Streaming Maximal Matching with Bounded Deletions

Sanjeev Khanna
School of Engineering and Applied Sciences, University of Pennsylvania, Philadelphia, PA, USA

Christian Konrad
School of Computer Science, University of Bristol, UK

Jacques Dark
Unaffiliated Researcher, London, UK

Abstract

We initiate the study of the Maximal Matching problem in bounded-deletion graph streams. In this setting, a graph G is revealed as an arbitrary sequence of edge insertions and deletions, where the number of insertions is unrestricted but the number of deletions is guaranteed to be at most K, for some given parameter K. The single-pass streaming space complexity of this problem is known to be $\Theta(n^2)$ when K is unrestricted, where n is the number of vertices of the input graph. In this work, we present new randomized and deterministic algorithms and matching lower bound results that together give a tight understanding (up to poly-log factors) of how the space complexity of Maximal Matching evolves as a function of the parameter K: The randomized space complexity of this problem is $\tilde{\Theta}(n \cdot \sqrt{K})$, while the deterministic space complexity is $\tilde{\Theta}(n \cdot K)$. We further show that if we relax the maximal matching requirement to an α-approximation to Maximum Matching, for any constant $\alpha > 2$, then the space complexity for both, deterministic and randomized algorithms, strikingly changes to $\tilde{\Theta}(n + K)$.

A key conceptual contribution of our work that underlies all our algorithmic results is the introduction of the *hierarchical maximal matching* data structure, which computes a hierarchy of L maximal matchings on the substream of edge insertions, for an integer L. This deterministic data structure allows recovering a Maximal Matching even in the presence of up to $L - 1$ edge deletions, which immediately yields an optimal deterministic algorithm with space $\tilde{O}(n \cdot K)$. To reduce the space to $\tilde{O}(n \cdot \sqrt{K})$, we compute only $\sqrt{K}$ levels of our hierarchical matching data structure and utilize a randomized linear sketch, i.e., our *matching repair data structure*, to repair any damage due to edge deletions. Using our repair data structure, we show that the level that is least affected by deletions can be repaired back to be globally maximal. The repair data structure is computed independently of the hierarchical maximal matching data structure and stores information for vertices at different scales with a gradually smaller set of vertices storing more and more information about their incident edges. The repair process then makes progress either by rematching a vertex to a previously unmatched vertex, or by strategically matching it to another matched vertex whose current mate is in a better position to find a new mate in that we have stored more information about its incident edges.

Our lower bound result for randomized algorithms is obtained by establishing a lower bound for a generalization of the well-known Augmented-Index problem in the one-way two-party communication setting that we refer to as Embedded-Augmented-Index, and then showing that an instance of Embedded-Augmented-Index reduces to computing a maximal matching in bounded-deletion streams. To obtain our lower bound for deterministic algorithms, we utilize a compression argument to show that a deterministic algorithm with space $o(n \cdot K)$ would yield a scheme to compress a suitable class of graphs below the information-theoretic threshold.

2012 ACM Subject Classification Theory of computation → Streaming models; Theory of computation → Streaming, sublinear and near linear time algorithms; Theory of computation → Graph algorithms analysis

Keywords and phrases Streaming Algorithms, Maximal Matching, Maximum Matching, Bounded-Deletion Streams

52nd International Colloquium on Automata, Languages, and Programming (ICALP 2025).
Editors: Keren Censor-Hillel, Fabrizio Grandoni, Joël Ouaknine, and Gabriele Puppis
Article No. 106; pp. 106:1–106:20

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Digital Object Identifier 10.4230/LIPIcs.ICALP.2025.106

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2502.15330

Funding *Sanjeev Khanna*: Supported in part by NSF award CCF-2402284 and AFOSR award FA9550-25-1-0107.
Christian Konrad: Supported by EPSRC New Investigator Award EP/V010611/1.
Jacques Dark: Part of the work of J.D. was done while at the University of Warwick, supported by an EMEA Microsoft Research PhD studentship and ERC grant ERC-2014-CoG 647557.

1 Introduction

In the streaming model of computation, an algorithm is tasked with computing the solution to a given problem by performing a single pass over the input data while maintaining a memory of sublinear size in the input.

Graph Streams. Streaming algorithms have been studied for more than 25 years [3], and, since early on, significant attention has been devoted to the study of graph problems in this setting [18, 15]. The dominant model is the *insertion-only* model, where the input stream consists of a sequence of edges that make up a graph. In 2012, Ahn et al. [1] initiated the study of graph problems in the *insertion-deletion* setting, where the input stream consists of a sequence of edge insertions and deletions so that previously inserted edges can be deleted again. Subsequent research has revealed that some problems are equally difficult to solve in both the insertion-only and the insertion-deletion models in that algorithms require roughly the same amount of space, e.g., Connectivity [1] and Maximum Independent Set [16], while others are significantly harder to solve in the context of deletions, e.g., Maximal Matching, Maximum Matching, and Minimum Vertex Cover [4, 14].

Handling deletions requires a very different algorithmic toolkit than when deletions are not allowed, in particular, randomization is crucial even for solving seemingly trivial problems. For example, in insertion-deletion streams, deterministic algorithms for outputting a single edge of the input graph require space $\Omega(n^2)$, where n is the number of vertices in the input graph, while this problem can be solved via ℓ_0-sampling [19, 12] and poly-logarithmic space when randomization is allowed. The predominant algorithmic technique for designing insertion-deletion streaming algorithms are *linear sketches*, which are perfectly suited to the insertion-deletion model since they can naturally handle an arbitrary (unbounded) number of deletions (and insertions). It is even known that, under some conditions, linear sketches are universal for the class of insertion-deletion streaming algorithms in that the behavior of any such algorithm can be replicated by one that solely relies on the computation of linear sketches [25, 2] (see also [20] that further investigates said conditions). It is therefore not surprising that most previous work on streaming algorithms either considers the insertion-only setting with no deletions at all, or the insertion-deletion setting that allows for an unbounded/arbitrary number of deletions.

Bounded-deletion Graph Streams. In this work, we take a more refined view and consider graph problems in *bounded-deletion streams*. In this setting, the input stream consists of an unrestricted number of edge insertions and at most K edge deletions, for some integer K that is known to the algorithm. We are interested in how the space requirements of streaming algorithms change as a function of K.

Bounded-deletion, and, similarly, bounded-length, streams have been considered at least since the work of Cormode et al. [13] in 2017 (see, e.g., [13, 9, 20, 28, 1, 26, 7, 17, 27, 11]), who exploited a bound on the input stream length for the design of a sampling-based algorithm for estimating the size of a largest matching in bounded-arboricity graphs. Besides this work, bounded-deletion graph streams appear also in Kallaugher and Price [20], who gave a graph problem for which linear sketching is exponentially harder than bounded-deletion streaming, and Chou et al. [11], who studied CSPs in this context, which in some cases can also be represented as graph problems.

The bounded-deletion setting is well-motivated from a practical perspective. In all growing data sets, deletions are naturally less frequent than insertions since otherwise the data set would not grow in size. Regarding massive graphs, in social networks, new connections/friendships between entities are established much more often than deleted, and significantly more new hyperlinks are introduced into the web graph than deleted.

The Maximal Matching Problem. In this work, we study the Maximal Matching problem through the lens of bounded-deletion streams. A *matching* in a graph is a subset of non-adjacent edges, and a *maximal matching* is one that is inclusion-wise maximal, i.e., it cannot be extended by adding an edge outside the matching to it. Specifically, our goal is to understand the function $f(n, K)$ that describes the streaming space complexity of computing a maximal matching in an n-vertex graph as a function of the bound K on the number of deletions. We will, however, also consider approximation algorithms to Maximum Matching, where the goal is to compute a matching of size at least $\alpha \cdot |M^*|$, with approximation factor $0 < \alpha \leq 1$ and M^* being a largest matching. It is well-known that a maximal matching constitutes a $\frac{1}{2}$-approximation to Maximum Matching.

The single-pass streaming space complexity of Maximal Matching is well-understood in both insertion-only and insertion-deletion streams with unrestricted deletions. In insertion-only streams, the simple Greedy matching algorithm, which greedily inserts every incoming edge into an initially empty matching if possible and constitutes the main building block of most streaming algorithms for matchings (e.g. [15, 22, 8, 5, 23, 24]), yields an $O(n \log n)$ space algorithm for Maximal Matching, and this is also tight. In insertion-deletion streams, Dark and Konrad [14] showed an $\Omega(n^2)$ space lower bound for computing a Maximal Matching, strengthening a previous lower bound of $n^{2-o(1)}$ [4] (see also [21]). In other words, there is essentially no better algorithm than storing the entire graph. The lower bound of Dark and Konrad requires $\Theta(n^2)$ deletions in the input stream. When the number of deletions is restricted to be $K \in [n^2]$, the Dark and Konrad lower bound can be restated as showing that, for any $O(1)$-approximation to Maximum Matching, $\Omega(K)$ space is necessary.

Summarizing the previous discussion, we know that $f(n, 0) = \tilde{\Theta}(n)$, and that $f(n, n^2) = \Theta(n^2)$, but for arbitrary $K \in [n^2]$, known results only tell us that $f(n, K) = \Omega(n + K)$. On the algorithmic side, even when K is just $O(n)$, no results are known for computing a maximal matching that utilize $o(n^2)$ space (i.e. do better than storing the entire graph). Our current state of knowledge, for instance, gives us the inequalities $\Omega(n) \leq f(n, n) \leq O(n^2)$, leaving a huge gap between the upper and lower bounds. This raises a natural question: Does the required space $f(n, K)$ abruptly transitions to $\Omega(n^2)$ when K is just $O(n)$, or is there an algorithm that achieves a space complexity that smoothly interpolates between the $\tilde{O}(n)$ space for insertion-only streams and $\Omega(n^2)$ space for unrestricted deletion streams? We show that, indeed, the space complexity smoothly interpolates between the two extremes.

1.1 Our Results

We resolve the space complexity of bounded-deletion streaming algorithms for Maximal Matching and show that $f(n, K) = \tilde{\Theta}(n \cdot \sqrt{K})$. We obtain our result by designing a new space-efficient algorithm for bounded deletion streams, as well as by establishing a stronger new lower bound.

▶ **Theorem 1.** *There is a single-pass randomized $\tilde{O}(n \cdot \sqrt{K})$ space streaming algorithm that, with high probability, outputs a maximal matching in any dynamic graph stream with at most K deletions.*

▶ **Theorem 2.** *Any possibly randomized single-pass streaming algorithm that outputs a maximal matching with probability at least $2/3$ in dynamic graph streams with at most K deletions requires $\Omega(n \cdot \sqrt{K})$ space.*

It is worth noting that Theorem 1 implies that, up to poly-log factors, the $\Theta(n^2)$ deletions used in the lower bound by Dark and Konrad are necessary in order to obtain an $\Omega(n^2)$ space lower bound for Maximal Matching. Together these results show that the streaming space complexity of maximal matching increases smoothly as a function of the number of deletions, and there is no abrupt phase transition. On one extreme, when the number of deletions is $O(1)$, that is, when the deletions change the graph negligibly, the space complexity of $\tilde{\Theta}(n)$ is essentially the same as the space needed to store a maximal matching in an insertion-only stream. But then as the number of deletions reaches $\Omega(n^2)$, that is, when deletions can alter almost the entire graph, the space complexity rises to $\tilde{\Theta}(n^2)$, essentially the same as storing the entire graph.

We also observe that our work is the first that establishes a complete characterization of the space complexity of streaming algorithms for a graph problem as a function of the number of edge deletions K.

We show that randomization is crucial to achieving the space complexity given in Theorem 1, as deterministic algorithms for maximal matching require $\Theta(n \cdot K)$ space.

▶ **Theorem 3.** *There is a deterministic single-pass streaming algorithm that uses $\tilde{O}(n \cdot K)$ space and outputs a maximal matching in any dynamic graph stream with at most K deletions. Moreover, any deterministic algorithm for Maximal Matching requires $\Omega(n \cdot K)$ space.*

Finally, we show that, unlike in unrestricted dynamic graph streams, in the bounded-deletion model, the space complexity of Maximal Matching behaves fundamentally differently to the space complexity of computing an $O(1)$-approximation to Maximum Matching. Let $g_c(n, K)$ be the streaming space complexity of computing a c-approximation to Maximum Matching in an n-vertex graph when the number of deletions is bounded by K. Then we know that for any constant $c > 2$, $g_c(n, 0) = \tilde{\Theta}(n) = f(n, 0)$, and that $g_c(n, n^2) = \Theta(n^2) = f(n, n^2)$. Furthermore, for arbitrary $K \in [n^2]$, we know that $g_c(n, K) = \Omega(n + K)$. We show that in a sharp contrast to the Maximal Matching problem, there is an algorithm that achieves the space complexity of $\tilde{O}(n + K)$, that is, $g_c(n, K) = \tilde{\Theta}(n + K)$, and this is achieved by a deterministic algorithm.

▶ **Theorem 4.** *For any $\epsilon > 0$, there is a deterministic single-pass streaming algorithm that uses $O((n + K/\epsilon) \cdot \log n)$ space and outputs a $(2 + \epsilon)$-approximation to Maximum Matching in any dynamic graph stream with at most K deletions.*

1.2 Techniques

Bounded-deletions Maximal Matching Algorithm. We start by motivating and explaining the main ideas underlying our main algorithmic result, namely, Theorem 1. We start by observing that in absence of any deletions, one can simply store a maximal matching to solve the problem. On the other hand, when deletions are unbounded, we can simply store $\tilde{\Theta}(n^2)$ ℓ_0-samplers to solve the problem by recreating the surviving graph. Our algorithmic approach below is based on a new data structure, called *hierarchical maximal matching*, which in conjunction with a hierarchical approach for storing ℓ_0-samplers, allows us to design an algorithm whose space complexity smoothly interpolates between these two extremes.

Let I denote the edges inserted during the stream, and let D denote the edges deleted in the stream. Observe that I and D may be multisets since an edge can be inserted, subsequently deleted, and then reinserted again, and so on. Together, these sets define the graph $G(V, E)$ revealed by the dynamic stream where $E = I \setminus D$.

Our algorithm has two phases where in the first phase, we build a *hierarchical* collection of maximal matchings using only edges in I along with a data structure $\mathbf{D}$ for *matching repair* to account for deletions. In the second phase, we recover a maximal matching by starting with the least damaged maximal matching in our hierarchical collection, and repairing it using the edges stored in the data structure $\mathbf{D}$. We now explain this approach in more detail.

Suppose that we create a sequence of hierarchical maximal matchings, say, $M_1, M_2, ..., M_L$, using only the edges inserted in the stream (that is, the multiset I), ignoring all deletions. Specifically, we start by initializing $M_1, M_2, ..., M_L$ to be $\emptyset$. Now, whenever an edge, say (x, y), is inserted, we first try adding it to matching M_1. If one of x or y is already matched, then we try adding this edge (x, y) to M_2, and continue in this manner. If we are unsuccessful all the way up to M_L, then we simply discard this edge. It is clear that this hierarchical collection can be implemented as a streaming algorithm using only $\tilde{O}(n \cdot L)$ space, since each matching can have only $O(n)$ edges. Now suppose there is an index $\ell \in [L]$ such that *none* of the edges in M_ℓ are deleted, that is, $M_\ell \cap D = \emptyset$. Then we can recover a maximal matching M in the graph $G(V, E)$ as follows. We initialize $M = M_\ell$, and then greedily add edges in $(M_1 \cup M_2 \cup ...M_{\ell-1}) \setminus D$ to M so that M is a maximal matching w.r.t. edges in $(M_1 \cup M_2 \cup \cdots \cup M_\ell) \setminus D$. We now claim that M must be maximal in $G(V, E)$. Suppose not, then there is an edge $(u, v) \in E$, such that neither u nor v are matched in M. But then the edge (u, v) must not be present in any of $(M_1 \cup M_2 \cup \cdots \cup M_\ell) \setminus D$. This means that when the edge (u, v) arrived in the stream, at least one of u or v must have been matched in M_ℓ. This now is a contradiction to our assumption, $M_\ell \cap D = \emptyset$, and hence it cannot be that both u and v are unmatched in M. Thus M is maximal with respect to $E = I \setminus D$.

Of course, the only way to ensure that there exists some index $\ell \in [L]$ such that *none* of the edges in M_ℓ are deleted is to set $L = \Omega(K)$, and doing so immediately yields our deterministic algorithm that uses space $\tilde{O}(n \cdot K)$. To obtain an $\tilde{O}(n \cdot \sqrt{K})$ space algorithm, we instead set $L = \sqrt{K}$, and observe that this ensures that there exists an index $\ell \in [L]$ such that at most $\sqrt{K}$ edges in M_ℓ are deleted. The second phase of the algorithm now starts on the task of repairing M_ℓ to be a maximal matching. Suppose an edge (u, w) is deleted from M_ℓ. We would like to see if the edges in $E \setminus (M_1 \cup M_2 \cup ...M_\ell)$ can match u and/or w again. Let us focus on vertex u. Our data structure $\mathbf{D}$ will store $\Theta(\log^3 n)$ ℓ_0-samplers for each vertex $x \in V$, with each sampler sampling uniformly at random from the incident edges on x. We open these ℓ_0-samplers for u, and if we find an edge (u, v) such that the vertex v is unmatched, we add it to M_ℓ. Otherwise, there are two possibilities: (a) the degree of u is $O(\log^2 n)$ and we have recovered all incident edges on u, or (b) the degree of u is $\Omega(\log^2 n)$, and we recover at least $\Omega(\log^2 n)$ distinct neighbors of u, all of whom are matched

in M_ℓ. In case (a), we are immediately in good shape because we have recovered all edges incident on u, and we can use them to match u at the end if one of the neighbors of vertex u remains unmatched. The more interesting situation is case (b) where on the one hand, the recovered information is not sufficient to repair u. On the other hand, we cannot rule out the possibility that u could have been matched, if only we had allocated more space in $\mathbf{D}$ to recover additional edges incident on u. We next describe how we eliminate this uncertainty.

We create a collection $V_1, V_2, ..., V_R$ of subsets of V where the set V_i is a random subset of V of size $n/\log^i n$, and $R = \Theta(\log n/\log\log n)$. For each vertex $w \in V_i$, our data structure $\mathbf{D}$ stores $\Theta(\log^{i+3} n)$ ℓ_0-samplers. It is worth noting that while we are storing more and more ℓ_0-samplers per vertex as i increases, the total space used by vertices in V_i remains $\tilde{O}(n)$ as the size of V_i shrinks proportionately. We now return to repairing vertex u that ended in case (b) above: we recovered a set $N(u)$ of $\Omega(\log^2 n)$ neighbors of u, such that each vertex in $N(u)$ is currently matched. Since every vertex is contained in V_1 with probability $1/\log n$, it follows that, with high probability, there must be a vertex $v_1 \in N(u)$ such that v_1's mate in M_ℓ, say u_1, belongs to V_1. We now add the edge (u, v_1) to M_ℓ, thereby matching vertex u, but now creating a new unmatched vertex, namely, u_1. It may seem as if we have not made any progress and the vertices u and u_1 have simply traded places. But in fact we have made *some progress*. Since $u_1 \in V_1$, compared to vertex u, our data structure $\mathbf{D}$ stores $\Theta(\log n)$ times *more information* about edges incident on u_1, better positioning us to find a mate for u_1. We now repeat the above process for vertex u_1, either successfully matching it to an unmatched vertex, or recovering all incident edges on u_1, or finding a matched vertex $v_2 \in N(u_1)$ such that v_2's mate in M_ℓ, say u_2, belongs to V_2. The repair successfully terminates if either of the first two events occurs. Otherwise, we now add the edge (u_1, v_2) to M_ℓ, thereby matching vertex u_1, but now creating a new unmatched vertex in M_ℓ, namely, u_2. We then continue this process from u_2. The repair process is guaranteed to successfully terminate when we reach V_R since for each vertex in the final set V_R, our data structure $\mathbf{D}$ stores $\Theta(n \log n)$ ℓ_0-samplers each, enough to recover their entire neighborhoods.

To summarize, using $\tilde{O}(n)$ space, the data structure $\mathbf{D}$ provides a mechanism to repair an unmatched vertex u in M_ℓ due to edge deletions. Since M_ℓ can have up to $\sqrt{K}$ edge deletions, to repair all of them, the data structure $\mathbf{D}$ independently replicates the above strategy $O(\sqrt{K})$ times to repair all deletions in M_ℓ. The overall space used by the algorithm is thus $\tilde{O}(n \cdot \sqrt{K})$ for storing the edges in the hierarchical matching, and another $\tilde{O}(n \cdot \sqrt{K})$ space for the repair data structure $\mathbf{D}$, giving us the desired space bound of $\tilde{O}(n \cdot \sqrt{K})$.

Finally, it is worth underlining that the computations of the hierarchical matching and the matching repair data structures are independent. Furthermore, the matching repair data structure is computed by a linear sketching algorithm, as it is usual in the insertion-deletion setting, and the hierarchical matching data structure is computed by a (non-linear) deterministic sketch, i.e., a Greedy algorithm, as is it typical in the insertion-only setting.

Space Lower Bound for Bounded-deletions Maximal Matching. We now explain the ideas behind our main lower bound result, established in Theorem 2. Our lower bound is best understood as an extension of the tight $\Omega(n^2/\alpha^3)$ space lower bound by Dark and Konrad [14] for one-pass insertion-deletion streaming algorithms that compute an α-approximation to Maximum Matching[1]. We will denote this lower bound as the DK20 lower bound in the following. Due to the well-known fact that any maximal matching is at least half the size of a largest matching, DK20 immediately yields an $\Omega(n^2)$ space lower bound for Maximal Matching.

[1] [6] gives an algorithm that achieves this space bound up to constant factors, see also [21, 4, 10].

In the following, we will treat the DK20 lower bound as if it was established for Maximal Matching. DK20 is proved in the one-way two-party communication setting. In this setting, the first party, denoted Alice, holds the edge set E of a bipartite graph $G = (A, B, E)$, and the second party, denoted Bob, holds edge deletions $D \subseteq E$. Alice sends a message to Bob, and, upon receipt, Bob is required to output a maximal matching in the graph $G' = (A, B, E \setminus D)$, i.e., Alice's input graph with Bob's deletions applied. Since Alice and Bob can simulate the execution of an insertion-deletion streaming algorithm for Maximal Matching on their input, by forwarding the memory state of the algorithm from Alice to Bob in form of a message, a lower bound on the size of the message used in the communication setting therefore also constitutes a lower bound on the memory required by such algorithms.

In DK20, Alice holds a bipartite random graph $G = (A, B, E)$ with $|A| = |B| = n$ so that every edge is inserted into the graph with probability $\frac{1}{2}$. Bob holds subsets $A' \subseteq A$ and $B' \subseteq B$, with $|A'| = |B'| = \frac{4}{5}n$, and inserts deletions into the vertex-induced subgraph $G[A' \cup B']$ such that, after the deletions are applied, the remaining edges in $G[A' \cup B']$ form a large matching M. It is proved that recovering a constant fraction of the edges of M – a task that a protocol for Maximal Matching necessarily must achieve – requires space $\Omega(n^2)$. On a technical level, DK20 give a sophisticated reduction to the well-known one-way two-party Augmented-Index communication problem. In Augmented-Index, Alice holds a bitstring $X \in \{0,1\}^n$, and Bob holds a index $J \in [n]$ as well as the suffix $X[J+1, n]$. Alice sends a message to Bob who is tasked with reporting the bit $X[J]$. It is well-known that solving Augmented-Index with probability bounded away from $\frac{1}{2}$ requires a message of size $\Omega(n)$. In DK20, the reduction is such that the bits X of an Augmented-Index instance correspond to edges in the random graph with the property that, with constant probability, the bit $X[J]$ corresponds to an edge in the matching M. This construction is such that the mapping between $X[J]$ and the edges in M is random and, most importantly, unknown to the underlying protocol. Hence, a protocol that reports a constant fraction of the edges of M will therefore report the edge that corresponds to bit $X[J]$ with constant probability, which then solves Augmented-Index.

Bob holds $\Theta(n^2)$ deletions in the DK20 construction. In order to decrease the number of deletions to K, we proceed as follows. In our lower bound construction, Alice holds a graph $G = G_1 \dot\cup G_2 \dot\cup \ldots \dot\cup G_s$, which is a vertex-disjoint union of s graphs $(G_i)_{1 \le i \le s}$. Each graph G_i has $\sqrt{K}$ vertices and constitutes a scaled-down version of Alice's input graph in the DK20 construction, namely a bipartite random graph with edge probability $\frac{1}{2}$. Bob holds an index $I \in \{1, \ldots, s\}$, which identifies one of the graphs G_I. Furthermore, Bob holds the counterpart to Bob's input in the DK20 construction to graph G_I, i.e., edge deletions that apply to G_I. Bob leaves all the other graphs G_j, with $j \neq I$, untouched.

We establish a direct sum argument to show that this problem requires a large message size. The key insight is that, since Alice does not know the index I, the message sent from Alice to Bob must contain sufficient information so that Bob can output a maximal matching no matter from which graph G_i Bob has deleted some edges. In other words, Alice and Bob must be able to solve s independent copies of the DK20 lower bound instance. Since each graph G_i has $\sqrt{K}$ vertices, DK20 implies that $\Omega(K)$ bits are required for solving one copy. Hence, overall, space $\Omega(K \cdot s)$ is required. Last, to make sure that the final graph has n vertices, we need to set $s = \Theta(\frac{n}{\sqrt{K}})$, which delivers the claimed $\Omega(n \cdot \sqrt{K})$ space lower bound.

To implement this approach, we first define a generalization of Augmented-Index denoted Embedded-Augmented-Index, where Alice holds s binary strings $X_1, \ldots, X_s \in \{0,1\}^t$, Bob holds two indices $I \in [s]$ and $J \in [t]$ as well as the suffix $X_I[J+1, t]$, and the objective for Bob is to output the bit $X_I[J]$. Using by now standard information-theoretic arguments, we

show that this problem requires $\Omega(t \cdot s)$ bits of communication. Next, following the proof outline of DK20, we show that a bounded-deletion streaming algorithm can be used to solve Embedded-Augmented-Index, which completes the proof.

Lower Bound for Deterministic Algorithms for Bounded-Deletion Maximal Matching. Our lower bound for deterministic algorithms works with a family of bipartite graphs $G = (A, B, E)$ that have the property that, even when any K edges $D \subseteq E$ are deleted from G, then still every maximal matching in $G - D$ matches all A-vertices. The family of graphs $\mathcal{G}_K(n)$ is obtained as follows. Given the deletion budget K, define $\mathcal{H}_K$ to be the family of bipartite graphs $H = (A, B, E)$ with $|A| = K, |B| = 3K$, and the degree of every A-vertex is $2K$. Then, $\mathcal{G}_K(n)$ is the family of graphs consisting of the disjoint union of any $n/(4K)$ graphs from $\mathcal{H}_K$.

Similar to our lower bound for randomized algorithms, we prove our lower bound in the one-way two-party communication setting. Alice holds a graph $G \in \mathcal{G}_K(n)$ as input, and Bob holds up to K edge deletions $D \subseteq E(G)$. We now claim that a protocol π for Maximal Matching that outputs a maximal matching in the graph $G - D$ allows Bob to learn $K + 1$ incident edges on every A-vertex. Once this claim is established, we finalize the proof by observing that, if the message from Alice to Bob was of size $o(n \cdot K)$, then π also constitutes an encoding of these overall $\frac{n(K+1)}{4}$ edges‘using only $o(n \cdot K)$ bits. This in turn can be used to encode the graph class $\mathcal{G}_K(n)$ with fewer bits than dictated by the information-theoretic threshold - a contradiction.

To see that Bob can learn $K + 1$ edges incident on every A-vertex in input graph $G \in \mathcal{G}_K(n)$, Bob proceeds as follows. Let $a \in A$ be any vertex and π the message received from Alice. Bob then completes the protocol without introducing any deletions, recovering a maximal matching M_1 that, as discussed above, necessarily matches the vertex a. Let e_1 be the edge incident on a in M. Next, Bob completes another run of the protocol starting with message π and feeding the edge e_1 as edge deletion into the protocol. In doing so, Bob obtains another maximal matching M_2 that necessarily matches a, thereby recovering a second edge e_2 incident on a. Repeating this process, Bob learns more and more edges incident on a, feeding these edges as deletions into the protocol in the next simulation. Overall, Bob can repeat this process K times, thereby exhausting the deletion budget, which allows him to learn $K + 1$ edges incident on a. Finally, this process can be repeated for every $a \in A$, which completes the argument.

Bounded-deletion $(2 + o(1))$-Approximate Matching Algorithm. We now briefly describe the main idea behind the algorithmic result stated in Theorem 4. Once again, similar to Theorem 1, our algorithm utilizes the hierarchical matching data structure, but with some crucial modifications.

Specifically, this time we implement a *budgeted version* of the hierarchical matching data structure on the insertions part of the stream, where instead of having a fixed number of levels, we now have an overall budget B on the total number of edges stored in the hierarchical matching data structure. The data structure maintains a lexicographic maximality property whereby an edge insertion that may cause the number of stored edges to exceed the budget B triggers removal of an edge from the *current last level* of the data structure. The number of levels in the data structure forms a *bitonic* sequence over time, in that, it may keep increasing as long as the space budget B has not been reached, but once that happens, the number of levels steadily decreases as enforcing the lexicographic property can trigger many deletions from the last levels of the hierarchical matching as the remainder of the insertion stream is processed.

An immediate consequence of this lexicographic property is that if upon termination, we have a sequence of hierarchical matchings, say, $M_1, M_2, ..., M_L$, then $M_1, M_2, ..., M_{L-1}$ must necessarily constitute a $(L-1)$-level hierarchical maximal matching data structure. We also separately store all deletions that appear in the stream. Now by setting our budget B to be $\tilde{\Theta}(n+K)$, we can show that when we apply deletions, one of the $(L-1)$ maximal matchings, say M_i, necessarily loses at most an $o(1)$-fraction of its edges. We can then extend M_i to be an "almost" maximal matching M in the overall stream, by greedily inserting surviving edges in $M_1, M_2, ..., M_{i-1}$ into it. Here the "almost" refers to the fact that unlike the maximal matching algorithm of Theorem 1, we do not need to repair the missing $o(1)$-fraction edges in M_i to ensure maximality. The final matching M is easily shown to be a $(2+o(1))$-approximate matching, giving us the desired result.

1.3 Outline

In Section 2, we give notation and the definition of bounded-deletion graph streams. We then present our bounded-deletion algorithm for Maximal Matching in Section 3, and give our matching lower bound in Section 4. Due to space restrictions, we postpone the presentation of our results on deterministic algorithms to the full version of this paper. We conclude with some directions for future research in Section 5.

2 Preliminaries

Notation. For an integer k we write $[k] := \{1, 2, \ldots, k\}$. For a graph $G = (V, E)$, we denote by $N(v)$ the neighborhood of a vertex $v \in V$.

Bounded-deletion Graph Streams. Given an integer K, in the *bounded-deletion graph stream setting*, the input stream consists of a sequence of edge insertions and at most K edge deletions, which together make up the edge set of a graph $G = (V, E)$. An edge deletion can only occur if the edge was previously inserted. Observe that it is possible that an edge is introduced, subsequently deleted, and then introduced again, and so on. We assume that every prefix of the input stream describes a simple graph, i.e., a previously inserted edge can only be inserted again if it was deleted in the meantime. Observe, however, that the substream of edge insertions I may constitute a multiset, and, when regarded as a graph, then I yields a multigraph. The same holds for the substream of edge deletions D. In the following, we will write $E = I \setminus D$ to denote the edges of the final graph described by the input stream. We will see that our algorithm considers the substream of edge insertions and deals with edge multiplicities in a rather natural way.

We also assume that the parameter K is known in advance. This is a necessary assumption since, if K was not known, any algorithm that uses space $o(n^2)$ could not produce a maximal matching in case $K = \Theta(n^2)$, as demonstrated by the Dark and Konrad lower bound [14]. As it is standard, for simplicity, we also assume that algorithms know the vertex set V of the graph described by the input stream in advance.

Our algorithm makes use of ℓ_0-sampling. Given an insertion-deletion stream that describes a vector on m-coordinates, ℓ_0-sampling refers to the process of sampling a uniform random non-zero coordinate of this vector. We will use the ℓ_0-samplers of Jowhari et al. [19]:

▶ **Theorem 5** (Jowhari et al. [19]). *There exists a ℓ_0-sampler that uses $O(\log^2 m \log(1/\delta))$ bits of space and outputs a uniform random non-zero coordinate $i \in [m]$ with probability at least $1 - \delta$.*

Observe that an edge stream of an n-vertex graph describes a $\Theta(n^2)$-dimensional vector. ℓ_0-sampling on a graph stream therefore produces a uniform random edge of the input graph with space $O(\log^2 n \cdot \log(1/\delta))$. We will use ℓ_0-sampling to sample uniform random edges incident to a given vertex $v \in V$, by feeding only edge insertions and deletions incident on v into the sampler.

3 An $\tilde{O}(n\sqrt{K})$ Space Algorithm for Maximal Matching

3.1 Description of the Algorithm

We start by briefly summarizing the main steps of our algorithm. We refer the reader to the detailed overview in Section 1.2 for further intuition underlying the steps of our algorithm.

Let $G = (V, E)$ be the underlying graph that is revealed as a sequence I of edge insertions and a sequence D of deletions, that is, $E = I \setminus D$. In the first phase, our algorithm builds a sequence of hierarchical matchings, say, $M_1, M_2, ..., M_L$, using only I, the edges inserted in the stream, ignoring all deletions, where $L = \sqrt{K}$. Since there are at most K deletions, there exists an index $\ell \in [L]$ such that at most $\sqrt{K}$ edges in M_ℓ are deleted in the set D. The second phase of the algorithm is then focused on restoring the maximality of the matching M_ℓ after the deletions are applied.

This is achieved as follows. We create a collection $V_0, V_1, V_2, \ldots, V_R$ of subsets of V where the set V_i is a random subset of V of size $n/\log^i n$, and $R = \Theta(\log n/\log\log n)$. For each vertex $w \in V_i$, our data structure $\mathbf{D}$ stores $\Theta(\log^{i+3} n)$ ℓ_0-samplers, sampling uniformly at random from the edges incident on w. Now suppose vertex u used to be matched in M_ℓ via an edge (u, v), and this edge is deleted by D, then the repair process starts by opening the ℓ_0-samplers of u. This results in one of the following three events with high probability:

- **(i)** either u can be matched to a previously unmatched vertex, or
- **(ii)** u recovers all its incident edges in G, or
- **(iii)** u can be matched to an already matched vertex v_1 whose current partner u_1 is in set V_1.

Events (i) and (ii) are clearly (successful) termination events for u's repair as we have either re-matched u, or we have all incident edges on u available for us to try to rematch u – we, however, only do this after the repair process of all vertices has terminated. In case (iii), the repair process now continues from u_1, exploiting the fact that our data structure stores more ℓ_0-samplers for each vertex in V_1. We will once again encounter one of the above three events, and in case of the third event, we will once again repair u_1 by unmatching one of its neighbors, and continue repair from a vertex, say $u_2 \in V_2$. The process is guaranteed to terminate successfully when it reaches a vertex in V_R since we store enough ℓ_0-samplers to recover the entire neighborhood of each vertex in V_R. Finally, the algorithm replicates the data structure $\Theta(\sqrt{K})$ times to ensure the repair of every vertex in M_ℓ that is affected by deletions in D.

Our main algorithm is depicted as Algorithm 2, and it uses the Hierarchical-Greedy() algorithm, which is stated as Algorithm 1, as a subroutine.

We now give some more implementation details of our main algorithm that are not covered by the pseudo-code.

In Line 9 in the listing of Algorithm 2, we apply the deletions D to the matchings of the hierarchical matching. While this is straightforward if the substream I of edge insertions does not contain any multiedges, some care needs to be taken if I does contain multiedges. We apply the deletions in order as they arrived in the stream and bottom-up, i.e., we first apply deletions to M_1, then M_2, and so on. This is to ensure that deletions are matched to the relevant insertions.

Algorithm 1 Hierarchical-Greedy(L).

Require: Input stream of edge insertions (multi-edges allowed), integer parameter $L \geq 1$

$M_1, M_2, \ldots, M_L \leftarrow \varnothing$
while stream not empty **do**
  Let e be the next edge in the stream
  Let $\ell \geq 1$ be the smallest integer such that $M_\ell \cup \{e\}$ is a matching, if no such matching exists then let $\ell = -1$
  if $\ell \neq -1$ **then**
    $M_\ell \leftarrow M_\ell \cup \{e\}$
  end if
end while
return $(M_1, M_2, \ldots, M_L)$

In Line 18 of our main algorithm, Algorithm 2, we evaluate whether the entire neighborhood of a vertex is contained in a set of ℓ_0-samplers. This condition can, for example, be checked by comparing the number of different incident edges produced by the ℓ_0-samplers to the degree of the vertex. Observe that, in insertion-deletion streams, the degree of a vertex can easily be computed by using a counter in $O(\log n)$ space.

3.2 Analysis

Before analyzing our algorithm, we point out that the description of the algorithm uses large constants C, C' with $C > C'$ that are appropriately chosen. We will see that the analysis only imposes weak constraints on C and C' and that such constants are easy to pick. We also assume that all our ℓ_0-samplers succeed. Since we run the ℓ_0-samplers with error parameter $\delta = \frac{1}{n^4}$, and there are less than n^2 such samplers, by the union bound, this is a high probability event.

The first key ingredient of our analysis is the fact that, in the absence of deletions, every matching M_ℓ, $1 \leq \ell \leq L$, produced by Hierarchical-Greedy() can easily be extended to a globally maximal matching by greedily adding edges of the matchings $M_1, \ldots, M_{\ell-1}$ to it if possible. The next lemma captures this idea and combines it with a key insight that allows us to fix edge deletions: We do not need to immediately rematch vertices incident to a deleted edge as long as we know their entire neighborhoods. This lemma is key for establishing our algorithm's correctness.

▶ **Lemma 6.** *Let $G = (V, E)$ be a graph where $E = I \setminus D$ such that I is a multiset of edge insertions and $D \subseteq I$ is a multiset of edge deletions. Let $(M_1, M_2, \ldots, M_L)$ be the output of Hierarchical-Greedy(L), i.e., a sequence of hierarchical matchings constructed using the edges in I, processed in an arbitrary order. Let $F \subseteq E$ be a subset of edges. Then, for any $\ell \in [L]$, let M be any matching such that, for every vertex v matched in M_ℓ, either (i) v is also matched in M, or (ii) the entire neighborhood of v is known, i.e., $N(v) \subseteq F$. Then M can be extended to become a maximal matching of the graph G by simply greedily adding to it edges in $F \cup ((M_1 \cup M_2 \cup \ldots M_{\ell-1}) \setminus D)$ if possible.*

Proof. We prove this lemma by contradiction. To this end, assume that there exists an edge $(u, v) \in E \setminus M$ such that $M \cup \{(u, v)\}$ is a matching.

We first argue by contradiction that $V(M_\ell) \cap \{u, v\} = \varnothing$. Indeed, suppose that this was not the case and, w.l.o.g., assume that $u \in V(M_\ell)$. By the statement of the lemma, then either u is matched in M or the entire neighborhood of u is known, i.e., $N(u) \subseteq F$, which

Algorithm 2 Bounded-deletion Streaming Algorithm for Maximal Matching.

Require: Stream of edge insertions and at most K edge deletions making up a graph $G = (V, E)$; large constants C, C' with $C > C'$ suitably chosen

$R \leftarrow \frac{\log(n)+\log(C')}{\log\log n} - 2$ {Number of vertex levels}
Let $V_0 = V$, and for $i \in [R]$, let $V_i \subseteq V$ be a random subset of size $C \cdot \frac{n}{\log^i(n)}$
Input Stream Processing: Run in parallel
    1. $(M_1, \ldots, M_{\sqrt{K}}) \leftarrow$ Hierarchical-Greedy$(\sqrt{K})$ on substream of edge insertions
    2. Store all edge deletions observed in the stream in variable D
    3. For each $i \in [R] \cup \{0\}$ and every $v \in V_i$, run $2 \cdot \sqrt{K}\log^{i+3}(n)$ ℓ_0-samplers on the substream of edges incident on v; each sampler uses failure parameter $\delta = \frac{1}{n^4}$
Notation: For any $i \in [R] \cup \{0\}$, $v \in V_i$, and $j \in [\sqrt{K}]$ denote by $N_{i,j}(v)$ the jth $\frac{1}{2\cdot\sqrt{K}}$-fraction of v's level i ℓ_0-samplers
Post-processing:
Let $(M'_1, M'_2, \ldots, M'_L)$ be the matchings $(M_1, M_2, \ldots, M_L)$ with the deletions D applied

Let ℓ be such that $|M_\ell \setminus M'_\ell| \leq \sqrt{K}$, and let $V(M_\ell \setminus M'_\ell) = \{u_1, u_2, \ldots\}$
$M \leftarrow M'_\ell$ {repair M back to a maximal matching}
for $j \leftarrow 1 \ldots |V(M_\ell \setminus M'_\ell)|$ **do** {fix u_j}
  if $u_j \in V(M)$ **then**
    continue {u_j was matched while fixing some u_b with $b < j$}
  end if
  $u \leftarrow u_j$
  for $i = 0 \ldots R$ **do** {Iterate through the vertex levels}
    if $N_{i,j}(u)$ contains entire neighborhood of u **then**
      continue {u will be dealt with in Line 28}
    else if $N_{i,j}(u)$ contains a vertex v that is not matched in M **then**
      $M \leftarrow M \cup \{(u, v)\}$
    else
      Let $v \in N_{i,j}(u)$ be such that its mate u' in M is such that $u' \in V_{i+1}$ (Lemma 7 shows that such a vertex exists w.h.p.)
      $M \leftarrow (M - (u', v)) \cup \{(u, v)\}$
      $u \leftarrow u'$
    end if
  end for
  Greedily attempt to add all edges in $M'_1, \ldots, M'_{\ell-1}$ as well as all edges recovered by the ℓ_0-samplers to M if possible
end for
return M

implies that $(u, v) \in F$. In the first case, we immediately arrive at a contradiction since the edge (u, v) could not be added to M since u is already matched. In the second case, observe that we attempted to greedily add the edges of F to M, in particular, we already attempted to add the edge (u, v) to M, which also yields a contradiction.

Assume therefore that $V(M_\ell) \cap \{u, v\} = \varnothing$. This, however, implies that, when the edge (u, v) arrived in the stream, it was not included in M_ℓ despite both endpoints being free in M_ℓ. Suppose the edge (u, v) occurs β times in the stream I. It must then be the case that *all* β copies of this edge in I must have been added to one or more of the matchings

$M_1, \ldots, M_{\ell-1}$. On the other hand, since edge $(u, v) \in E \setminus M$, it means that the numbers of times (u, v) is deleted in D is strictly less than β. So at least one copy of the edge (u, v) in $M_1, \ldots, M_{\ell-1}$ remains undeleted, after we remove edges in D. Since we attempted to add the surviving edges of $M_1, \ldots, M_{\ell-1}$ to M, we also arrive at a contradiction.

It follows then that the matching M is indeed maximal in G. ◀

The second key ingredient of our analysis is a *progress lemma* that shows that the fixing process yields the desired result. In more detail, we show that, in iteration i of the loop in Line 17, when fixing any vertex $u_j \in (V(M_\ell) \cap V_i)$ that is currently unmatched but was matched in M_ℓ, then we either (i) recover all of u_j's neighbors, (ii) we are able to match u_j directly to a yet unmatched vertex, or (iii) we can match u_j to an already matched vertex v such that v's mate u' in the current matching is contained in level $i+1$, i.e., $u' \in V_{i+1}$.

▶ **Lemma 7** (Progress Lemma). *Let M be any matching in $G = (V, E)$, where G is the final graph described by the input stream, and let $u \in V_i \setminus V(M)$ be an unmatched vertex contained in level i. Then, for any j, with high probability, at least one of the following assertions is true:*

1. *$N_{i,j}(u)$ contains the entire neighborhood of u;*
2. *$N_{i,j}(u)$ contains a vertex v such that $M \cup \{(u, v)\}$ is a matching;*
3. *$N_{i,j}(u)$ contains a vertex v that is matched in M to a vertex u' such that $u' \in V_{i+1}$.*

Proof. We first recall that $N_{i,j}(u)$ consists of $\log^{i+3}(n)$ ℓ_0-samplers.

Suppose that Items 1 and 2 are false. Since Item 1 is false, the degree of u cannot be too small since otherwise the ℓ_0-samplers would have picked up every single edge incident to u. We prove in Lemma 8 that, w.h.p., the degree of u is at least $\log^{i+2}(n)/C'$. Then, since Item 2 is false, all vertices produced by the ℓ_0-samplers in $N_{i,j}(u)$ are matched in M. Denote by U their mates in M. Then, $|U| \geq \log^{i+2}(n)/C'$. Then, the probability that none of the vertices in U are contained in V_{i+1} is:

$$\frac{\binom{n-|U|}{|V_{i+1}|}}{\binom{n}{|V_{i+1}|}} \leq \left(1 - \frac{|V_{i+1}|}{n}\right)^{|U|} \leq \exp(-|V_{i+1}| \cdot |U|/n) = \exp(-\frac{C}{C'} \log n) \leq \frac{1}{n^5} \ ,$$

where we used Lemma 9 stated in the appendix, then used the inequality $1 + x \leq \exp(x)$, and then assumed that C and C' are picked such that $\frac{C}{C'}$ is large enough. Hence, such a vertex exists with high probability, which completes the proof. ◀

▶ **Lemma 8.** *Let $v \in V_i$ be any vertex. Then, if $\deg(v) \leq \log^{i+2}(n)/C'$ then $N(v) \subseteq N_{i,j}(v)$, for every j, with high probability.*

Proof. First, observe that $N_{i,j}(v)$ consists of $\log^{i+3}(n)$ ℓ_0-samplers for each vertex $v \in V_i$. Let $u \in N(v)$ be a vertex. Then, assuming that none of the ℓ_0-samplers fail, the probability that the outcome of an ℓ_0-sampler for v is u is at least $\frac{1}{\deg(v)} \geq \frac{C'}{\log^{i+2}(n)}$. Since all samplers operate independently, the probability that none of the samplers produce u is at most:

$$\left(1 - \frac{C'}{\log^{i+2} n}\right)^{\log^{i+3} n} \leq \exp\left(-\frac{C'}{\log^{i+2} n} \cdot \log^{i+3} n\right) = \exp\left(-C' \log n\right) \leq \frac{1}{n^{10}} \ ,$$

using the inequality $1 + x \leq \exp(x)$ and by picking a sufficiently large value for C'. By a union bound over the vertices in $N(v)$, we obtain that all vertices in $N(v)$ are contained in the ℓ_0-samplers with high probability. Last, by a union bound over all values of j, the result follows. ◀

We observe that the previous lemma implies the important property that, for every vertex $v \in V_R$ with $R = \frac{\log(n)+\log(C')}{\log\log n} - 2$ as in the algorithm, i.e., a vertex contained in the last level R, the entire neighborhood of v is contained in $N_{R,j}(v)$ since $\log^{R+2}(n)/C' \geq n$.

We are now ready to prove our main result, i.e., Theorem 1.

▶ **Theorem 1.** *Algorithm 2 is a $\tilde{O}(n \cdot \sqrt{K})$ space streaming algorithm for Maximal Matching, where K is an upper bound on the number of deletions in the stream, and succeeds with high probability.*

Proof. We first address the space complexity and then establish correctness.

Space Complexity. The algorithm stores the $\sqrt{K}$ matchings $M_1, \ldots, M_{\sqrt{K}}$, which requires space $\tilde{O}(n\sqrt{K})$. In addition, the algorithm stores various ℓ_0-samplers. The number of these samplers is bounded by

$$\sum_{i=0}^{R} |V_i| \cdot 2\sqrt{K}\log^{i+3}(n) \leq \sum_{i=0}^{R} C \cdot \frac{n}{\log^i n} \cdot 2\sqrt{K}\log^{i+3}(n) = n \cdot \sqrt{K} \cdot \log^3 n \cdot R = \tilde{O}(n\sqrt{K}) \ ,$$

and, since each ℓ_0-sampler uses space $\tilde{O}(1)$, the space bound follows.

Correctness. To establish correctness, we will argue that after the loop in Line 12, which ends in Line 27, but before Line 28 is executed, the matching M together with the set of ℓ_0-samplers fulfill the premises of Lemma 6. Invoking the lemma then establishes that executing Line 28 turns M into a globally maximal matching, which completes the proof.

To argue that M fulfills the premises of Lemma 6 after Line 27 (but before Line 28), we need to ensure that, for every vertex $u \in V(M_\ell)$, either u is matched in M or the entire neighborhood of u is contained in our set of ℓ_0-samplers. In the following, we say that a vertex $u \in V(M_\ell)$ fulfills property P if it is either matched in M or its entire neighborhood is contained in our ℓ_0-samplers.

Observe that when entering the loop in Line 12, the vertices $V(M_\ell \setminus M'_\ell)$ are exactly those vertices that violate property P. We will now argue that each iteration of this for-loop fixes one of these vertices and, in particular, it does not introduce any new vertices that may violate this property. This then completes the proof.

Consider thus the iteration j that fixes vertex u_j. First, it may happen that u_j was matched when fixing a vertex $u_{j'}$, for some $j' < j$. This is checked in Line 13, and if this happens then we are done. Otherwise, we enter the loop in Line 17. The vertex u is initialized as the vertex u_j that we attempt to fix. The loop satisfies the invariant that, in iteration i, we are guaranteed that $u \in V_i$. This is clearly the case for the first iteration $i = 0$ since $V_0 = V$. To see that this holds throughout, we will argue that an iteration of the loop always establishes property P for the current vertex u. Fixing vertex u may however come at the cost of introducing a new violation of property P for a previously matched vertex u'. This vertex u', however, is then necessarily contained in the vertex set V_{i+1}. Since at the end of the loop, we set $u \leftarrow u'$, we therefore see that in the subsequent iteration $i + 1$, the vertex u is now contained in V_{i+1}, as desired.

Now, to see that u is always fixed, we invoke our progress lemma, Lemma 7, which states that either the entire neighborhood of u is contained in our ℓ_0-samplers (case 1) or u is rematched (cases 2 and 3), which implies that u now satisfies property P. Case 3 matches u to a previously matched vertex v whose mate u' becomes unmatched. However, as proved in Lemma 7, u' is then necessarily contained in V_{i+1}.

Last, to see that in the final iteration of the loop in Line 17, no new vertex that violates property P is introduced, we observe that the last level V_R is such that the entire neighborhood of every vertex in V_R is stored in the ℓ_0-samplers $N_{R,j}(u)$, for every j, which is a consequence of Lemma 8.

Thus, overall, we have established that each iteration of the main for loop fixes a vertex and does not introduce any new vertices that violate property P. Lemma 6 thus establishes that Line 28 turns M into a globally maximal matching, which establishes that the output matching is maximal and completes the proof. ◀

4 Space Lower Bound for Maximal Matching

In this section, we give our space lower bound for bounded-deletion streaming algorithms for Maximal Matching. To this end, in Subsection 4.1 we define the Embedded-Augmented-Index problem, an extension of the Augmented-Index problem, and lower-bound its communication complexity. Then, in Subsection 4.2, we show that a bounded-deletion streaming algorithm for Maximal Matching can be used to solve Embedded-Augmented-Index, which yields our main lower bound result.

4.1 The Embedded-Augmented-Index Problem

The Embedded-Augmented-Index$_{s,t}$ problem is a one-way two-party communication problem and is parametrized by two integers s and t. Alice holds s bitstrings $X := X_1, \ldots, X_s \in \{0,1\}^t$. Bob holds two indices $I \in [s]$ and $J \in [t]$ as well as the suffix $X_I[J+1, \ldots, t]$. Alice sends a single message to Bob, and, upon receipt, Bob needs to output the bit $X_I[J]$. Protocols can be randomized and need to be correct on every input with probability at least $\frac{1}{2} + \epsilon$, for some small constant $\epsilon > 0$.

Let μ denote the uniform input distribution, where all variables (X, I, J) are chosen uniformly at random from their domains.

We now prove that the communication complexity of Embedded-Augmented-Index$_{s,t}$ is $\Omega(s \cdot t)$ (proof provided in the full version of this paper).

▶ **Theorem 9.** *Every randomized communication protocol for Embedded-Augmented-Index$_{s,t}$ that is correct with probability $\frac{1}{2} + \epsilon$, for any $\epsilon > 0$, requires a message of size at least*

$$(1 - H_2(\frac{1}{2} - \epsilon)) \cdot s \cdot t \ ,$$

where $H_2(.)$ denotes the binary entropy function.

4.2 From Embedded-Augmented-Index to Maximal Matching

In this section, we will prove that every randomized bounded-deletion streaming algorithm for Maximal Matching requires space $\Omega(n\sqrt{K})$. The proof closely follows [14], adapted to Embedded-Augmented-Index as the underlying hard communication problem and somewhat simplified since we only require a lower bound for small constant factor approximations as they are provided by a Maximal Matching algorithm.

▶ **Theorem 2.** *Every bounded-deletion streaming algorithm that computes a Maximal Matching with probability at least $\frac{2}{3}$ requires space $\Omega(n\sqrt{K})$.*

Proof. We will give a reduction to Embedded-Augmented-Index$_{s,t}$. For integers s, t whose values we will determine later, let $X = X_1, \ldots, X_s \in \{0,1\}^t, I \in [s]$, and $J \in [t]$ be an Embedded-Augmented-Index$_{s,t}$ instance, and let $\mathcal{A}$ be a bounded-deletion streaming algorithm for Maximal Matching that outputs a maximal matching with probability $\frac{2}{3}$ on every instance. We assume that, if $\mathcal{A}$ fails, then it still outputs a matching, but it may not be maximal.

Alice. Given X, Alice constructs the bipartite graph $G = G_1 \dot\cup G_2 \dot\cup \ldots \dot\cup G_s$, which is the disjoint union of the graphs $(G_i)_{1 \le i \le s}$. We now describe the construction of graph G_i, for any $1 \le i \le s$. The vertex set $A_i \cup B_i$ of G_i is such that $|A_i| = |B_i| = 10 \cdot \sqrt{t}$. The edge set E_i of G_i is best described via its bipartite incidence matrix M_i. To obtain M_i, we first construct the matrix $\tilde{M}_i$, which has the same dimensions as M_i. The construction process is as follows:

1. The top-left square sub-matrix of $\tilde{M}_i$ with side length $\sqrt{t}$ is filled with the entries of X_i from left-to-right and top-to-bottom.
2. All other entries of $\tilde{M}_i$ are filled with uniform random bits obtained via public randomness.
3. Alice and Bob sample a random binary square matrix Y_i with side length $10\sqrt{t}$ such that each entry is 1 with probability $\frac{1}{2}$. They compute the entry-wise XOR between $\tilde{M}_i$ and Y_i. Let $\tilde{M}'_i$ denote the resulting matrix.
4. Alice and Bob sample random permutations $\sigma_i, \pi_i : [10\sqrt{t}] \to [10\sqrt{t}]$ from public randomness. The matrix M_i is obtained from $\tilde{M}'_i$ by permuting the rows and columns with the permutations σ_i and π_i, respectively.

Alice then runs algorithm $\mathcal{A}$ on the edges of G, presented to $\mathcal{A}$ in random order.

Bob. Recall that Bob holds the indices $I \in [s]$ and $J \in [t]$ as well as the suffix $X_I[J+1, \ldots, t]$. Bob introduces edge deletions, but only into the graph G_I. To this end, consider the matrix $\tilde{M}_I$, and let a, b be such that the entry $\tilde{M}_I[a,b]$ corresponds to the bit $X_I[J]$. Then, observe that Bob knows all entries of the submatrix $\tilde{M}'_I(a,b)$ with top-left corner at position (a,b) and bottom-right corner being the bottom-right corner of $\tilde{M}'_I$ except the top-left entry $\tilde{M}'_I(a,b)[a,b]$, either because the bits $X_I[J+1,t]$ (XORed with entries of Y_I) correspond to these entries or the entries were constructed entirely from public randomness. Denote by

$$\mathcal{I} = \{(i,j) \mid a \le i \le 10\sqrt{t}, b \le j \le 10\sqrt{t} \text{ and } i - a \ne j - b\}$$

the entries of $\tilde{M}'_I$ that constitute off-diagonal entries in the submatrix $\tilde{M}'_I(a,b)$. Then, for each index $(i,j) \in \mathcal{I}$, if $M_I[\sigma(i), \pi(j)] = 1$ then Bob feeds an edge deletion that turns this entry to 0 into the algorithm $\mathcal{A}$, i.e., $M_I[\sigma(i), \pi(j)] = 0$ after the deletion. All deletions are fed into $\mathcal{A}$ in random order.

Bob then obtains the output matching OUT produced by algorithm $\mathcal{A}$.

Analysis. We denote by $OUT_i \subseteq OUT$ the subset of edges that are contained in graph G_i.

We assume from now on that the algorithm $\mathcal{A}$ does not fail. Then, we claim that, with high probability, $|OUT_I| \ge \frac{1}{4} \cdot 9\sqrt{t} - o(\sqrt{t})$. Indeed, observe that none of the diagonal entries in $\tilde{M}'_I(a,b)$ are subsequently deleted, and the edges corresponding to the 1-entries of this diagonal form a matching. Observe that both dimensions of $\tilde{M}'_I(a,b)$ are at least $9\sqrt{t}$. By concentration bounds, with high probability, there are at least $\frac{1}{2} \cdot 9\sqrt{t} - o(\sqrt{t})$ 1-entries. The claim then follows from the observation that M_I is obtained from $\tilde{M}'_I$ by permuting the rows and columns, which are operations that preserve the matching size, and by the fact that a maximal matching is at least half the size of a maximum matching.

Hence, with probability $2/3 - o(1)$, the algorithm $\mathcal{A}$ succeeds and $|OUT_I| \ge \frac{1}{4} \cdot 9\sqrt{t} - o(\sqrt{t})$ holds. We assume that these events hold from now on.

We say that the entry $X_I[J]$ materializes as an edge in graph G_I if $\tilde{M}'_I[a,b] = 1$. Recall that $\tilde{M}'_I[a,b] = \tilde{M}_I[a,b]\ XOR\ Y_I[a,b]$, where $Y_I[a,b]$ is a uniform random bit. Hence, $X_I[J]$ materializes as an edge with probability $1/2$.

We will now argue that, if the edge corresponding to $X_I[J]$ materializes, then it is also contained in OUT_I with constant probability. Indeed, due to the symmetry of the construction that is achieved by the applications of the random permutations σ_i and π_i and the XOR computations with Y_I, each of the diagonal entries of $\tilde{M}'_I(a,b)$ with value 1 are reported in the matching OUT_I with equal probability. Hence, conditioned on the entry $X_I[J]$ materializing as an edge in G_I, the algorithm $\mathcal{A}$ succeeding, and the event $|OUT_I| \geq \frac{9}{4}\sqrt{t} - o(t)$, this edge is contained in OUT_I with probability at least

$$\frac{|OUT_I| - 2\sqrt{t}}{10\sqrt{t}} \geq \frac{\frac{9}{4}\sqrt{t} - o(\sqrt{t}) - 2\sqrt{t}}{10\sqrt{t}} \geq \frac{1}{40} - o(1)\ ,$$

since OUT_I contains at least $|OUT_I| - 2\sqrt{t}$ diagonal entries from the matrix $\tilde{M}'_I(a,b)$, and any matching in G_I is of size at most $10\sqrt{t}$.

Hence, the probability that the edge corresponding to $X_I[J]$ materializes and is reported in OUT_I is at least $\frac{1}{80} - o(1)$ conditioned on $\mathcal{A}$ succeeding and on the event $|OUT_I| \geq \frac{9}{4}\sqrt{t} - o(t)$. If the edge is not observed in OUT_I then the algorithm reports either 0 or 1 as a guess for $X_I[J]$, each with probability $\frac{1}{2}$.

Using the following definition of p:

$$p = \Pr[\mathcal{A} \text{ succeeds and } |OUT_I| \geq \frac{9}{4}\sqrt{t} - o(t) \text{ and } X_I[J] \text{ materializes as an edge and is reported in } OUT_I] \geq (\frac{2}{3} - o(1)) \cdot (\frac{1}{80} - o(1)) = \frac{1}{120} - o(1)\ ,$$

we can bound the overall success probability of this strategy as follows:

$$\Pr[X_I[J] \text{ recovered correctly}] \geq p + (1-p) \cdot \frac{1}{2} = \frac{1}{120} + \frac{119}{120} \cdot \frac{1}{2} - o(1) = \frac{1}{2} + \frac{1}{240} - o(1)\ .$$

By Theorem 9, $\mathcal{A}$ therefore requires space $\Omega(s \cdot t)$. Lastly, to ensure that at most K deletions are introduced, we set $t = K$, and to ensure that the input graph has n vertices, we set $s = \frac{n}{2 \cdot 10\sqrt{t}}$. The lower bound thus gives $\Omega(s \cdot t) = \Omega(\frac{n}{\sqrt{K}} \cdot K) = \Omega(n \cdot \sqrt{K})$, as desired. ◀

5 Conclusion

In this work, we initiated the study of Maximal Matching in bounded-deletion streams. We settled the space complexity of both randomized and deterministic algorithms up to poly log factors by given algorithms as well as lower bounds, where the randomized space complexity is $\tilde{\Theta}(n \cdot \sqrt{K})$, and the deterministic space complexity is $\tilde{\Theta}(n \cdot K)$. Our results constitute the first trade-off results between space complexity and the number of deletions for a fundamental graph problem.

We hope that our work will spark work on other fundamental graph problems in the bounded-deletion stream setting, especially when the presence of deletions seems to significantly increase the space complexity of the streaming algorithm. Two natural and closely related problems for future investigation are approximate Maximum Matching and approximate Minimum Vertex Cover, where a refined understanding of space complexity in terms

of the deletion parameter K will reveal how the space complexity changes as we gradually move from insertion-only to unrestricted deletions streams. Another problem that will be interesting to study through the lens of bounded deletions is the problem of building low stretch graph spanners. Similar to Maximum Matching and Minimum Vertex Cover, there is a polynomial-factor separation in the space needed to build a low stretch spanner in an insertion-only stream, and in dynamic streams with unrestricted deletions.

Finally, we believe that our algorithmic approach of building a complementary pair of sketches, one that is non-linear, focusing on the insertions, and another one that is non-uniform and linear, focusing on the dynamic component of the stream, will also prove useful for other graph problems.

References

1 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. 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 459–467. SIAM, 2012. doi:10.1137/1.9781611973099.40.

2 Yuqing Ai, Wei Hu, Yi Li, and David P. Woodruff. New characterizations in turnstile streams with applications. In Ran Raz, editor, *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*, volume 50 of *LIPIcs*, pages 20:1–20:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICS.CCC.2016.20.

3 Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In Gary L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 20–29. ACM, 1996. doi:10.1145/237814.237823.

4 Sepehr Assadi, Sanjeev Khanna, Yang Li, and Grigory Yaroslavtsev. Maximum matchings in dynamic graph streams and the simultaneous communication model. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1345–1364. SIAM, 2016. doi:10.1137/1.9781611974331.CH93.

5 Sepehr Assadi, S. Cliff Liu, and Robert E. Tarjan. An auction algorithm for bipartite matching in streaming and massively parallel computation models. In Hung Viet Le and Valerie King, editors, *4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021*, pages 165–171. SIAM, 2021. doi:10.1137/1.9781611976496.18.

6 Sepehr Assadi and Vihan Shah. An asymptotically optimal algorithm for maximum matching in dynamic streams. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPIcs*, pages 9:1–9:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ITCS.2022.9.

7 Omri Ben-Eliezer, Rajesh Jayaram, David P. Woodruff, and Eylon Yogev. A framework for adversarially robust streaming algorithms. *J. ACM*, 69(2):17:1–17:33, 2022. doi:10.1145/3498334.

8 Lidiya Khalidah binti Khalil and Christian Konrad. Constructing large matchings via query access to a maximal matching oracle. In Nitin Saxena and Sunil Simon, editors, *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2020, December 14-18, 2020, BITS Pilani, K K Birla Goa Campus, Goa, India (Virtual Conference)*, volume 182 of *LIPIcs*, pages 26:1–26:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.FSTTCS.2020.26.

9 Mark Braverman, Sumegha Garg, and David P. Woodruff. The coin problem with applications to data streams. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 318–329. IEEE, 2020. doi:10.1109/FOCS46700.2020.00038.

10 Rajesh Chitnis, Graham Cormode, Hossein Esfandiari, MohammadTaghi Hajiaghayi, Andrew McGregor, Morteza Monemizadeh, and Sofya Vorotnikova. Kernelization via sampling with applications to finding matchings and related problems in dynamic graph streams. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1326–1344. SIAM, 2016. doi:10.1137/1.9781611974331.CH92.

11 Chi-Ning Chou, Alexander Golovnev, Madhu Sudan, and Santhoshini Velusamy. Sketching approximability of all finite csps. *J. ACM*, 71(2):15:1–15:74, 2024. doi:10.1145/3649435.

12 Graham Cormode and Donatella Firmani. A unifying framework for ℓ_0-sampling algorithms. *Distributed Parallel Databases*, 32(3):315–335, 2014. doi:10.1007/S10619-013-7131-9.

13 Graham Cormode, Hossein Jowhari, Morteza Monemizadeh, and S. Muthukrishnan. The sparse awakens: Streaming algorithms for matching size estimation in sparse graphs. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, volume 87 of *LIPIcs*, pages 29:1–29:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICS.ESA.2017.29.

14 Jacques Dark and Christian Konrad. Optimal lower bounds for matching and vertex cover in dynamic graph streams. In Shubhangi Saraf, editor, *35th Computational Complexity Conference, CCC 2020, July 28-31, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 169 of *LIPIcs*, pages 30:1–30:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.CCC.2020.30.

15 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. In Josep Díaz, Juhani Karhumäki, Arto Lepistö, and Donald Sannella, editors, *Automata, Languages and Programming: 31st International Colloquium, ICALP 2004, Turku, Finland, July 12-16, 2004. Proceedings*, volume 3142 of *Lecture Notes in Computer Science*, pages 531–543. Springer, 2004. doi:10.1007/978-3-540-27836-8_46.

16 Magnús M. Halldórsson, Xiaoming Sun, Mario Szegedy, and Chengu Wang. Streaming and communication complexity of clique approximation. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part I*, volume 7391 of *Lecture Notes in Computer Science*, pages 449–460. Springer, 2012. doi:10.1007/978-3-642-31594-7_38.

17 Avinatan Hassidim, Haim Kaplan, Yishay Mansour, Yossi Matias, and Uri Stemmer. Adversarially robust streaming algorithms via differential privacy. *J. ACM*, 69(6):42:1–42:14, 2022. doi:10.1145/3556972.

18 Monika Rauch Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. Computing on data streams. In James M. Abello and Jeffrey Scott Vitter, editors, *External Memory Algorithms, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, May 20-22, 1998*, volume 50 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 107–118. DIMACS/AMS, 1998. doi:10.1090/DIMACS/050/05.

19 Hossein Jowhari, Mert Saglam, and Gábor Tardos. Tight bounds for lp samplers, finding duplicates in streams, and related problems. In Maurizio Lenzerini and Thomas Schwentick, editors, *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2011, June 12-16, 2011, Athens, Greece*, pages 49–58. ACM, 2011. doi:10.1145/1989284.1989289.

20 John Kallaugher and Eric Price. Separations and equivalences between turnstile streaming and linear sketching. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1223–1236. ACM, 2020. doi:10.1145/3357713.3384278.

21 Christian Konrad. Maximum matching in turnstile streams. In Nikhil Bansal and Irene Finocchi, editors, *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece,*

September 14-16, 2015, Proceedings, volume 9294 of *Lecture Notes in Computer Science*, pages 840–852. Springer, 2015. doi:10.1007/978-3-662-48350-3_70.

22 Christian Konrad. A simple augmentation method for matchings with applications to streaming algorithms. In Igor Potapov, Paul G. Spirakis, and James Worrell, editors, *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018, August 27-31, 2018, Liverpool, UK*, volume 117 of *LIPIcs*, pages 74:1–74:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.MFCS.2018.74.

23 Christian Konrad and Kheeran K. Naidu. On two-pass streaming algorithms for maximum bipartite matching. In Mary Wootters and Laura Sanità, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2021, August 16-18, 2021, University of Washington, Seattle, Washington, USA (Virtual Conference)*, volume 207 of *LIPIcs*, pages 19:1–19:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.APPROX/RANDOM.2021.19.

24 Christian Konrad, Kheeran K. Naidu, and Arun Steward. Maximum matching via maximal matching queries. In Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté, editors, *40th International Symposium on Theoretical Aspects of Computer Science, STACS 2023, March 7-9, 2023, Hamburg, Germany*, volume 254 of *LIPIcs*, pages 41:1–41:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.STACS.2023.41.

25 Yi Li, Huy L. Nguyen, and David P. Woodruff. Turnstile streaming algorithms might as well be linear sketches. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 174–183. ACM, 2014. doi:10.1145/2591796.2591812.

26 Fuheng Zhao, Divy Agrawal, Amr El Abbadi, and Ahmed Metwally. Spacesaving± an optimal algorithm for frequency estimation and frequent items in the bounded deletion model. *Proc. VLDB Endow.*, 15(6):1215–1227, 2022. doi:10.14778/3514061.3514068.

27 Fuheng Zhao, Divyakant Agrawal, Amr El Abbadi, Claire Mathieu, Ahmed Metwally, and Michel de Rougemont. A detailed analysis of the spacesaving± family of algorithms with bounded deletions. *CoRR*, abs/2309.12623, 2023. doi:10.48550/arXiv.2309.12623.

28 Fuheng Zhao, Sujaya Maiyya, Ryan Weiner, Divy Agrawal, and Amr El Abbadi. Kll±: Approximate quantile sketches over dynamic datasets. *Proc. VLDB Endow.*, 14(7):1215–1227, 2021. doi:10.14778/3450980.3450990.

A Technical Lemma

▶ **Lemma 9.** *Let a, b, n be integers such that $a < n$ and $b \leq n - a$. Then, the following inequality holds:*

$$\frac{\binom{n-a}{b}}{\binom{n}{b}} \leq \left(1 - \frac{b}{n}\right)^a .$$

Proof. We compute as follows:

$$\frac{\binom{n-a}{b}}{\binom{n}{b}} = \frac{(n-a)! \cdot (n-b)!}{(n-a-b)! \cdot n!} = \frac{(n-b) \cdot (n-b-1) \cdot \ldots \cdot (n-a-b+1)}{n \cdot (n-1) \cdot \ldots \cdot (n-a+1)}$$
$$\leq \left(\frac{n-b}{n}\right)^a = \left(1 - \frac{b}{n}\right)^a ,$$

where we used the inequality $\frac{n-b-j}{n-j} \leq \frac{n-b}{n}$, for every $0 \leq j \leq a$. ◀