \( k \geq 1 \) be an arbitrary integer. There is a deterministic (poly-time computable) linear sketch, called \( \text{Sparse-Recovery} \), for Problem 2 for \( k \)-sparse vectors \( x \in \mathbb{F}_q^m \), with size
\[
s_{SR} = s_{SR}(m, k, q) = O(k \cdot \log m \cdot \log q)
\]
bits that always outputs the correct answer on \( k \)-sparse vectors. Moreover, all computations of this linear sketch are also performed over the field \( \mathbb{F}_q \).
We shall note that for our application, we actually need the “moreover” part of Proposition 5 (which limits the use of more standard approaches).

2.3 Probabilistic Tools

We use the following standard concentration inequalities. The first is a standard form of Chernoff bounds.

► Proposition 6 (Chernoff bound; c.f. [19]). Suppose $X_1, \ldots, X_m$ are $m$ independent random variables with range $[0, 1]$ each. Let $X := \sum_{i=1}^{m} X_i$ and $\mu_L \leq \mathbb{E}[X] \leq \mu_H$. Then, for any $\varepsilon > 0$,

$$\Pr(X > (1 + \varepsilon) \cdot \mu_H) \leq \exp\left(-\frac{\varepsilon^2 \cdot \mu_H}{3 + \varepsilon}\right) \quad \text{and} \quad \Pr(X < (1 - \varepsilon) \cdot \mu_L) \leq \exp\left(-\frac{\varepsilon^2 \cdot \mu_L}{2 + \varepsilon}\right).$$

We also need McDiarmid’s inequality when there is non-trivial correlation between random variables.

► Proposition 7 (McDiarmid’s inequality [35]). Let $X_1, \ldots, X_m$ be $m$ independent random variables where each $X_i$ has some range $\mathcal{X}_i$. Let $f : \mathcal{X}_1 \times \cdots \times \mathcal{X}_m \to \mathbb{R}$ be any $\varepsilon$-Lipschitz function meaning that for all $i \in [m]$ and all choices of $(x_1, \ldots, x_m), (x'_1, \ldots, x'_m) \in \mathcal{X}_1 \times \cdots \times \mathcal{X}_m$,

$$|f(x_1, \cdots, x_{i-1}, x_i, x_{i+1}, \cdots, x_m) - f(x_1, \cdots, x_{i-1}, x'_i, x_{i+1}, \cdots, x_m)| \leq \varepsilon.$$

Then, for all $b > 0$,

$$\Pr(|f(X_1, \ldots, X_m) - \mathbb{E}[f(X_1, \ldots, X_m)]| \geq b) \leq 2 \cdot \exp\left(-\frac{2b^2}{m \cdot \varepsilon^2}\right).$$

Finally, in certain places, we also use limited independence hash functions in our algorithms to reduce their space complexity.

► Definition 8 (Limited-independence hash functions). For integers $n, m, k \geq 1$, a family $\mathcal{H}$ of hash functions from $[n]$ to $[m]$ is called a $k$-wise independent hash function iff for any two $k$-subsets $a_1, \ldots, a_k \subseteq [n]$ and $b_1, \ldots, b_k \subseteq [m]$,

$$\Pr_{h \sim \mathcal{H}}(h(a_1) = b_1 \land \cdots \land h(a_k) = b_k) = \frac{1}{m^k}.$$

Roughly speaking, a $k$-wise independent hash function behaves like a totally random function when considering at most $k$ elements. We use the following standard result for $k$-wise independent hash functions.

► Proposition 9 ([37]). For every integers $n, m, k \geq 2$, there is a $k$-wise independent hash function $\mathcal{H} = \{ h : [n] \to [m] \}$ so that sampling and storing a function $h \in \mathcal{H}$ takes $O(k \cdot (\log n + \log m))$ bits of space.

We shall also use the following concentration result on the extension of Chernoff-Hoeffding bounds for limited independence hash function.

► Proposition 10 ([41]). Suppose $h$ is a $k$-wise independent hash function and $X_1, \ldots, X_m$ are $m$ random variables in $\{0, 1\}$ where $X_i = 1$ iff $h(i) = 1$. Let $X := \sum_{i=1}^{m} X_i$. Then, for any $\varepsilon > 0$,

$$\Pr(|X - \mathbb{E}[X]| \geq \varepsilon \cdot \mathbb{E}[X]) \leq \exp\left(-\min\left\{ \frac{k}{2}, \frac{\varepsilon^2}{4 + 2\varepsilon} \cdot \mathbb{E}[X]\right\}\right).$$
3 New Sketching Toolkit

We present two novel linear sketches in this section that are needed for our main algorithm. The first one is a simple way of sampling random edges from a group of vertices to obtain an edge to a random neighbor of this set (as opposed to a random edge). The second (and main) linear sketch is a sparse-recovery-type sketch that allows for finding neighborhood of a vertex (or group of vertices) assuming we already know a set that intersects largely with the neighborhood. Due to space limitations, we postpone the proofs in this section to the full version of the paper.

3.1 Neighborhood-Edge Sampler

Suppose we have a group $S$ of vertices, and we want to sample a vertex $v$ from the neighborhood of $S$ ($v \in R(N(S))$). If we want the probability of each vertex $v$ to be proportional to $\deg(v)$, we can simply sample an edge incident on $S$ (using an L0-Sampler) and return the other endpoint; but what if we would like to sample $v$ uniformly at random from $N(S)$?

There is a simple (and standard) solution for this problem using an L0-Sampler if we do not need to recover the edge incident on $v$. However, for our purpose, we crucially need the edge as well therefore just an L0-Sampler will not work. We formulate the following problem to address this formally.

▶ Problem 3. Given a graph $G = (V, E)$ specified in a dynamic stream, and a set $S \subseteq V$ of vertices at the start of the stream, output an edge $(u, v)$ such that $u \in S$ and $v$ is sampled uniformly at random from $N(S)$.

The following lemma gives a linear sketch for solving this problem.

▶ Lemma 11. There is a linear sketch, called NE-Sampler($G$, $S$), for Problem 3 with size $s_{NE} = s_{NE}(n) = O(\log^3 n)$ bits, that outputs FAIL with probability at most $1/100$ and gives a wrong answer with probability at most $n^{-8}$.

3.2 Sparse-Neighborhood Recovery

The second problem we would like to tackle is a sparse-recovery-type problem: suppose we have a group $S$ of vertices, and at the end of the stream, we (somehow) managed to find a superset $T$ of all but a “tiny” fraction of vertices in $N(S)$. Can we recover the remainder of $N(S) - T$ efficiently now using our sketch? Formally,

▶ Problem 4 (Sparse-Neighborhood Recovery). Let $a, b \geq 1$ be known integers. Consider a graph $G = (V, E)$ specified in a dynamic stream and let $S \subseteq V$ be a known subset of vertices. The goal is to, given a set $T \subseteq V$ at the end of the stream, return the set $N(S) - T$, assuming the following promises:
1. size of $T$ is at most $a$;
2. size of $N(S) - T$ is at most $b$;
3. for every vertex $v \in N(S) - T$, $|S \cap N(v)| < c$.

5 The reason we consider this the most important of our sketches is that essentially all our saving of polylog($n$) factors comes from the ultra efficiency of this sketch. For the first sketch, even a somewhat loose (in terms of extra polylog($n$) factors) bound in the space suffices for our purpose.

6 Create an $n$-dimensional vector where entry $i$ denotes the number of edges incident on $v_i$ from $S$; then use an L0-Sampler to return an element from the support of this vector uniformly at random.
In words, in Problem 4, we have a set $S$ of vertices, known at the start of the stream, and we are interested in their neighbors outside a given set $T$, specified at the end of the stream. Our guarantees are roughly that $T$ is not “too large” (parameter $a$), neighborhood of $S$ outside $T$ is not “large” (parameter $b$), and each vertex outside $T$ only has “few” neighbors inside $S$ (parameter $c$). See Figure 1 for an illustration.

![Figure 1](image)

**Figure 1** The neighborhood of $S$, $N(S)$ intersects with $T$. $T$ has size at most $a$ and $N(S) - T$ has size at most $b$. Also, every vertex $v$ in $N(S) - T$ has at most $c$ neighbors in $S$.

**Lemma 12.** There is a linear sketch, called $SN$-Recovery$(G, S)$, for Problem 4 that sketch size and number of random bits, respectively,

$$s_{SN} = s_{SN}(n, a, b, c) = O(a \cdot \log c + b \cdot \log^2 n \cdot \log c) \quad \text{and} \quad s_{SN} + O(a \cdot \log n)$$

bits and outputs a wrong answer with probability at most $1 - \exp(-b/400)$ for any $^7 b \gg \log \log n$.

The important part of Lemma 12 is that the dependence on $\log n$ is only on the (much) smaller $b$-term, as opposed to the $a$-term (otherwise, this result would be immediate by Proposition 58). This saving is a key factor in the success of our algorithms in achieving asymptotically optimal bounds for the matching problem in dynamic streams.

Before we move on from this section, we also mention a simple helper sketch that allows to approximately verify if the promises of Problem 4 are satisfied for a given input (which is needed for our main algorithm). We formally define the problem as follows:

**Problem 5 (Neighborhood Size Testing).** Let $a, \tilde{b} \geq 1$ be known integers. Consider a graph $G = (V, E)$ specified in a dynamic stream and let $S \subseteq V$ be a known subset of vertices. The goal is to, given a set $T \subseteq V$ at the end of the stream, return “Yes” if $|N(S) - T| \leq \tilde{b}$ and “No” if $|N(S) - T| \geq 2\tilde{b}$, assuming that the size of $T$ is at most $a$ and that $a$ itself satisfies $a \geq 16\tilde{b}$.

The setting of Problem 5 is quite similar to Problem 4 with the difference that we know are only interested in testing whether the promise of Problem 4 is approximately satisfied or not. In other words, we have a set $S$ of vertices, known at the start of the stream, and we are interested in the size of the neighborhood outside a given set $T$, specified at the end of the stream. We guarantee that $T$ is not “too large” (parameter $a$) and that $a$ is slightly larger than $\tilde{b}$ and want to know the size of the neighborhood of $S$ outside $T$. If the size is between $\tilde{b}$ and $2\tilde{b}$ then the answer can be arbitrary.

---

7 We note that this condition is strictly speaking not needed and is only provided to simplify the algorithm.

8 What makes Problem 4 particularly different from sparse-recovery is that since we only know $T$ is a superset of $N(S)$ and not equal to it, our underlying vector is only $(a + b)$-sparse as opposed to $b$-sparse.
Lemma 13. There is a linear sketch, called \textbf{NE-Tester}(G,S), for Problem 5 that uses

\[ s_{\text{net}}(n,a,\tilde{b}) = O\left(\frac{a}{b} \log^3 n\right) \]

bits and at the end of the stream given the set \( T \), with probability at least \( 1 - n^{-3} \) correctly outputs “Yes” if size of \( N(S) - T \) is at most \( \tilde{b} \) and “No” if it is at least \( 2\tilde{b} \) (the answer may be arbitrarily otherwise).

4 Main Result and Setup

In this section, we present our main results for \( \alpha \)-approximating the maximum matching of any given graph in dynamic streams using \( O(n^2/\alpha^3) \) bits of space. Specifically, we prove the following theorem for linear sketches which immediately gives a dynamic streaming algorithm with the same guarantees by Proposition 3.

Theorem 14. There is a linear sketch that given any parameter \( \alpha \leq n^{1/2-\delta} \), for any constant \( \delta > 0 \), and any \( n \)-vertex graph \( G = (V,E) \) specified via \( \text{vec}(E) \), with high probability outputs an \( \alpha \)-approximate maximum matching of \( G \) using \( O(n^2/\alpha^3) \) bits of space.

We will make the following (more or less standard) assumptions when designing our algorithms. Both assumptions are made for simplicity of exposition and we show how to remove them later in this section.

Assumption 1 (Knowledge of matching size). At the beginning of the stream, we are given an estimate opt with the promise that the maximum matching size of the input graph \( G \) has size at least \( \text{opt} \). The goal is then to return a matching of size \( (\eta_0 \cdot \text{opt}/\alpha) \) for some absolute constant \( \eta_0 > 0 \) at the end of the stream.

Assumption 2 (Range of parameters). We assume that the parameter \( \text{opt} \) of Assumption 1 and approximation factor \( \alpha \) satisfy the following equations: \( \text{opt} \geq \alpha^2 \cdot n^\delta \) and \( \alpha > 100 \).

Remark 15. While we assume Assumption 1 and Assumption 2 when designing our algorithms, even if these assumptions are not satisfied, the algorithms (with high probability) will not output an edge that does not belong to the graph, but may output a matching that is not sufficiently large for our purpose.

The plan for designing our main algorithms is then to focus on the problem of Assumption 1 (and further assume Assumption 2). We first give a linear sketch that can handle “easy” graphs for this problem. In particular, we prove the following lemma.

Lemma 16 (Match-Or-Sparsify Lemma). There is a linear sketch that given any graph \( G = (V,E) \) specified via \( \text{vec}(E) \), uses \( O(\text{opt}^2/\alpha^3) \) bits of space and with high probability outputs a matching \( M_{\text{easy}} \) that satisfies at least one of the following conditions:

- **Match-case:** The matching \( M_{\text{easy}} \) has at least \( \eta_0 \cdot \text{opt}/\alpha \) edges;
- **Sparsify-case:** The induced subgraph of \( G \) on vertices not matched by \( M_{\text{easy}} \), denoted by \( \overline{G}_{\text{easy}} \), has at most \( (20\cdot \text{opt} \cdot \log^4 n) \) edges and a matching of size at least \( 3 \cdot \text{opt}/4 \).

This lemma should be interpreted as: we can either find a matching of size \( (\text{opt}/8\alpha) \) (solving the problem of Assumption 1 with \( \eta_0 = 1/8 \)), or certify that we had a “hard” graph to work on. Our main saving in the space then comes from the subsequent algorithm that handles any input that leads to the sparsify-case of Lemma 16. We prove Lemma 16 in Section 6.
The following lemma is the heart of the proof. We emphasize that the information provided by algorithm of Lemma 16 will only be available to the algorithm of this lemma at the end of the stream as we have to run both algorithms in parallel in a single pass.

▶ Lemma 17 (Algorithm for Sparsify-Case). There is a linear sketch that given any graph $G = (V,E)$ specified via $\text{vec}(E)$, uses $O(\text{opt}^2/\alpha^3)$ bits of space and with high probability, given the matching $M_{\text{easy}}$ of Lemma 16 in the recovery step, can recover a matching of size $(\text{opt}/8\alpha)$ in $G$.

We prove Lemma 17 in Section 5. Theorem 14 now follows easily from Lemma 16 and Lemma 17 and by lifting Assumption 1 and Assumption 2.

Proof of Theorem 14. By Lemma 16 and Lemma 17, we can use $O(\text{opt}^2/\alpha^3)$ bits of space under Assumption 1 and Assumption 2 and find a matching of size at least $(\text{opt}/8\alpha)$ in $G$ with high probability.

Removing Assumption 2. Firstly, if $\text{opt} < 2^{1/2} \cdot n^\delta$, then by the promise of Theorem 14 that $\alpha < n^{1/2-\delta}$, we get that $\text{opt} < n^{1-\delta}$. At this point, even if we run an algorithm with $O((\text{opt}^2/\alpha^3) \cdot \text{poly log (n)})$ space, it will still be $o(n^2/\alpha^3)$ bits as required by Theorem 14. Thus, we can run any of the previously-best algorithms for this problem, e.g. the ones in [7, 13], to solve the problem (one can also run our Lemma 16 with poly log (n) bits of space overhead so that it only leads to the match-case and thus solve the problem this way; we omit the details here).

Secondly, if $\alpha \leq 100$, we can simply maintain a counter mod two between every pairs of vertices to store all edges of $G$ in $O(n^2)$ bits of space which is permitted by Theorem 14 when $\alpha = O(1)$. This allows us to solve the problem exactly.

Removing Assumption 1. Let $\mathcal{A}(\text{opt}, \alpha)$ be the algorithm we obtained so far under Assumption 1. We simply run $\mathcal{A}(o, \beta)$ for all choices of $o \in \{2^i \mid i = 0 \text{ to } \log n\}$ and $\beta = (\alpha/2\eta_0)$ in parallel and return the largest matching found. By Remark 15, these matchings all belong to the input graph with high probability and for the choice of $o \geq \mu(G)/2$, where $\mu(G)$ is the maximum matching size of $G$, we can apply our results for $\text{opt} = o$ to get a matching of size $\eta_0 \cdot \text{opt} / (\beta/2) = \mu(G)/\alpha$ in the graph, which is precisely an $\alpha$-approximation as desired. Finally, the space of this new algorithm is

$$O(1) \cdot \sum_{o \in \{2^i \mid i = 1 \to \log n\}} \frac{o^2}{\alpha^3} = O(n^2/\alpha^3) \text{ bits}$$

as the sum is forming a geometric series. This concludes the proof of Theorem 14.

We conclude this section by making the following remark about the sketches we use.

▶ Remark 18. Throughout our main algorithms in the remainder of the paper, we use at most $O(n^2)$ copies of the sketching primitives $\text{NE-Tester}$, $\text{NE-Sampler}$ and $\text{SN-Recovery}$ developed in Section 3. For all these sketches the probabilities of failure and error are at most, respectively, $1/100$ and $n^{-3}$. We can simply do a union bound over all these sketches and have that with a high probability, none of them are going to err. Hence, we condition on this high-probability event here and do not explicitly account for the error probability of this part each time. However, we will consider the case that these sketches may output FAIL still.
5 Main Algorithm: Handling the Sparsify-Case

As the main part of our work in this paper is the algorithm for the sparsify-case of Lemma 16, we first present that algorithm, and postpone the proof of Lemma 16 to the next section instead.

Lemma 16 allows us to find a matching $M_{easy}$ which is either large enough, or the subgraph induced by its unmatched vertices is sparse and has a large matching. Our task now is to handle the latter case efficiently, i.e., prove Lemma 17. We emphasize that we can only know this particular sparse subgraph of the input after the pass over the input, and by that point we should have collected all the required information from the graph already.

The following lemma is a slightly weaker version of Lemma 17.

Lemma 19 (Slightly weaker version of Lemma 17). There is a linear sketch that given any graph $G = (V, E)$ specified via $\text{vec}(E)$, with high probability uses $O(\frac{\text{opt}^2}{\alpha^3})$ bits of space and given the matching $M_{easy}$ of Lemma 16 in the recovery step that satisfies the sparsify-case, can recover a matching of size $(\frac{\text{opt}}{8\alpha})$ in $G$ with probability at least $1 - \frac{n - \delta}{6}$ and does not output any edge that is not in $G$ with high probability.

Let us show that Lemma 19 immediately proves Lemma 17 in its full generality. Firstly, it is without loss of generality to assume that $M_{easy}$ satisfies the sparsify-case as otherwise, the algorithm can simply return $M_{easy}$ itself which is of size at least $(\frac{\text{opt}}{8\alpha})$ in the match-case and satisfies the promise of Lemma 17.

Secondly, to improve the success probability to a high-probability bound, we can run the algorithm of above lemma in parallel for $\frac{60}{\delta} = O(1)$ times and return the largest matching output by any copy. With high probability, the algorithm still does not output an edge not in the graph and uses $O(\frac{\text{opt}^2}{\alpha^3})$ bits of space (as $\delta = \Theta(1)$). The probability that none of these matchings are large enough is only $(n - \delta/6)^{60/}\delta = n^{-10}$; thus, the algorithm also outputs a large enough matching with high probability. This proves Lemma 17 assuming Lemma 19. As such, in this section, we focus on proving Lemma 19. To simplify the exposition, we present and analyze the sketching matrix and recovery step of the linear sketch in Lemma 19 separately.

5.1 The Sketching Matrix

The sketching matrix of Lemma 19 is computed via the following algorithm. We create $k \approx \frac{\text{opt}}{\alpha}$ groups of vertices and each group is obtained by sampling each vertex independently with probability $\frac{1}{k}$ (so vertices can belong to more than one group or none at all). We connect these groups using a fixed $(k/\alpha)$-regular graph and throughout the stream, we only focus on the edges appearing between vertices of connected groups. Over these edges then, we maintain one NE-Tester and one SN-Recovery for each group with parameters $a \approx (\frac{\text{opt}}{\alpha^2}), b \approx n^4$, and $c = \Theta(1)$. Moreover, to save space in the sketching matrices, we use the same set of random bits for sketching matrices of all SN-Recovery copies. This amounts to $O(\frac{\text{opt}^2}{\alpha^3})$ bits of space.

We note that in Algorithm 1, each vertex of $V$ may appear in more than one group of $V$ or no group at all. We start the analysis by measuring the size of the sketching matrix and the extra information stored by Algorithm 1. The proof is deferred to the full version.

Lemma 20. Algorithm 1 uses $O(\frac{\text{opt}^2}{\alpha^3})$ bits of space with high probability.

We now analyze Algorithm 1. To continue, we need some notation and definitions.
We will focus on recovering edges of the paper. We write "as the following subset of the main lemma for this subsection whose detailed proof is postponed to the full version of the paper.

\[ M := \{ (u,v) \in E \mid u \in V_i, v \in V_j, F(i,j) = 1, \forall j \} \]

For any group \( i \in [k] \), define the subgraph \( G(V_i) \) on vertices \( V \) but only consisting of edges between \( V_i \) and its neighbor-groups, i.e., with edges \( \{(u,v) \in E \mid u \in V_i, v \in V_j, F(i,j) = 1, \forall j \} \).

For every \( i \in [k] \), return the sketching matrices of \( \text{SN-Recovery}(G(V_i), V_i) \) with parameters \( a, b, c \) and \( \text{NE-Tester}(G(V_i), V_i) \) with parameters \( a \) and \( b \) as the final sketching matrix – to save space, use the same random bits for sketching matrices of all copies of \( \text{SN-Recovery} \).

**Notation.** We say an edge \( e = (u,v) \) appears between two groups \( V_i, V_j \) if \( u \in V_i \) and \( v \in V_j \), and \( V_i \) and \( V_j \) are neighbor groups, i.e., \( F(i,j) = 1 \). Similarly, we say \( e \) appears inside a group \( V_i \in V \) if there exists some group \( V_j \) such that \( e \) appears between \( (V_i, V_j) \). We write "\( e \) appears between \( (V_i, V_j) \)" or "\( e \) appears in \( V_i \)" when \( e \) appears between \( (V_i, V_j) \) or inside \( V_i \), respectively.

**Definition 21 (Group definitions).** For each group \( V_i \in V \), we say that \( V_i \) is:
- clean if it does not contain any vertex of \( M_{\text{easy}} \).
- expanding if more than \( b/2 \) edges of \( \overline{G}_{\text{easy}} \) appear inside \( V_i \) and non-expanding otherwise.

Let \( M \) be the matching of size at least \( 3 \alpha / 4 \) in \( \overline{G}_{\text{easy}} \) as guaranteed by Lemma 16 (recall that \( \overline{G}_{\text{easy}} \) is the subgraph of \( G \) induced on vertices not matched by \( M_{\text{easy}} \)). We define \( M^* \) as the following subset of \( M \) on "low-degree" vertices of \( \overline{G}_{\text{easy}} \), namely:
\[
M^* := \{ (u,v) \in M \mid \text{each of } u \text{ and } v \text{ has at most } (\tilde{b}/2) \text{ neighbors in } \overline{G}_{\text{easy}} \}.
\]

We will focus on recovering edges of \( M^* \) (which we show are sufficiently many). For this, we need to define several conditions for each edge \( (u,v) \in M^* \) that if satisfied, allows us to recover this edge via our recovery algorithm using the sketches stored by Algorithm 1.

**Definition 22 (\( M^* \)-edges definitions).** For any edge \( e \) of \( M^* \), we say that \( e \) is:
- weakly-represented by pairs of groups \( V_i \neq V_j \in V \) iff:
  1. \( e \) appears between \( V_i \) and \( V_j \) (this means \( V_i \) and \( V_j \) has to be neighbor groups),
  2. no edge of \( \overline{G}_{\text{easy}} \) other than \( e \) appear between \( V_i \) and \( V_j \), and
  3. both \( V_i \) and \( V_j \) are clean.
- strongly-represented by pairs of groups \( V_i \neq V_j \in V \) iff:
  1. \( e \) is weakly-represented by \( (V_i, V_j) \), and
  2. both of \( V_i \) and \( V_j \) are non-expanding.

Figures 2 and 3 give illustrations of this definition.

In this subsection, we show that \( \approx \alpha \) edges of \( M^* \) are strongly represented. Then, in the next subsection, we design our recovery algorithm in a way that can recover all strongly represented edges with high probability. Since these edges are coming from a matching themselves, this allows us to find a large enough matching in the input graph. We now state the main lemma for this subsection whose detailed proof is postponed to the full version of the paper.
This figure shows groups $V_i$ and $V_j$ with exactly one edge $e$ of $\overline{G}_{\text{easy}}$ between them but $e$ is not weakly-represented because $V_i$ contains a vertex of $M_{\text{easy}}$.

(a) Illustration of edges that satisfy the first condition, but not other conditions of weakly-represented edges.

This figure shows two groups $V_i$ and $V_j$ with multiple edges of $\overline{G}_{\text{easy}}$, none of which can be weakly-represented.

(b) Illustration of edges that satisfy the first condition, but not other conditions of weakly-represented edges.

This figure shows a strongly-represented edge (in blue). There is exactly one edge $e$ of $\overline{G}_{\text{easy}}$ between $V_i$ and $V_j$ both of which are non-expanding and clean. Thus, $e$ is strongly-represented.

Figure 3

Lemma 23. The number of strongly-represented edges is at least $\alpha \overline{G}_{\text{easy}} / 8\alpha$ with probability at least $1 - n^{-\delta/6}$.

5.2 The Recovery Algorithm

We now show how to recover a large matching from the sketch computed by Algorithm 1. The idea is to find all strongly-represented edges (or rather a superset of them). This is enough because the number of strongly-represented edges is at least $\alpha \overline{G}_{\text{easy}} / 8\alpha$ with high enough probability (Lemma 23) and they form a matching. To do so, we first remove all groups that have a vertex from $M_{\text{easy}}$ inside them. Next, we run a “weak tester” using sketches for $\text{NE-Tester}$ to essentially remove all expanding sets (this step is done slightly differently in the algorithm). Finally, we use $\text{SN-Recovery}$ to recover the neighborhood of each group inside $\overline{G}_{\text{easy}}$ by setting the $T$-set of the sketches in the recovery as vertices matched by $M_{\text{easy}}$. Then, whenever between two groups we only recovered a single pair of vertices, we consider this pair as an edge and store them. At the end, we compute a maximum matching among the stored edges.

Our goal now is to show that with high probability $H_{\text{rec}}$ contains all strongly-represented edges and moreover it does not contain any edge that is not part of $G$. Putting these two together with Lemma 23 then finalizes the proof.

The first step is to show that for both $\text{NE-Tester}$ and $\text{SN-Recovery}$ sketches run by the algorithm, the promise on the input is satisfied. This is done in the following series of claims.

We first prove that the set $T_i$ satisfies $|T_i| \leq a$ for all $i \in [k]$ (as required by both $\text{NE-Tester}$ and $\text{SN-Recovery}$ sketches with given parameter $a$).

Claim 24. With high probability, for every $i \in [k]$, size of $T_i$ is at most $a$.

The above claim along with $a \geq 16\overline{b}$ (by Assumption 2) is enough for running $\text{NE-Tester}$. We now show that the guarantees for $\text{SN-Recovery}$ are also satisfied. This first requires proving that $N(V_i) - T_i$, for each $V_i$ not removed has size at most $2b$ (from hereon, $N(V_i)$ is the neighborhood in the graph $G(V_i)$).
Algorithm 2 The recovery algorithm of Lemma 19.

Input: Groups \( \mathcal{V} \) and sketches computed by Algorithm 1 and a matching \( M_{\text{easy}} \) (of Lemma 16).

Output: A matching \( M_{\text{hard}} \) in \( G \).

1. For every group \( V_i \in \mathcal{V} \), define \( T_i \) as the vertices in the graph \( G(V_i) \) (defined in Algorithm 1) that also appear in \( M_{\text{easy}} \). Run the following tests:
   - \( M_{\text{easy-test}} \): if \( V_i \) has any vertex of \( M_{\text{easy}} \) inside it, remove \( V_i \);
   - \( \text{Expanding-test} \): Run the recovery algorithm of \( \text{NE-Tester}(G(V_i), V_i) \) with the set \( T = T_i \) to test if \( V_i \) has at most \( \bar{b} \) neighbors or at least \( 2\bar{b} \) neighbors out of \( T_i \) in \( G(V_i) \).

2. If \( 2\bar{b} \), remove \( V_i \).

3. For any remaining group \( V_i \), run the recovery algorithm \( \text{SN-Recovery}(G(V_i), V_i) \) with set \( T = T_i \) to recover \( NR(V_i) \) in the graph \( G(V_i) \).

4. Define the following recovered graph \( H_{\text{rec}} \) on vertices \( V \). For any two remaining groups \( V_i, V_j \), if sizes of both \( NR(V_i) \cap V_j \) and \( NR(V_j) \cap V_i \) is 1, add the edge \((u, v)\) to \( H \) where \( u \) and \( v \) are the unique vertices in the aforementioned sets. Return \( M_{\text{hard}} \) as a maximum matching of \( H \).

Claim 25. With high probability, for every remaining group \( V_i \), size of \( N(V_i) - T_i \) is at most \( 2\bar{b} \).

Finally, we also need to prove that for any vertex \( v \in N(V_i) - T_i \), size of \( V_i \cap N(v) \) is at most \( c \) (again \( N(V_i) \) in the graph \( G(V_i) \)). This is done in a rather indirect way in the following claim.

Claim 26. With high probability, for every remaining group \( V_i \) and any of its neighbor group \( V_j \), size of \( N(V_i) \cap V_j \) is at most \( c \).

Let us now argue that Claim 26 implies that for every vertex \( v \in N(V_i) - T_i \), size of \( V_i \cap N(v) \) is at most \( c \), thus satisfying the guarantee of \( \text{SN-Recovery} \). Suppose the event of Claim 26 happens for all remaining groups. This means that among the remaining groups, between every pair of neighbor groups \( V_i \) and \( V_j \), there can be at most \( c \) vertices in \( V_i \) that have an edge to \( V_j \). Naturally, any vertex in the union of all \( V_j \)'s can then also only have \( c \) neighbors in \( V_i \). Since all of \( N(V_i) - T_i \) is now a subset of these remaining \( V_j \)'s, we get the desired guarantee.

To conclude, by Claim 24, Claim 25, and Claim 26 we established that the guarantees required by \( \text{NE-Tester} \) and \( \text{SN-Recovery} \) are all satisfied.

We now show Algorithm 2 recovers all the strongly-represented edges. The first step is to show that the endpoint-groups of any strongly-represented edge will not be removed by Algorithm 2 with high probability.

Claim 27. Suppose \( e \) is a strongly-represented edge by groups \( (V_i, V_j) \). Then, with high probability, neither of \( V_i \) nor \( V_j \) will be removed by Algorithm 2.

The final step is the following lemma.

Lemma 28. With high probability, \( H_{\text{rec}} \) contains all strongly-represented edges and no edge that does not belong to \( G \).

We can now conclude the proof of Lemma 19. Firstly, by Lemma 28, with high probability the algorithm does not make an error. Moreover, by Lemma 23, with probability at least \( 1 - n^{-6/8} \), there are at least \((\text{opt}/8\alpha)\) strongly-represented edges. Since these edges are
coming from a matching \( M^* \) and by Lemma 28 we get all of those in \( H_{rec} \), the output matching \( M_{hard} \) is going to have size at least \((\opt/8\alpha)\). As we already established the bound on the space in Lemma 20, this concludes the proof.

6 Match-Or-Sparsify Lemma

We prove Lemma 16, restated below, in this section. Informally speaking, given a graph \( G \), this lemma gives an algorithm that either finds a large matching in \( G \) or identifies a sparse induced subgraph of \( G \) that contains a large matching.

▶ Lemma (Re-statement of Lemma 16). There is a linear sketch that given any graph \( G = (V,E) \) specified via \( \vec{E} \), uses \( O(\opt^2/\alpha^3) \) bits of space and with high probability outputs a matching \( M_{easy} \) that satisfies at least one of the following conditions:

- **Match-case:** The matching \( M_{easy} \) has at least \((\opt/8\alpha)\) edges;
- **Sparsify-case:** The induced subgraph of \( G \) on vertices not matched by \( M_{easy} \), denoted by \( G_{easy} \), has at most \( 20\opt \cdot \log 4n \) edges and a matching of size at least \( 3\opt/4 \).

The algorithm in Lemma 16 samples \( \approx \opt^2/(\alpha \cdot \log n) \) edges from the graph using a non-uniform distribution as follows: for each sample, we first pick \( \approx \opt/\alpha \) vertices \( S \) uniformly at random and then use \texttt{NE-Sampler} to sample an edge from \( S \) to a vertex of \( N(S) \) chosen uniformly at random. Given the bound of \( O(\log^3 n) \) bits on the size of sketches for \texttt{NE-Sampler}, the total space of the algorithm can be bounded by \( O(\opt^2/\alpha^3) \) bits. In the recovery phase then, we compute a greedy matching over these sampled edges and return it as \( M_{easy} \). Formally, the algorithm is as follows.

\begin{algorithm}
\caption{The algorithm of Match-Or-Sparsify Lemma (Lemma 16).}
\begin{description}
\item[Input:] A graph \( G = (V,E) \) specified via \( \vec{E} \); \hspace{1cm} \item[Output:] A matching \( M_{easy} \) in \( G \).
\item[Parameters:] Let \( k := \opt/\alpha \) and \( s := \opt^2/(\alpha \cdot \log n) \).
\end{description}
\item[Sketching matrix:] 1. For \( i = 1 \) to \((2s)\) steps:
   \begin{enumerate}
   \item Sample a pair-wise independent hash function \( h_i : V \to [k] \) and set \( V_i := \{v \in V \mid h_i(v) = 1\} \).
   \item Let \( \Phi(V_i) \) be the sketching matrix of \texttt{NE-Sampler}(\( G,V_i \)).
   \end{enumerate}
2. Return \( \Phi := [\Phi(V_1); \cdots; \Phi(V_{2s})] \) as the sketching matrix.
\item[Recovery algorithm:] 1. For all \( i \in [2s] \), run the recovery algorithm of \texttt{NE-Sampler}(\( G,V_i \)) using \( \Phi(V_i) \) and \( \Phi(V_i) \cdot \vec{E} \) to get an output edge \( e_i \) (we write \( e_i = \bot \) if the sampler outputs \texttt{FAIL}).
2. Let \( M_{easy} \leftarrow \emptyset \) initially. For \( i = 1 \) to \( 2s \) steps: greedily include \( e_i \) in \( M_{easy} \) whenever \( e_i \neq \bot \) and both its endpoints are unmatched by \( M_{easy} \).
\end{algorithm}

Note that it is equivalent to think of the edges being recovered one by one and fed to the greedy matching algorithm. We will use this in our analysis. The space complexity of this algorithm can be easily bounded by \( O(\opt^2/\alpha^3) \) bits of space with high probability and we omit the details here.

We now prove that the matching \( M_{easy} \) output by Algorithm 3 satisfies the guarantees of Lemma 16. To continue, we need some notation.
Notation. For any $i \in [2s]$, let $M_i$ be the set of edges included in $M_{\text{easy}}$ in the first $i - 1$ steps of the recovery, i.e., from $\{e_j\}_{j=1}^{i-1}$, and $\overline{G}_i$ to be the subgraph of $G$ induced on unmatched vertices of $M_i$. We use $\deg_i(v)$ to denote the degree of each vertex in $\overline{G}_i$ to other vertices in $\overline{G}_i$. We partition vertices of $\overline{G}_i$ based on their degrees in $\overline{G}_i$ into low-, medium-, and high-degree as follows:

$$
\text{Low}_i := \{ v : \deg_i(v) < (\alpha \log^3 n) \}, \quad \text{Med}_i := \{ v : (\alpha \log^3 n) \leq \deg_i(v) < \frac{\text{opt}}{8 \alpha} \}, \text{ and }
$$

$$
\text{High}_i := \left\{ v : \deg_i(v) \geq \frac{\text{opt}}{8 \alpha} \right\}.
$$

We define the following two events:

- $\mathcal{E}_M(i)$: the matching $M_i$ has less than $(\text{opt}/8\alpha)$ edges (i.e., matching-case not happened);
- $\mathcal{E}_S(i)$: the subgraph $\overline{G}_i$ has more than $(20 \cdot \text{opt} \cdot \log^4 n)$ edges (i.e., sparsify-case not happened).

Finally, we say that a choice of $V_i$ in step $i \in [2s]$ is clean if $V_i$ does not contain any matched vertices of $M_i$.

We start by proving that if for some $i \in [2s]$ at least one of these events do not happen, then Algorithm 3 succeeds in outputting the desired matching of Lemma 16. The proof is straightforward.

\textbf{Claim 29.} Suppose for some $i \in [2s]$, either of $\mathcal{E}_M(i)$ or $\mathcal{E}_S(i)$ does not happen; then, $M_{\text{easy}}$ of Algorithm 3 satisfies the guarantees of Lemma 16.

The goal at this point is to show that with high probability, for some $i \in [2s]$, one of the events $\mathcal{E}_M(i)$ or $\mathcal{E}_S(i)$ is not going to happen. In order to do so, we partition the steps of the algorithm into two 	extbf{batches} of size $s$ each and analyze each one separately as follows:

- 	extbf{First batch:} We first show that as long as $\mathcal{E}_M(i)$ and $\mathcal{E}_S(i)$ happen for all $i \in [s]$, with high probability, the set $\text{High}_{s+1}$ (and thus $\text{High}_j$ for all $j \in (s, 2s]$) will be empty for the second batch (a technical condition needed for our variance reduction ideas in the next part). Formally,

  \textbf{Lemma 30. With high probability, either at least one of $\mathcal{E}_M(i)$ and $\mathcal{E}_S(i)$ does not happen for some step $i \in [s]$ or $\text{High}_{s+1}$ will be empty.}

- 	extbf{Second batch:} We then show that whenever both $\mathcal{E}_M(i)$ and $\mathcal{E}_S(i)$ happen in a step $i \in (s, 2s]$, there will be a probability of $\approx k/s$ in increasing the size of $M_i$ by one in this step (this is the main part of the argument). Given that we repeat this process for $s$ steps also, this allows us to argue $M_{\text{easy}}$ will eventually become of size $\approx k = \text{opt}/\alpha$, thus satisfying the matching-case condition (or one of the events happen along the way, and we can use Claim 29 instead). Formally,

  \textbf{Lemma 31. Assuming $\text{High}_{s+1}$ is empty, with high probability, at least one of the events $\mathcal{E}_M(i)$ or $\mathcal{E}_S(i)$ does not happen for some $i \in (s : 2s]$.}

Lemma 16 then follows immediately from these two lemmas combined with Claim 29.

Before we get to the proofs of these lemmas, we make the following important remark.

\textbf{Remark 32.} The actions of Algorithm 3 are clearly not independent across different steps (in the recovery phase). However, in our upcoming probability analysis in each step $i \in [2s]$ we fix the randomness of all prior steps conditioned on that events $\mathcal{E}_M(i)$ and $\mathcal{E}_S(i)$, and use only the randomness of the choice of $(V_i, e_i)$ in this step. This randomness is independent of prior steps. As such, in the following, all our probability calculations in a step $i$ are conditioned on randomness of prior steps and events $\mathcal{E}_M(i)$ and $\mathcal{E}_S(i)$, without writing it explicitly each time. These probability calculations may not necessarily remain correct when either of these events do not happen, but we will be done by Claim 29 in those cases anyway.
6.1 First Batch: Proof of Lemma 30

Let $v$ be any vertex in $V$ and consider any step $i \in [s]$. If $\deg_i(v) < (\text{opt}/8\alpha)$, then $v$ cannot be part of $\text{High}_i$ and subsequently $\text{High}_{i+1}$ since $\overline{G}_{i+1}$ is a subgraph of $\overline{G}_i$. In the following, we consider the case where $\deg_i(v) \geq (\text{opt}/8\alpha)$ and prove that there is a non-trivial chance of “progress” (to be defined later) in each step. We first bound the probability of the following useful event for our analysis.

▷ Claim 33. In step $i \in [s]$, if $\deg_i(v) \geq \text{opt}/8\alpha$ then $\Pr_{V_i}(v \in N(V_i) \text{ and } V_i \text{ is clean}) \geq 1/16$.

Let us now condition on the choice of $V_i$ and assume the event of Claim 33 has happened. We say that this step $i$ is a matching-step if $N(V_i) > \text{opt}/2\alpha$; otherwise, we call this step a vertex-step. We argue that in a matching-step we have a constant probability of increasing the size of $M_i$ by one and in a vertex-step we have a probability $\approx \alpha/\text{opt}$ of matching the vertex $v$ and thus no longer including it in $\overline{G}_{i+1}$ and $\text{High}_{i+1}$. We formalize this in the following.

▷ Claim 34. Fix $V_i$ and suppose step $i$ is a matching-step and the event of Claim 33 has happened. Then, $\Pr_{e_i}(e_i \in M_{i+1} | V_i) \geq 1/3$.

▷ Claim 35. Fix $V_i$ and suppose step $i$ is a vertex-step and the event of Claim 33 has happened. Then, $\Pr_{e_i}(v \in V(M_{i+1}) | V_i) \geq \alpha/\text{opt}$.

We can now conclude the proof of Lemma 30 as follows. We have that at least half the steps are matching-steps or half of them are vertex-steps. We consider each case as follows.

When half the steps are matching-steps

In this case, each matching-step $i$ increases size of $M_i$ by one with probability at least $1/48$ by Claim 33, and Claim 34. Thus,

$$\mathbb{E} |M_{s+1}| \geq \left(\frac{s}{2}\right) \cdot \frac{1}{48} = \frac{\text{opt}^2}{(\alpha \cdot \log n)^3} \cdot \frac{1}{48} \gg \text{opt}/\alpha,$$

given that $\text{opt} \gg \alpha^2$ by Assumption 1. Moreover, the distribution of $M_{s+1}$ statistically dominates sum of $(s/2)$ Bernoulli random variables with mean $1/48$. As such, by the Chernoff bound (Proposition 6),

$$\Pr(M_{s+1} < (\text{opt}/8\alpha)) < \exp(-\text{opt}/\alpha) \ll 1/\text{poly}(n),$$

as $\text{opt} \gg \alpha$ by Assumption 1. This implies that $\mathcal{E}_M(s + 1)$ happens, proving Lemma 30 in this case.

When half the steps are vertex-steps

In this case, each vertex-step $i$ can independently match the vertex $v$ with probability at least $(\alpha/16\text{opt})$ by Claim 33, and Claim 35. Thus,

$$\Pr(v \in \text{High}_{s+1}) \leq (1 - \frac{\alpha}{16 \text{opt}})^{s/2} \leq \exp\left(-\frac{\alpha}{16 \text{opt}} \cdot \frac{\text{opt}^2}{2 \cdot (\alpha \cdot \log n)^3}\right) \ll 1/\text{poly}(n),$$

as $\text{opt} \gg \alpha$ by Assumption 1.
where we use $\text{opt} \geq \alpha^2 \cdot n^\delta$ by Assumption 2. Thus, with high probability $v$ will not be part of $\text{High}_{s+1}$. A union bound over all the vertices $v \in V$ then ensures that $\text{High}_{s+1}$ will be empty with high probability, thus proving Lemma 30 in this case too.

Remark: We note that the definition of matching-steps and vertex-steps are tailored to individual vertices in $V$; however, even if one vertex leads to having at least half of the steps as matching-steps, we can apply the argument of first part and conclude the proof. Thus, when applying the second part of the argument, we can assume that all vertices lead to half of the steps being vertex-steps and union bound over all of them.

6.2 Second Batch: Proof of Lemma 31

We now prove Lemma 31. In the following, we condition on the event that $\text{High}_{s+1}$ (and $\text{High}_i$ for every $i \in (s, 2s]$) is empty. Our goal is then to prove that at some step $i \in (s, 2s]$, one of the events $E_M(i)$ or $E_S(i)$ is not going to happen. The key to the proof of Lemma 31 (and Lemma 16 itself) is the following.

\begin{lemma}
For any $i \in (s, 2s)$,
\[ \Pr_{(V_i, e_i)} \left( M_{i+1} > M_i \right) \geq \frac{\alpha^2 \cdot \log^3 n}{4 \cdot \text{opt}}. \]
\end{lemma}

We first identify a simple structure in the graph $G_i$. The following claim is based on a standard low-degree orientation of the graph plus geometric grouping of degrees of vertices.

\begin{claim}
At least one of the following two conditions is true about $G_i$:
\begin{enumerate}
\item for some $d \in \left[ \alpha \cdot \log^3 n, \frac{\text{opt}}{8\alpha} \right]$, there are \( \left( \frac{\text{opt} \cdot \log^3 n}{2d} \right) \) vertices in $\text{Med}_i$ with $\deg_i(v) \geq d$;
\item for some $d \in [1, \alpha \cdot \log^3 n]$, there are \( \left( \frac{19 \cdot \text{opt} \cdot \log^3 n}{2d} \right) \) vertices in $\text{Low}_i$ with $\geq d$ neighbors in $\text{Low}_i$.
\end{enumerate}
\end{claim}

In the following, we refer to a step $i \in (s, 2s]$ as a $V_i$-step whenever case (i) of Claim 37 happens and a $N(V_i)$-step otherwise. We will show that:

\begin{itemize}
\item In a $V_i$-step, we have “enough” large degree vertices and even if we sample one of them in $V_i$ it will make the intersection of $N(V_i)$ and $G_i$ large;
\item In a $N(V_i)$-step, we have “so many” low degree vertices in $G_i$ that many of them will appear in $N(V_i)$ and thus there is a large intersection between $N(V_i)$ and $G_i$ again.
\end{itemize}

In each case, we can finalize the proof by showing that having $N(V_i)$ intersect largely with $G_i$ allows us to recover an edge $e_i$ via $\text{NE-Sampler}(G, V_i)$ that can increase size of $M_i$ with sufficiently large probability.

Case (i) of Claim 37: $V_i$-steps

Let
\[ d \in [\alpha \log^3 n, \frac{\text{opt}}{8\alpha}] \quad \text{and} \quad D \subseteq \text{Med}_i \quad \text{with} \quad |D| = \frac{\text{opt} \cdot \log^3 n}{2d} \quad (2) \]
be, respectively, the degree-parameter and corresponding set guaranteed by Case (i) of Claim 37. We now lower bound the probability that $V_i$ is both clean and samples a vertex from $D$.
Claim 38. \( \Pr_{V_i} (V_i \cap D \neq \emptyset \text{ and } V_i \text{ is clean}) \geq \frac{\alpha \cdot \log^3 n}{8d} \).

Let us now condition on the choice of \( V_i \) and assume the event of Claim 38 happens. Given that any vertex in \( D \) already has \( d \) neighbors in \( G_i \), we have that \( N(V_i) \cap \overline{G}_i \) has size at least \( d \) in this case. On the other hand, \( N(V_i) \) can have at most \( (\text{opt}/4\alpha) \) neighbors outside \( G_i \) by the bound on the total number of matched vertices by \( E_M(i) \). As the choice of \( e_i \) from \( \text{NE-Sampler}(G, V_i) \) is uniform over \( N(V_i) \), we have,

\[
\Pr_{e_i} (e_i \text{ is from } V_i \text{ to } N(V_i) \cap \overline{G}_i \mid V_i) \geq (1 - \delta_F) \frac{|N(V_i) \cap \overline{G}_i|}{|N(V_i)|} \geq \frac{(3/4) \cdot d}{(\text{opt}/4\alpha) + d} \geq 2d \cdot \alpha \cdot \text{opt}.
\]

as \( d \leq (\text{opt}/8\alpha) \) in Equation (2) and \( \delta_F < 1/4 \). Given that all of \( V_i \) is also unmatched (as \( V_i \) is clean by conditioning on the event of Claim 38), we can include \( e_i \) in \( M_{i+1} \) greedily whenever \( e_i \) is between \( V_i \) and \( N(V_i) \cap \overline{G}_i \). Consequently, combining the two events above, we have,

\[
\Pr_{(V_i, e_i)} (M_{i+1} > M_i) \geq \Pr_{V_i} (V_i \cap D \neq \emptyset \text{ and } V_i \text{ is clean}) \cdot \Pr_{e_i} (e_i \text{ is from } V_i \text{ to } N(V_i) \cap \overline{G}_i \mid V_i) \geq \frac{\alpha \log^3 n}{8d} \cdot 2d \cdot \alpha \cdot \text{opt} = \frac{\alpha^2 \log^3 n}{4 \cdot \text{opt}}.
\]

This concludes the proof of Lemma 36 in this case.

Case (ii) of Claim 37: \( N(V_i) \)-steps

Let

\[
d \in [1, \alpha \log^3 n] \quad \text{and} \quad D \subseteq \text{Low}_i \quad \text{with} \quad |D| = \frac{19 \cdot \text{opt} \cdot \log^3 n}{2d} \quad (3)
\]

be, respectively, the degree-parameter and corresponding set guaranteed by Case (ii) of Claim 37. For the rest of this analysis, we focus only on the subgraph of \( \overline{G}_i \) induced on vertices of \( \text{Low}_i \) and for each \( v \in D \), we pick exactly \( d \) (arbitrary) neighbors from \( \text{Low}_i \) and denote them by \( \text{NL}(v) \). Our goal is to show that \( N(V_i) \) and \( \overline{G}_i \) intersect largely. We will do so by counting the elements in \( D \) that have neighbors in \( V_i \). This works because \( D \subseteq \overline{G}_i \) and having neighbors in \( V_i \) means that the vertex itself is in \( N(V_i) \).

For any vertex \( v \in D \), define an indicator random variable \( X_v \in \{0, 1\} \) which is 1 iff \( \text{NL}(v) \cap V_i \neq \emptyset \) (see Figure 4a). Notice that \( X = \sum_{v \in D} X_v \) is a random variable that denotes the number of vertices in \( D \) that have a neighbor in \( \text{NL}(v) \) that belongs to \( V_i \). Note that we do not consider all neighbors of \( v \) in \( \text{Low}_i \), only the ones in \( \text{NL}(v) \); this is okay since we just need a lower bound on \( |N(V_i) \cap \overline{G}_i| \). It is easy to see that \( X \leq |N(V_i) \cap \overline{G}_i| \) since \( v \) contributes to \( |N(V_i) \cap \overline{G}_i| \) if \( X_v = 1 \) (see Figure 4b). We first bound the probability of the event \( \text{NL}(v) \cap V_i \neq \emptyset \).

Claim 39. For any \( v \in D \),

\[
(1 - o(1)) \cdot \frac{d \cdot \alpha}{\text{opt}} \leq \Pr (X_v = 1) \leq \frac{d \cdot \alpha}{\text{opt}}.
\]

By Claim 39 and the size of \( D \) in Equation (3), we have,

\[
(1 - o(1)) \cdot \frac{19}{2} \cdot \alpha \cdot \log^3 n \leq \mathbb{E} [X] \leq \frac{19}{2} \cdot \alpha \cdot \log^3 n.
\]
Our goal now is to prove that $X$ is concentrated. This requires a non-trivial proof as the variables $\{X_v\}_{v \in D}$ are correlated through their shared neighbors in $V_i$. But the fact that the subgraph induced on $\text{Low}_i$ is low-degree allows us to bound the variance of $X$ using a combinatorial argument in the following claim.

\begin{claim} \text{Claim 40.} \ Var[X] \leq \frac{1}{8} \cdot \mathbb{E}[X]^2. \end{claim}

The rest of the proof is then standard (and similar to the previous case) and is omitted here. Lemma 31 now follows easily.

References


7. Sepehr Assadi, Sanjeev Khanna, Yang Li, and Grigory Yaroslavtsev. Maximum matchings in dynamic graph streams and the simultaneous communication model. In Proceedings of...
An Asymptotically Optimal Algorithm for Matchings in Dynamic Streams


Yi Li and David P. Woodruff. A tight lower bound for high frequency moment estimation with small error. In Prasad Raghavendra, Sofya Raskhodnikova, Klaus Jansen, and José D. P. Rolim,


