Streaming Algorithms for Graph k-Matching with Optimal or Near-Optimal Update Time

Jianer Chen
Department of Computer Science & Engineering, Texas A&M University, College Station, TX, USA

Qin Huang
Department of Computer Science & Engineering, Texas A&M University, College Station, TX, USA

Iyad Kanj
School of Computing, DePaul University, Chicago, IL, USA

Qian Li
Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China

Ge Xia
Department of Computer Science, Lafayette College, Easton, PA, USA

Abstract

We present streaming algorithms for the graph k-matching problem in both the insert-only and dynamic models. Our algorithms, while keeping the space complexity matching the best known upper bound, have optimal or near-optimal update time, significantly improving on previous results. More specifically, for the insert-only streaming model, we present a one-pass randomized algorithm that runs in optimal $\mathcal{O}(k^2)$ space and has optimal $\mathcal{O}(1)$ update time, and that, *w.h.p.* (with high probability), computes a maximum weighted k-matching of a weighted graph. Previously, the best upper bound on the update time was $\mathcal{O}(\log k)$, which was achieved by a deterministic streaming algorithm that however only works for unweighted graphs [16]. For the dynamic streaming model, we present a one-pass randomized algorithm that, *w.h.p.*, computes a maximum weighted k-matching of a weighted graph in $\tilde{O}(Wk^2)$ space[1] and with $\tilde{O}(1)$ update time, where W is the number of distinct edge weights. Again the update time of our algorithm improves the previous best upper bound $\tilde{O}(k^2)$ [7]. Moreover, we prove that in the dynamic streaming model, any randomized streaming algorithm for the problem requires $k^2 \cdot \Omega(W(\log W + 1))$ bits of space. Hence, both the space and update-time complexities achieved by our algorithm in the dynamic model are near-optimal. A streaming approximation algorithm for k-matching is also presented, whose space complexity matches the best known upper bound with a significantly improved update time.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases streaming algorithms, matching, parameterized algorithms, lower bounds

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2021.48

1 Introduction

Streaming algorithms for graph matching have been studied extensively, in which most of the work has been focused on approximating a maximum matching. A graph *stream* $\mathcal{S}$ for an underlying graph G is a sequence of edge operations. In the *insert-only* streaming model, each operation is an edge-insertion, while in the *dynamic* streaming model each operation is either an edge-insertion or an edge-deletion (with a specified weight if G is weighted). The majority of the work on streaming algorithms for graph matching has been on the (simpler) insert-only model. More recently, streaming algorithms for graph k-matching

[1] The notation $\tilde{O}()$ hides a poly-logarithmic factor in the input size.

(i.e., constructing a matching of k edges in an unweighted graph or a maximum weighted matching of k edges in a weighted graph), in both insert-only and dynamic models, have drawn increasing interests [7, 8, 9, 16].

The performance of streaming algorithms is measured by the limited memory (*space*) and the limited processing time per item (*update time*). For the space complexity, a lower bound $\Omega(k^2)$ has been known for the graph k-matching problem on unweighted graphs for randomized streaming algorithms, even in the simpler insert-only model [7]. Nearly space-optimal streaming algorithms for graph k-matching have been developed [7].

The current paper will be focused on the update time of streaming algorithms for graph k-matching. While there has been much work on space complexity of streaming algorithms for graph matching, much less is known regarding the update time complexity of the problem. Note that the update time sometimes could be even more important than the space complexity [25], since the data stream can come at a very high rate. If the update processing does not catch the updating rate, the whole system may fail (see, e.g., [3, 34]).

We start with the insert-only model. We present a one-pass randomized streaming algorithm that constructs a maximum weighted k-matching in a weighted graph. Our algorithm runs in $\mathcal{O}(k^2)$ space and has $\mathcal{O}(1)$ update time, both are optimal. Our techniques rely on partitioning the graph (using hashing), and defining an auxiliary graph whose vertices are the different parts of the partition. The auxiliary graph is updated during the stream. By querying this auxiliary graph, the algorithm can compute a "compact" subgraph of size $\mathcal{O}(k^2)$ that, *w.h.p.*, contains the edges of the desired k-matching. A maximum weighted k-matching can then be extracted from this compact subgraph.

Previously, Fafianie and Kratsch [16] studied kernelization streaming algorithms in the insert-only model. Their result implies a one-pass deterministic streaming algorithm for k-matching on unweighted graphs that uses $\mathcal{O}(k^2)$ space and $\mathcal{O}(\log k)$ update time. In comparison, our algorithm achieves the same space complexity but has optimal update time $\mathcal{O}(1)$. While improving the update time from $\mathcal{O}(\log k)$ on the deterministic algorithm to $\mathcal{O}(1)$ on randomized algorithms for unweighted graphs may not look surprising, our streaming algorithm with optimal space and update time for weighted graphs is a significant advance.

We then study steaming algorithms for graph k-matching in the dynamic model. We give a one-pass randomized streaming algorithm that, for a weighted graph G containing a k-matching, constructs a maximum weighted k-matching of G with probability at least $1 - \frac{11}{20k^3 \ln(2k)}$, and in case G does not contain a k-matching, reports correctly. The algorithm runs in $\tilde{O}(Wk^2)$ space and has $\tilde{O}(1)$ update time, where W is the number of distinct weights in the graph. This result directly implies a one-pass randomized streaming algorithm for unweighted k-matching running in $\tilde{O}(k^2)$ space with $\tilde{O}(1)$ update time.

In order to achieve the faster update time, we prove a structural result that can be useful in its own right for k-subset problems. Intuitively, the result states that, for any k-subset $S \subseteq U$, *w.h.p.* we can compute k subsets $T_1, \ldots, T_k$ of U that interact "nicely" with S. More specifically, (1) the sets T_i, for $i \in [k]$, are pairwise disjoint, (2) S is contained in their union $\bigcup_{i \in [k]} T_i$, and (3) each T_i contains exactly one element of S. We then apply the above result to obtain the sets T_i of vertices that *w.h.p.* induce the edges of the desired k-matching. Afterwards, we use ℓ_0-sampling to select a smaller subset of edges induced by the vertices of the T_i's that *w.h.p.* contains the desired k-matching. From this smaller subset of edges, a maximum weighted k-matching can be extracted.

Employing lower bounds for communication complexity protocols, we prove that, modulo a poly-logarithmic function of the input size, the space complexity $\tilde{O}(Wk^2)$ achieved by our algorithm is optimal with respect to both k and W. More specifically, neither the linear

term W nor the quadratic function k^2 in the space complexity of our algorithm for weighted k-matching can be improved/reduced (by more than a poly-logarithmic function). Given that our algorithms have $\tilde{O}(1)$ update time, this implies that our algorithms are essentially near-optimal in terms of both space and update time complexities.

Chitnis et al. [7] proposed a streaming algorithm on the dynamic model for maximum matching. Under the promise that the cardinality of the maximum matching is not larger than k during the entire graph stream, their algorithm runs in space $\tilde{O}(k^2)$ and has update time $\tilde{O}(1)$ for unweighted graphs. The assumption that the cardinality of the maximum matching is at most k during the entire graph stream is essential for their techniques to work since it is used to upper bound the number of vertices of degree larger than or equal to $10k$ by $\mathcal{O}(k)$, and the number of edges whose both endpoints have degree bounded by $10k$ by $\mathcal{O}(k^2)$. They also developed a streaming algorithm that approximates the maximum matching for unweighted graphs. These two algorithms can be combined to construct a k-matching with update time $\tilde{O}(k^2)$ and space $\tilde{O}(k^2)$ for unweighted graphs. The algorithm for unweighted graphs can be extended to construct a maximum weighted k-matching for weighted graphs, which runs in space $\tilde{O}(Wk^2)$ with update time $\tilde{O}(k^2)$. In comparison, our algorithm keeps the space complexity $\tilde{O}(Wk^2)$ while has significantly improved update time $\tilde{O}(1)$.

A byproduct of our result is a one-pass streaming approximation algorithm that, for any $\epsilon > 0$, *w.h.p.* computes a k-matching that is within a factor of $1+\epsilon$ from a maximum weighted k-matching in G. The algorithm runs in $\tilde{O}(k^2\epsilon^{-1}\log W')$ space and has $\tilde{O}(1)$ update time, where W' is the ratio of the maximum edge-weight to the minimum edge-weight in G. This result improves the update time complexity over the approximation result in [7], which has the same space complexity but has update time $\tilde{O}(k^2)$.

We observe that most work on weighted graph streams, including our current paper, assumes that the weight of an edge remains the same during the stream (see, e.g., [1, 2, 7, 20, 23]). To justify this assumption, we present an interesting lower bound result showing that, if this assumption is lifted, then the space complexity of the k-matching problem is at least linear in the size of the graph, and hence, can be much larger than the desirable space complexity for streaming algorithms.

The paper is organized as follows. Section 2 provides necessary definitions and a brief review on the related research. Improved streaming algorithms for graph k-matching in insert-only model and in dynamic model are presented and discussed in sections 3-5. Some lower bound results are give in section 6. Section 7 concludes with remarks.

2 Preliminaries

We refer to the following books for more detailed definitions [14, 15, 30]. We use "*u.a.r.*" as an abbreviation for "uniformly at random". For an integer i, let $[i]^-$ denote the set $\{0, 1, \ldots, i-1\}$, $[i]$ the set $\{1, \ldots, i\}$, and $\llcorner i \lrcorner$ the binary representation of i.

Computational Model and Problem Definition. In a parameterized graph streaming problem Q, we are given an instance of the form $(\mathcal{S}, k)$, where $\mathcal{S}$ is graph stream of some underlying graph G and $k \in \mathbb{N}$, and we are asked to compute a solution for $(\mathcal{S}, k)$ [9]. A *k-matching* in a graph G is a matching of k edges in G. We study the following problems:

- p-Matching: Given a graph stream $\mathcal{S}$ of an unweighted graph G and a parameter k, compute a k-matching in G or report that no k-matching exists.
- p-WT-Matching: Given a graph stream $\mathcal{S}$ of a weighted graph G and a parameter k, compute a k-matching of maximum weight in G or report that no k-matching exists.

We will assume that $V(G) = [n]^-$, and that the length of $\mathcal{S}$ is polynomial in n. We will design parameterized streaming algorithms for the above problems. Our algorithms first sample a subgraph G' of the underlying graph G in the stream such that *w.h.p.* G' contains a desired k-matching of G if and only if G has one. In the case where the size of G' is a function of k, such algorithms are referred to as *kernelization streaming algorithms* [7]. We note that result in [7] also computes a subgraph containing the edges of the desired matching, without computing the matching itself, as there are efficient algorithms for extracting the desired matching from that subgraph [18].

ℓ_0-Sampler. Let $\mathcal{S} = (i_1, \Delta_1), \ldots, (i_p, \Delta_p), \ldots$ be a stream of updates of an underlying vector $\mathbf{x} \in \mathbb{R}^n$, where $i_j \in [n]$ and $\Delta_j \in \mathbb{R}$. The j-th update (i_j, Δ_j) updates the i_j-th coordinate of $\mathbf{x}$ by setting $\mathbf{x}_{i_j} = \mathbf{x}_{i_j} + \Delta_j$. Fix a parameter $0 < \delta < 1$. An *ℓ_0-sampler* for $\mathbf{x} \neq 0$ either fails with probability at most δ, or conditioned on not failing, for any non-zero coordinate $\mathbf{x}_j$ of $\mathbf{x}$, returns the pair $(j, \mathbf{x}_j)$ with probability $\frac{1}{||\mathbf{x}||_0}$, where $||\mathbf{x}||_0$ is the ℓ_0-norm of $\mathbf{x}$, which is the same as the number of non-zero coordinates of $\mathbf{x}$. (We refer to [12].)

▶ **Lemma 1** (Follows from Theorem 2.1 in [7]). *Let $0 < \delta < 1$ be a parameter. There exists an ℓ_0-sampler algorithm that, given a dynamic graph stream, either returns FAIL with probability at most δ, or returns an edge chosen* u.a.r. *from the edges of the stream that have been inserted and not deleted. This algorithm can be implemented using $\mathcal{O}(\log^2 n \cdot \log(\delta^{-1}))$ bits of space and $\tilde{O}(1)$ update time, where n is the number of vertices in the underlying graph.*

We give a brief review on the known work that is related to our current paper.

Most work on graph matching in the streaming model has focused on approximating a maximum matching (e.g., [4, 5, 19, 21, 22, 24, 26, 31]), with the majority of the work pertaining to the (simpler) insert-only model. The most relevant to ours are [7, 8, 9, 16], which studied parameterized streaming algorithms for the maximum matching problem.

Under the promise that the cardinality of the maximum matching at *every* instant of the stream is at most k, the authors of [8, 9] presented a one-pass dynamic streaming algorithm that *w.h.p.* computes a maximum matching in an unweighted graph stream. The algorithms given in [8, 9] run in $\tilde{O}(k^2)$ space and the algorithm in [9] has $\tilde{O}(k^2)$ update time.

The authors of [7] considered the problem of computing maximum matchings in the dynamic streaming model. For an unweighted graph G, under the promise that the cardinality of the maximum matching at every instant of the stream is at most k, a sketch-based algorithm is presented, which *w.h.p.* computes a maximum matching of G, runs in $\tilde{O}(k^2)$ space, and has $\tilde{O}(1)$ update time. They proved an $\Omega(k^2)$ lower bound on the space complexity of any randomized algorithm for the parameterized maximum matching problem, even in the insert-only model, thus showing that the space complexity of their algorithm is optimal (modulo a poly-logarithmic factor). The algorithm for unweighted graphs has been extended to weighted graphs: under the same promise, there is an algorithm for computing a maximum weighted matching that runs in space $\tilde{O}(k^2 W)$ and has $\tilde{O}(1)$ update time, where W is the number of distinct edge weights. For unweighted graphs with larger matchings, an approximation algorithm is proposed [7]. Specifically, if the graph contains matchings of size larger than k, then for any $1 \leq \alpha \leq \sqrt{k}$ and $0 < \epsilon \leq 1$, there exists an $\tilde{O}(k^2 \alpha^{-3} \epsilon^{-2})$-space algorithm that returns a matching of size at least $\frac{(1-\epsilon)k}{2\alpha}$. The algorithm has $\tilde{O}(k^2 \alpha^{-2} \epsilon^{-2})$ update time.

Fafianie and Kratsch [16] studied kernelization streaming algorithms in the insert-only model for the NP-hard d-Set Matching problem (among others), which for $d = 2$, is equivalent to the k-matching problem on unweighted graphs. Their result implies a one-pass kernelization streaming algorithm for k-matching in unweighted graphs that computes a kernel of size $\mathcal{O}(k^2 \log k)$, runs in $\mathcal{O}(k^2)$ space, and has $\mathcal{O}(\log k)$ update time.

Chen et al. [6] studied algorithms for k-matching on the RAM model with limited computational resources, which is clearly very different from the streaming model. In order to translate their algorithm to the streaming model, it would require $\Omega(nk)$ space and multiple passes. However, we remark that one of the steps of our algorithm in the insert-only model was inspired by the constructition of reduced graphs introduced in [6].

Finally, there has been work on computing matchings in special graph classes, and with respect to parameters other than the cardinality of the matching (e.g., see [27, 28]).

3 Algorithms in Insert-Only Streaming Model

In this section, we give a streaming algorithm for p-WT-Matching, and hence for p-Matching as a special case, in the insert-only model. We start with some notations.

Given a weighted graph $G = (V = [n]^-, E)$ along with a weight function $wt : E(G) \to \mathbb{R}_{\geq 0}$, and a parameter k, we define a new function $\beta : E(G) \longrightarrow \mathbb{R}_{\geq 0} \times [n]^- \times [n]^-$ as follows: for $e = [u, v] \in E$, where $u < v$, let $\beta(e) = (wt(e), u, v)$. Observe that β is injective.

Define a partial order relation $\prec$ on $E(G)$ as follows: for any two distinct edges $e, e' \in E(G)$, $e \prec e'$ if $\beta(e)$ is lexicographically smaller than $\beta(e')$. For a vertex $v \in V$ and an edge e incident to v, define Γ_v to be the sequence of edges incident to v, sorted in a decreasing order w.r.t. $\prec$. We say that e is the *i-heaviest* edge w.r.t. v if e is the i-th element in Γ_v.

Let $f : V \to [4k^2]^-$ be a hash function, and let H be a subgraph of G. The function f partitions $V(H)$ into a collection of subsets $\mathcal{V} = \{V_0, V_1, \ldots, V_{4k^2-1}\}$, where each V_i consists of the vertices in $V(H)$ that have the same image under f. A matching M in H is said to be *nice* w.r.t. f if no two vertices of M belong to the same V_i in $\mathcal{V}$. When the function f is clear from the context, we will simply say that "M is nice." We define the *compact* subgraph of H under f, denoted $\mathcal{C}ompact(H, f)$, as the subgraph of H consisting of the edges e in H whose endpoints belong to different subsets V_i and V_j in $\mathcal{V}$, with $i \neq j$, and such that $\beta(e)$ is the maximum over all edges between V_i and V_j. Finally, we define the *reduced compact* subgraph of H under f, denoted $\mathcal{R}ed\text{-}\mathcal{C}om(H, f)$, by (1) for each pair (V_i, V_j) of subsets, selecting edges $e \in \mathcal{C}ompact(H, f)$ with endpoints in V_i and V_j such that e is among the $8k$ heaviest edges incident to vertices in V_i *and* among the $8k$ heaviest edges incident to vertices in V_j (in both subsets, if there are not that many edges, then include all edges); and then (2) retaining from the selected edges in (1) the $q = k(16k - 1)$ heaviest edges (again if there are not that many edges, include all edges). We have the following:

▶ **Lemma 2.** *The subgraph $\mathcal{C}ompact(H, f)$ has a nice k-matching if and only if $\mathcal{R}ed\text{-}\mathcal{C}om(H, f)$ has a nice k-matching. If this is the case, then the weight of a maximum weighted nice k-matching in $\mathcal{C}ompact(H, f)$ is equal to that in $\mathcal{R}ed\text{-}\mathcal{C}om(H, f)$.*

Proof. Define the auxiliary weighted graph Φ whose vertices are the subsets V_i in the collection $\mathcal{V}$, where $i \in [4k^2]^-$, such that there is an edge $[V_i, V_j]$ in Φ if some vertex $u \in V_i$ is adjacent to some vertex $v \in V_j$ in $\mathcal{C}ompact(H, f)$. We associate with edge $[V_i, V_j]$ the value $\beta([u, v])$ and associate the edge $[u, v]$ with $[V_i, V_j]$. Obviously, there is one-to-one correspondence between the nice k-matchings in $\mathcal{C}ompact(H, f)$ and the k-matchings in Φ. Let $\mathcal{H}$ be the subgraph of Φ formed by selecting edges $[V_i, V_j]$ such that $[V_i, V_j]$ is among the $8k$ heaviest edges incident to V_i and among the $8k$ heaviest edges incident to V_j. Let $\mathcal{H}'$ consist of the $q = k(16k - 1)$ heaviest edges in $\mathcal{H}$ (if $\mathcal{H}$ has at most q edges, let $\mathcal{H}' = \mathcal{H}$). Since there is a one-to-one correspondence between the nice k-matchings in $\mathcal{C}ompact(H, f)$ and the k-matchings of Φ, it suffices to prove the statement of the lemma with respect to matchings in Φ and $\mathcal{H}'$: namely, if Φ has a maximum weighted k-matching M then $\mathcal{H}'$ has a maximum weighted k-matching of the same weight as M.

Suppose that Φ has a maximum weighted k-matching M. Choose M such that the number of edges in M that remain in $\mathcal{H}$ is maximized. We first show that all the edges in M remain in $\mathcal{H}$. Suppose not, then there is an edge $[V_{i_0}, V_{i_1}] \in M$ such that $[V_{i_0}, V_{i_1}]$ is not among the $8k$ heaviest edges incident to one of its endpoints, say V_{i_1}. Since $|V(M)| = 2k < 8k$, it follows that there is a heaviest edge $[V_{i_1}, V_{i_2}]$ incident to V_{i_1} such that $\beta([V_{i_1}, V_{i_2}]) > \beta([V_{i_0}, V_{i_1}])$ and $V_{i_2} \notin V_M$. If $[V_{i_1}, V_{i_2}] \in \mathcal{H}$, then $(M - [V_{i_0}, V_{i_1}]) + [V_{i_1}, V_{i_2}]$ is a maximum weighted k-matching of Φ that contains more edges of $\mathcal{H}$ than M, contradicting our choice of M. It follows that $[V_{i_1}, V_{i_2}] \notin \mathcal{H}$. Then, $[V_{i_1}, V_{i_2}]$ is not among the $8k$ heaviest edges incident to V_{i_2}. Now apply the above argument to V_{i_2} to select the heaviest edge $[V_{i_2}, V_{i_3}]$ such that $\beta([V_{i_2}, V_{i_3}]) > \beta([V_{i_1}, V_{i_2}]) > \beta([V_{i_0}, V_{i_1}])$ and $V_{i_3} \notin V_M$. By applying the above argument j times, we obtain a sequence of j vertices $V_{i_1}, V_{i_2}, \ldots, V_{i_j}$, such that (1) $\{V_{i_2}, \ldots, V_{i_j}\} \cap V_M = \emptyset$; and (2) $V_{i_a} \neq V_{i_b}$ for every $a \neq b \in [j]$, which is guaranteed by $\beta([V_{i_a}, V_{i_{a+1}}]) < \beta([V_{i_{a+1}}, V_{i_{a+2}}]) < \cdots < \beta([V_{i_{b-1}}, V_{i_b}])$ and $[V_{i_a}, V_{i_{a+1}}]$ is the heaviest edge incident to V_{i_a} such that $V_{i_{a+1}} \notin V_M$. Since Φ is finite, the above process must end at an edge e not in M and such that $\beta(e)$ exceeds $\beta([V_{i_0}, V_{i_1}])$, contradicting our choice of M. Therefore, $M \subseteq E(\mathcal{H})$.

Now, choose a maximum weighted k-matching of $\mathcal{H}$ that maximizes the number of edges retained in $\mathcal{H}'$. Without loss of generality, call it M. We prove that the edges of M are retained in $\mathcal{H}'$, thus proving the lemma. Suppose that this is not the case. Since each vertex in $V(M)$ has degree at most $8k$ and one of its edges must be in M, the number of edges in $\mathcal{H}$ incident to the vertices in M is at most $2k(8k-1) + k = k(16k-1) = q$. It follows that there is an edge e in $\mathcal{H}'$ whose endpoints are not in M and such that $\beta(e)$ is larger than the $\beta()$ value of some edge in M, contradicting our choice of M. ◀

▶ **Lemma 3.** *Let $f : V \to [4k^2]^-$ be a hash function, and let H be a subgraph of G. There is an algorithm **Alg-Reduce(H, f)** that constructs $\mathcal{R}ed\text{-}\mathcal{C}om(H, f)$ and has both its time and space complexities bounded by $\mathcal{O}(|H| + k^2)$.*

We now present the streaming algorithm $\mathcal{A}_{\text{Insert}}$ for p-WT-Matching. Let $(\mathcal{S}, k)$ be an instance of p-WT-Matching, where $\mathcal{S} = (e_1, wt(e_1)), \ldots, (e_i, wt(e_i)), \ldots$. For $i \in \mathbb{N}$, let G_i be the subgraph of G consisting of the first i edges $e_1, \ldots, e_i$ of $\mathcal{S}$, and for $j \leq i$, let $G_{j,i}$ be the subgraph of G whose edges are $\{e_j, \ldots, e_i\}$; if $j > i$, we let $G_{j,i} = \emptyset$. Let f be a hash function chosen u.a.r. from a universal set $\mathcal{H}$ of hash functions mapping V to $[4k^2]^-$. The algorithm $\mathcal{A}_{\text{Insert}}$, after processing the i-th element $(e_i, wt(e_i))$, computes two subgraphs G_i^f and G_i^s defined as follows. For $i = 0$, $G_i^f = G_i^s = \emptyset$. For $i > 0$, let $\hat{i}$ be the largest multiple of q that is smaller than i, that is, $i = \hat{i} + p$, where $0 < p \leq q$; and let i^* be the largest multiple of q that is smaller than $\hat{i}$ if $\hat{i} > 0$, and 0 otherwise. The subgraph G_i^f is defined only when i is a multiple of q, and is defined recursively for $i = j \cdot q > 0$ as $G_i^f = \mathcal{R}ed\text{-}\mathcal{C}om(G_{\hat{i}}^f \cup G_{i^*+1,\hat{i}})$; that is, G_i^f is the reduced compact subgraph of the graph consisting of $G_{\hat{i}}^f$ plus the subgraph consisting of the edges encountered after e_{i^*}, starting from e_{i^*+1} up to $e_{\hat{i}}$. The subgraph G_i^s is defined as $G_i^s = G_{\hat{i}}^f \cup G_{i^*+1,i}$; that is, G_i^s consists of the previous (before i) reduced compact subgraph plus the subgraph consisting of the edges starting after i^* up to i. We refer to Figure 1 for an illustration of the definitions of G_i^f and G_i^s.

▶ **Lemma 4.** *For each $i \geq 1$, if G_i contains a maximum weighted k-matching, then with probability at least $1/2$, G_i^s contains a maximum weighted k-matching of G_i.*

Proof. Let $M = \{[u_0, u_1], \ldots, [u_{2k-2}, u_{2k-1}]\}$ be a maximum weighted k-matching in G_i, and let $V_M = \{u_0, \ldots, u_{2k-1}\}$. Since f is a hash function chosen u.a.r. from a universal set $\mathcal{H}$ of hash functions mapping V to $[4k^2]^-$, with probability at least $1/2$, f is perfect w.r.t. V_M [11]. Now, suppose that f is perfect w.r.t. V_M. Thus, M is a nice matching (w.r.t. f) in G_i. By the

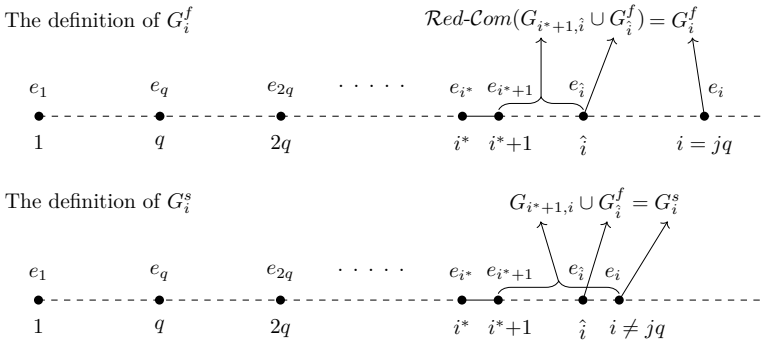

Figure 1 Illustration of the definitions of G_i^f and G_i^s.

definition of $\mathcal{C}ompact(G_i, f)$, there is a set M' of k edges $M' = \{[u'_0, u'_1], \ldots, [u'_{2k-2}, u'_{2k-1}]\}$ in $\mathcal{C}ompact(G_i, f)$ such that $\{f(u'_{2i}), f(u'_{2i+1})\} = \{f(u_{2i}), f(u_{2i+1})\}$ and $\beta([u'_{2i}, u'_{2i+1}]) \geq \beta([u_{2i}, u_{2i+1}])$ for $i \in [k]^-$. It follows that $wt([u'_{2i}, u'_{2i+1}]) \geq wt([u_{2i}, u_{2i+1}])$ for $i \in [k]^-$. Therefore, $\mathcal{C}ompact(G_i, f)$ contains a maximum weighted k-matching of G_i, namely $\{[u'_0, u'_1], \ldots, [u'_{2k-2}, u'_{2k-1}]\}$; moreover, this matching is nice. By Lemma 2, $\mathcal{R}ed\text{-}\mathcal{C}om(G_i, f)$ contains a maximum weighted k-matching of $\mathcal{C}ompact(G_i, f)$.

Next, we prove that G_i^s contains a maximum weighted k-matching of G_i. If $i \leq 2q$, then $G_i^s = G_{1,i} = G_i$ by definition, and hence G_i^s contains a maximum weighted k-matching of G_i. Suppose now that $i > 2q$. By definition, $G_i^s = G_{\hat{i}}^f \cup G_{i^*+1,i}$. (Recall that, by definition, $G_q^f = \emptyset$, $G_{2q}^f = \mathcal{R}ed\text{-}\mathcal{C}om(G_q^f \cup G_{1,q})$, $G_{3q}^f = \mathcal{R}ed\text{-}\mathcal{C}om(G_{2q}^f \cup G_{q+1,2q}), \ldots, G_{\hat{i}}^f = \mathcal{R}ed\text{-}\mathcal{C}om(G_{i^*}^f \cup G_{i^*-q+1,i^*})$.) For each $j \geq 1$ that is multiple of q, let $\mathcal{G}_j$ be the graph consisting of the edges that are in $G_{\hat{j}}^f \cup G_{j^*+1,\hat{j}}$ but are not kept in G_j^f. Consequently, $(\bigcup_{q \leq j < \hat{i}, j \text{ is a multiple of } q} \mathcal{G}_j) \bigcup G_{\hat{i}}^f = G_{i^*}$. By the definition of $\mathcal{R}ed\text{-}\mathcal{C}om(G_i, f)$, it is easy to verify that $\mathcal{R}ed\text{-}\mathcal{C}om(G_i, f)$ does not contain the edges in $\mathcal{G}_j$, for each $j \geq 1$. It follows that $\mathcal{R}ed\text{-}\mathcal{C}om(G_i, f)$ is a subgraph of G_i^s, and hence, G_i^s contains a maximum weighted k-matching of $\mathcal{R}ed\text{-}\mathcal{C}om(G_i, f)$, and hence of $\mathcal{C}ompact(G_i, f)$ by the above discussion. Since $\mathcal{C}ompact(G_i, f)$ contains a maximum weighted k-matching of G_i, G_i^s contains a maximum weighted k-matching of G_i. It follows that, with probability at least $1/2$, G_i^s contains a maximum weighted k-matching of G_i. ◀

The algorithm $\mathcal{A}_{\text{Insert}}$, when queried at the end of the stream, either returns a maximum weighted k-matching of G or the empty set. To do so, at every instant i, it will maintain a subgraph G_i^s that will contain the edges of the desired matching, from which this matching can be extracted. To maintain G_i^s, the algorithm keeps track of the subgraphs G_{i-1}^s, $G_{\hat{i}}^f$, the edges $e_{i^*+1}, \ldots, e_i$, and will use them in the computation of the subgraph G_i^s as follows. If i is not a multiple of q, then $G_i^s = G_{i-1}^s + e_i$, and the algorithm simply computes G_i^s as such. Otherwise (i.e., i is a multiple of q), $G_i^s = G_{\hat{i}}^f \cup G_{i^*+1,i}$, and the algorithm uses $G_{\hat{i}}^f$ and $G_{i^*+1,i} = \{e_{i^*+1}, \ldots, e_i\}$ to compute and return G_i^s; however, in this case (i.e., i is a multiple of q), the algorithm will additionally need to have G_i^f already computed, in preparation for the potential computations of subsequent G_j^s, for $j \geq i$. By Lemma 3, the subgraph G_i^f can be computed by invoking the algorithm **Alg-Reduce** in Lemma 3 on $G_{\hat{i}}^f \cup G_{i^*+1,\hat{i}}$, which runs in time $\mathcal{O}(q)$. Note that both $G_{\hat{i}}^f$ and $G_{i^*+1,\hat{i}}$ are available to $\mathcal{A}_{\text{Insert}}$ at each of the steps $\hat{i}+1, \ldots, i$. Therefore, the algorithm will stagger the $\mathcal{O}(q)$ many operations needed for the computation of G_i^f uniformly (roughly equally) over each of the steps $\hat{i}+1, \ldots, i$,

yielding an $\mathcal{O}(1)$ operations per step. Note that all the operations in **Alg-Reduce** can be explicitly listed, and hence, splitting them over an interval of q steps is easily achievable. Combining the above discussions and lemmas, we conclude with:

▶ **Lemma 5.** *The algorithm $\mathcal{A}_{Insert}$ runs in space $\mathcal{O}(k^2)$ and has update time $\mathcal{O}(1)$.*

▶ **Theorem 6.** *Let $0 < \delta < 1$ be a parameter. There is an algorithm for* p-WT-Matching *such that, on input $(\mathcal{S}, k)$, the algorithm outputs a matching M' satisfying that (1) if G contains a k-matching then, with probability at least $1 - \delta$, M' is a maximum weighted k-matching of G; and (2) if G does not contain a k-matching then $M' = \emptyset$. The algorithm runs in $\mathcal{O}(k^2 \log \frac{1}{\delta})$ space and has $\mathcal{O}(\log \frac{1}{\delta})$ update time. In particular, for any constant δ, the algorithm runs in space $\mathcal{O}(k^2)$ and has $\mathcal{O}(1)$ update time.*

Proof. Run $\lceil \log \frac{1}{\delta} \rceil$-many copies of algorithm $\mathcal{A}_{\text{Insert}}$ in parallel (i.e., using dove-tailing). Then, by the end of the stream, there are $\lceil \log \frac{1}{\delta} \rceil$ copies of G_m^s, where m is the length of the stream. Let G' be the union of all the G_m^s's produced by the runs of $\mathcal{A}_{\text{Insert}}$. If G' has a k-matching, let M' be a maximum weighted k-matching of G'; otherwise, let $M' = \emptyset$.

By Lemma 4, if G_m, i.e., G, contains a maximum weighted k-matching, with probability at least $1/2$, one copy of G_m^s contains a maximum weighted k-matching of G. Hence, with probability at least $1 - (1/2)^{\lceil \log \frac{1}{\delta} \rceil} \geq 1 - \delta$, G' contains a maximum weighted k-matching of G. It follows that if G contains a maximum weighted k-matching M then, with probability at least $1 - \delta$, G' contains a maximum weighted k-matching of the same weight as M and hence M' is a maximum weighted k-matching of G.

Observe that the graph G' is a subgraph of G. Therefore, statement (2) in the theorem clearly holds true. By Lemma 5, the above algorithm runs in space $\mathcal{O}(k^2 \log \frac{1}{\delta})$ and has update time $\mathcal{O}(\log \frac{1}{\delta})$, thus completing the proof. ◀

4 The Toolkit

In this section, we prove a theorem that can be useful in its own right for subset problems, that is, problems in which the goal is to compute a k-subset S ($k \in \mathbb{N}$) of some universe U such that S satisfies certain prescribed properties. Intuitively, the theorem states that, for any k-subset $S \subseteq U$, *w.h.p.* we can compute k subsets $T_1, \ldots, T_k$ of U that interact "nicely" with S. More specifically, (1) the sets T_i, for $i \in [k]$, are pairwise disjoint, (2) S is contained in their union $\bigcup_{i \in [k]} T_i$, and (3) each T_i contains exactly one element of S.

The above theorem will be used in Section 5 to design algorithms for p-Matching and p-WT-Matching in the dynamic streaming model. Intuitively speaking, the theorem will be invoked to obtain the sets T_i of vertices that *w.h.p.* induce the edges of the desired k-matching; however, these sets may not necessarily constitute the desired subgraph as they may not have "small" cardinalities. Sampling techniques will be used to select a smaller set of edges induced by the vertices of the T_i's that *w.h.p.* contains the edges of the k-matching.

A family $\mathcal{H}$ of hash functions, each mapping U to $[r]^-$, is called κ-wise independent if for any κ distinct keys $x_1, x_2, ..., x_\kappa \in U$, and any κ (not necessarily distinct) values $a_1, a_2, ..., a_\kappa \in [r]^-$, we have $\Pr_{h \in_{u.a.r} \mathcal{H}}[h(x_1) = a_1 \wedge h(x_2) = a_2 \wedge \cdots \wedge h(x_\kappa) = a_\kappa] = \frac{1}{r^\kappa}$.

Let $\mathbb{F}$ be a finite field. A κ-wise independent family $\mathcal{H}$ of hash functions can be constructed as follows (See Construction 3.32 in [35]): $\mathcal{H} = \{h_{a_0, a_1, \ldots, a_{\kappa-1}} : \mathbb{F} \to \mathbb{F}\}$, where $h_{a_0, a_1, \ldots, a_{\kappa-1}}(x) = a_0 + a_1 x + \cdots + a_{\kappa-1} x^{\kappa-1}$ for $a_0, \ldots, a_{\kappa-1} \in \mathbb{F}$.

The following theorem is proved in [35], and will be used in our discussion.

▶ **Theorem 7** (Corollary 3.34 in [35]). *For every $u, d, \kappa \in \mathbb{N}$, there is a family of κ-wise independent functions $\mathcal{H} = \{h : \{0,1\}^u \to \{0,1\}^d\}$ such that choosing a random function from $\mathcal{H}$ takes space $\mathcal{O}(\kappa \cdot (u+d))$. Moreover, evaluating a function from $\mathcal{H}$ takes time polynomial in u, d, κ.*

To prove our theorem, we proceed in two phases. We give an intuitive description of these two phases. In the first phase, we choose a hashing function f u.a.r. from an $\mathcal{O}(\ln k)$-wise independent set of hash functions, which hashes U to a set of $d_1 = \mathcal{O}(k/\ln k)$ integers. We use f to partition the universe U into d_1-many subsets U_i. Afterwards, we choose d_1 families $F_0, \ldots, F_{d_1-1}$ of hash functions, each containing $d_2 = \mathcal{O}(\ln k)$ functions, chosen independently and u.a.r. from a universal set of hash functions. The family F_i, $i \in [d_1]^-$, will be used restrictively to map the elements of U_i. Since each family F_i is chosen from a universal set of hash function, for the subset $S_i = S \cap U_i$, *w.h.p.* F_i contains a hash function f_i that is perfect w.r.t. S_i; that is, under the function f_i the elements of S_i are distinguished. This concludes the first phase of the process, which is described in **Algorithm 1**.

Algorithm 1 : An algorithm for partitioning U and constructing families of hash functions.

Input: $|U|, k \in \mathbb{N}$ where $|U| > 1$
Output: A family of sets of hash functions

1: let u and d be the unique positive integers satisfying $2^{u-1} < |U| \leq 2^u$ and $2^{d-1} < \frac{k}{\ln k} \leq 2^d$
2: choose f u.a.r. from $\mathcal{H}$, where $\mathcal{H} = \{h : \{0,1\}^u \to \{0,1\}^d\}$ is a $\lceil 12 \ln k \rceil$-wise independent set of hash functions
3: let $\mathcal{H}'$ be a set of universal hash functions from U to $[\lceil 13 \ln k \rceil^2]^-$
4: let F_i, for $i \in [2^d]^-$, be a set of $\lceil 8 \ln k \rceil$ hash functions chosen independently and u.a.r. from $\mathcal{H}'$
5: return $\{f, F_0, \ldots, F_{2^d-1}\}$

In the second phase, we define a relation $\mathcal{G}$ (from U) that, for each $x \in U$, associates a set $\mathcal{G}(x)$ of integers. This relation extends the hash functions in the F_j's above by (1) ensuring that elements in different parts of U (w.r.t. the partitioning) are distinguished, in the sense that they are associated with subsets of integers that are contained in disjoint intervals of integers; and (2) maintaining the property that elements of the same part U_j that are distinguished under some function in F_j remain so under the extended relation. To do so, for each part U_j, we associate an "offset" and create a large gap between any two (consecutive) offsets; we will ensure that all the elements in the same U_j fall within the same interval determined by two consecutive offsets. To compute the set $\mathcal{G}(x)$, for an element $x \in U_j$, we start with an offset o_j that depends solely on U_j ($o_j = j \cdot d_2 \cdot d_3$ in **Algorithm 2**), and consider every function in the family F_j corresponding to U_j. For each such function h_i, we associate an offset o'_i ($o'_i = (i-1) \cdot d_3$ in **Algorithm 2**), and for x and that particular function h_i, we add to $\mathcal{G}(x)$ the value $g(j, i, x) = o_j + o'_i + h_i(x)$. The above phase is described in **Algorithm 2**.

Now that the relations $\mathcal{G}(x)$, for $x \in U$, have been defined, we will show in the following theorem that, for any k-subset S of U, *w.h.p.* there exist k distinct elements $i_0, \ldots, i_{k-1}$, such that their pre-images $\mathcal{G}^{-1}(i_0), \ldots, \mathcal{G}^{-1}(i_{k-1})$ are pairwise disjoint, contain all elements of S, and each pre-image contains exactly one element of S; those pre-images serve as the desired sets T_i, for $i \in [k]$.

Consider **Algorithm 1** and **Algorithm 2**, and refer to them for the terminologies used in the subsequent discussions. For $i \in [d_1 \cdot d_2 \cdot d_3]^-$, define $T_i = \{x \in U \mid i \in \mathcal{G}(x)\}$. We define next two sequences of intervals, and prove certain properties about them, that will

Algorithm 2 : An algorithm that defines the relation $\mathcal{G}$ from U to $[d_1 \cdot d_2 \cdot d_3]^-$.

Input: $x \in U$, $k \in \mathbb{N}$, $\{f, F_0, \ldots, F_{d_1-1}\}$ is from Algorithm 1, where $|F_0| = \cdots = |F_{d_1-1}|$
Output: a set $\mathcal{G}(x)$

1: let $d_2 = |F_0| = \cdots = |F_{d_1-1}|$ and $d_3 = \lceil 13 \ln k \rceil^2$
2: $\mathcal{G}(x) = \emptyset$
3: compute $f(\llcorner x \lrcorner)$ and let j be the integer such that $\llcorner j \lrcorner = f(\llcorner x \lrcorner)$
4: **for** $i = 1$ to d_2 **do**
5: let h_i be the i-th function in F_j (assuming an arbitrary ordering on F_j)
6: let $g(j, i, x) = j \cdot d_2 \cdot d_3 + (i-1) \cdot d_3 + h_i(x)$ and let $\mathcal{G}(x) = \mathcal{G}(x) \cup \{g(j, i, x)\}$
7: return $\mathcal{G}(x)$

be used in the proof of Theorem 9. For $q \in [d_1 \cdot d_2]^-$, let $I_q = \{r \mid q \cdot d_3 \leq r < (q+1) \cdot d_3\}$. For $t \in [d_1]^-$, let $I'_t = \{r \mid t \cdot d_2 \cdot d_3 \leq r < t \cdot d_2 \cdot d_3 + d_2 \cdot d_3\}$. Note that each interval I'_t is partitioned into the d_2-many intervals I_q, for $q = t \cdot d_2, \ldots, t \cdot d_2 + d_2 - 1$.

▶ **Lemma 8.** *The following statements hold: (A) For any two distinct integers $a, b \in I_q$, where $q \in [d_1 \cdot d_2]^-$, we have $T_a \cap T_b = \emptyset$. (B) For $t \in [d_1]^-$, we have $\mathcal{G}(U_t) \subseteq I'_t$. Moreover, for any $a \in I'_t, b \in I'_s$, where $s \neq t$, we have $T_a \cap T_b = \emptyset$.*

▶ **Theorem 9.** *For any subset $S \subseteq U$ of cardinality $k \geq 2$, with probability at least $1 - \frac{4}{k^3 \ln k}$, there exist k sets $T_{i_0}, \ldots, T_{i_{k-1}}$ such that: (1) $|T_{i_j} \cap S| = 1$ for $j \in [k]^-$, (2) $S \subseteq \cup_{j \in [k]^-} T_{i_j}$, and (3) $T_{i_j} \cap T_{i_l} = \emptyset$ for $j \neq l \in [k]^-$.*

Proof. For $j \in [d_1]^-$, let U_j be the set of elements in U whose image is $\llcorner j \lrcorner$ under f (defined in Step 2 of **Algorithm 1**), that is $U_j = \{y \in U \mid f(\llcorner y \lrcorner) = \llcorner j \lrcorner\}$. Clearly, the sets U_j, for $j \in [d_1]^-$, partition the universe U. We will show that, with probability at least $1 - \frac{4}{k^3 \ln k}$, there exist k sets $T_{i_0}, \ldots, T_{i_{k-1}}$ that satisfy conditions (1)–(3) in the statement of the theorem.

Let $S \subseteq U$ be any subset such that $|S| = k$. For $j \in [d_1]^-$ and $y \in S$, let $X_{y,j}$ be the random variable defined as $X_{y,j} = 1$ if $f(\llcorner y \lrcorner) = \llcorner j \lrcorner$ and 0 otherwise. Let $X_j = \sum_{y \in S} X_{y,j}$, and $S_j = \{y \in S \mid f(\llcorner y \lrcorner) = \llcorner j \lrcorner\}$. Thus, $|S_j| = X_j$. Since f is $\lceil 12 \ln k \rceil$-wise independent, the random variables $X_{y,j}$, for $y \in S$, are $\lceil 12 \ln k \rceil$-wise independent and $\Pr(X_{y,j} = 1) = \frac{1}{d_1}$. Thus, $E[X_j] = |S| \cdot \frac{1}{d_1}$. Since $d_1 = 2^d$ and $2^{d-1} < \frac{k}{\ln k} \leq 2^d$ by definition, we have $\frac{k}{\ln k} \leq d_1 < \frac{2k}{\ln k}$ and $\frac{\ln k}{2} < E[X_j] \leq \ln k$. Applying Theorem 2 in [33] with $\mu = E[X_j]$ and $\delta = \frac{12 \ln k}{E[X_j]} > 1$, we get $\Pr(X_j \geq (1+\delta)E[X_j]) \leq e^{-E[X_j]\delta/3} = \frac{1}{k^4}$. Since $E[X_j] \leq \ln k$ and $\delta = \frac{12 \ln k}{E[X_j]}$, we have $(1+\delta)E[X_j] \leq 13 \ln k$. Hence, $\Pr(X_j \geq 13 \ln k) \leq \Pr(X_j \geq (1+\delta)E[X_j]) \leq \frac{1}{k^4}$. Let $\mathcal{E}$ denote the event $\bigwedge_{i \in [d_1]^-} (X_i \leq 13 \ln k)$. By the union bound, we have $\Pr(\mathcal{E}) \geq 1 - \frac{d_1}{k^4} \geq 1 - \frac{2}{k^3 \ln k}$, where the last inequality holds since $d_1 < 2k / \ln k$.

Assume that event $\mathcal{E}$ occurs, i.e., that $|S_j| \leq 13 \ln k$ holds for $j \in [d_1]^-$. Consider Step 4 in **Algorithm 1**. Fix $j \in [d_1]^-$, and let E_j be the event that F_j does not contain any perfect hash function w.r.t. S_j. Let h be a hash function picked from $\mathcal{H}'$ u.a.r. Since $|S_j| \leq 13 \ln k$ (by assumption), by Theorem 11.9 in [11], with probability at least $1/2$, h is perfect w.r.t. S_j. Since F_j consists of $\lceil 8 \ln k \rceil$ hash functions chosen independently and u.a.r. from $\mathcal{H}'$, we have $\Pr(E_j) \leq (1/2)^{\lceil 8 \ln k \rceil} < \frac{1}{k^4}$. Applying the union bound, we have $\Pr(\cup_{j \in [d_1]^-} E_j) \leq \frac{d_1}{k^4} < \frac{2}{k^3 \ln k}$. Let $\mathcal{E}'$ be the event that there exist d_1 functions $f_0, f_1, \ldots, f_{d_1-1}$ such that $f_j \in F_j$ and f_j is perfect w.r.t. S_j, $j \in [d_1]^-$. Therefore, $\Pr(\mathcal{E}') \geq \Pr(\mathcal{E})(1 - \Pr(\cup_{j \in [d_1]^-} E_j)) \geq 1 - \frac{4}{k^3 \ln k} + \frac{4}{k^6 \ln^2 k} \geq 1 - \frac{4}{k^3 \ln k}$. Suppose that such a set $\{f_0, \ldots, f_{d_1-1}\}$ of functions exists. Let $\eta(q)$ be the iteration number i in Step 5 of **Algorithm 2** during which $f_q \in F_q$ is chosen, for $q \in [d_1]^-$. We define the following (multi-)set B as follows. For each $q \in [d_1]^-$, and for

element $x \in S_q$, add to B the element $g(q, \eta(q), x))$ defined in Steps 5–6 of **Algorithm 2** (by $\{f, f_0, \ldots, f_{k-1}\}$). Observe that, by the definition of B, for every $x \in S$, there exists $a \in B$ such that $x \in T_a$. We will show next that B contains exactly k distinct elements, and that, for any $a \neq b \in B$, it holds that $T_a \cap T_b = \emptyset$. The above will show that the sets $\{T_a \mid a \in B\}$ satisfy conditions (1)–(3) of the theorem, thus proving the theorem.

It suffices to show that for any two distinct elements of S, the corresponding elements added to B are distinct. Let x_1 and x_2 be two distinct elements of S. Assume that $x_1 \in S_j$ and $x_2 \in S_l$, where $j, l \in [d_1]^-$. We distinguish two cases based on whether or not $j = l$.

If $j = l$, we have $g(j, \eta(j), x_1) = j \cdot d_2 \cdot d_3 + (\eta(j) - 1) \cdot d_3 + f_j(x_1)$ and $g(j, \eta(j), x_2) = j \cdot d_2 \cdot d_3 + (\eta(j) - 1) \cdot d_3 + f_j(x_2)$. Since f_j is perfect w.r.t. S_j, we have $g(j, \eta(j), x_1) \neq g(j, \eta(j), x_2)$. Moreover, both $g(j, \eta(j), x_1)$ and $g(j, \eta(j), x_2)$ are in $I_{j \cdot d_2 + (\eta(j)-1)}$ (since $0 \leq h_j(x_1), h_j(x_2) < d_3$), where $j \cdot d_2 + (\eta(j) - 1) \leq (d_1 - 1) \cdot d_2 + (d_2 - 1) \in [d_1 \cdot d_2]^-$. By part (A) of Lemma 8, it holds that $T_{g(j,\eta(j),x_1)} \cap T_{g(j,\eta(j),x_2)} = \emptyset$.

Suppose now that $j \neq l$. By definition of S_j, S_l, U_j, U_l, we have $S_j \subseteq U_j$ and $S_l \subseteq U_l$. Consequently, $g(j, \eta(j), x_1) \in \mathcal{G}(U_j)$ and $g(l, \eta(l), x_2) \in \mathcal{G}(U_l)$ hold. By part (B) of Lemma 8, we have $\mathcal{G}(U_j) \subseteq I'_j$ and $\mathcal{G}(U_l) \subseteq I'_l$. Therefore, $g(j, \eta(j), x_1) \neq g(l, \eta(l), x_2)$. Moreover, $T_{g(j,\eta(j),x_1)} \cap T_{g(l,\eta(l),x_2)} = \emptyset$ holds by part (B) of Lemma 8 as well. ◀

▶ **Theorem 10.** ***Algorithm 1*** *runs in space* $\mathcal{O}(k + (\log k)(\log |U|))$*, and* ***Algorithm 2*** *runs in space* $\mathcal{O}(\log k)$ *and in time polynomial in* $\log |U|$*.*

Proof. In **Algorithm 1**, since f is $\lceil 12 \ln k \rceil$-wise independent, by Theorem 7, storing f uses space $\mathcal{O}(\ln k \cdot \max\{u, d\}) = \mathcal{O}((\log k)(\log |U|))$ (since $k \leq |U|$). Storing a universal hash function uses $\mathcal{O}(1)$ space, and thus storing $\{F_0, \ldots, F_{d_1 - 1}\}$ uses $\mathcal{O}(d_1 \cdot d_2) = \mathcal{O}(k)$ space. Therefore, **Algorithm 1** can be implemented in space $\mathcal{O}(k + (\log k)(\log |U|))$.

For **Algorithm 2**, since $\mathcal{G}(x)$ contains exactly d_2 elements, storing $\mathcal{G}(x)$ takes $\mathcal{O}(d_2) = \mathcal{O}(\ln k)$ space. In Step 3, again by Theorem 7, computing $f(\llcorner x \lrcorner)$ takes time polynomial in $\log |U|$ and $\log k$, since f is a $\lceil 12 \ln k \rceil$-wise independent hash function from $\{0,1\}^u$ to $\{0,1\}^d$. Computing j in Step 3 takes time polynomial in $d = \mathcal{O}(\log k)$ since $f(\llcorner x \lrcorner) \in \{0,1\}^d$. Therefore, Step 3 can be performed in time polynomial in $\log |U|$ and $\log k$, and hence polynomial in $\log |U|$ (since $k \leq |U|$). Step 6 can be implemented in time polynomial in $\log k$, since $|F_j| = \lceil 8 \ln k \rceil$. Altogether, **Algorithm 2** takes time polynomial in $\log |U|$. This completes the proof. ◀

5 Algorithms in Dynamic Streaming Model

In this section, we present results on p-Matching and p-WT-Matching in the dynamic streaming model. The algorithm uses the toolkit developed in the previous section, together with the ℓ_0-sampling technique discussed in Section 2. We first give a high-level description of how the algorithm works.

Let $\mathcal{S}$ be a graph stream of a weighted graph $G = (V = [n]^-, E)$ along with the weight function $wt : E(G) \longrightarrow \mathbb{R}_{\geq 0}$, and k be a parameter. Suppose G has W distinct weights. We will hash the vertices of the graph to a range R of size $\mathcal{O}(k \log^2 k)$. For each element $(e = [u, v], wt(e), op) \in \mathcal{S}$, where op is either insertion or deletion, we use the relation $\mathcal{G}$, discussed in Section 4, and compute the two sets $\mathcal{G}(u)$ and $\mathcal{G}(v)$. For each $i \in \mathcal{G}(u)$ and each $j \in \mathcal{G}(v)$, we associate an instance of an ℓ_0-sampler primitive, call it $\mathcal{C}_{i,j,wt(u,v)}$, and update it according to the operation op. Recall that it is assumed that the weight of every edge does not change throughout the stream.

The solution computed by the algorithm consists of a set of edges created by invoking each of the $\tilde{O}(Wk^2)$ ℓ_0-sampler algorithms to sample at most one edge from each $\mathcal{C}_{i,j,w}$, for each pair of i, j in the range R and each edge-weight of the graph stream.

The intuition behind the above algorithm (i.e., why it achieves the desired goal) is the following. Suppose that there exists a maximum weighted k-matching M in G, and let $M = \{[u_0, u_1], \ldots, [u_{2k-2}, u_{2k-1}]\}$. By Theorem 9, *w.h.p.* there exist $i_0, \ldots, i_{2k-1}$ in the range R such that $u_j \in T_{i_j}$, for $j \in [2k]^-$, and such that the T_{i_j}'s are pairwise disjoint. Consider the k ℓ_0-samplers $\mathcal{C}_{i_{2j}, i_{2j+1}, wt(u_{2j}, u_{2j+1})}$, where $j \in [k]^-$. Then, *w.h.p.*, the k edges sampled from these k ℓ_0-samplers are the edges of a maximum weighted k-matching (since the T_{i_j}'s are pairwise disjoint) whose weight equals that of M.

Algorithm 3 The streaming algorithm $\mathcal{A}_{dynamic}$ in the dynamic streaming model.

$\mathcal{A}_{dynamic}$-Preprocess: The preprocessing algorithm

Input: $n = |V(G)|$ and a parameter $k \in \mathbb{N}$

let $\mathcal{C}$ be a set of ℓ_0-samplers and $\mathcal{C} = \emptyset$

let $\{f, F_0, F_1, \ldots, F_{d_1-1}\}$ be the output of **Algorithm 1** on input $(n, 2k)$

$\mathcal{A}_{dynamic}$-Update: The update algorithm

Input: An update $(e = [u, v], wt(e), op) \in \mathcal{S}$, where op is either insertion or deletion

let $\mathcal{G}(u)$ be the output of **Algorithm 2** on input $(u, 2k, \{f, F_0, F_1, \ldots, F_{d_1-1}\})$

let $\mathcal{G}(v)$ be the output of **Algorithm 2** on input $(v, 2k, \{f, F_0, F_1, \ldots, F_{d_1-1}\})$

for $i \in \mathcal{G}(u)$ and $j \in \mathcal{G}(v)$ **do**

  if $\mathcal{C}_{i,j,wt(uv)} \notin \mathcal{C}$ **then**

    create the ℓ_0-sampler $\mathcal{C}_{i,j,wt(uv)}$

  feed $\langle [u, v], op \rangle$ to the ℓ_0-sampler algorithm $\mathcal{C}_{i,j,wt(u,v)}$ with parameter δ

$\mathcal{A}_{dynamic}$-Query: The query algorithm after an update

let $E' = \emptyset$

for each $\mathcal{C}_{i,j,w} \in \mathcal{C}$ **do**

  apply the ℓ_0-sampler $\mathcal{C}_{i,j,w}$ with parameter δ to sample an edge e

  if $\mathcal{C}_{i,j,w}$ does not FAIL then set $E' = E' \cup \{e\}$

return a maximum weighted k-matching in $G' = (V(E'), E')$ if any; otherwise, return $\emptyset$

Choose $\delta = \frac{1}{20k^4 \ln(2k)}$. Let $\mathcal{A}_{dynamic}$ be the algorithm consisting of the sequence of three subroutines/algorithms $\boldsymbol{\mathcal{A}_{dynamic}}$**-Preprocess**, $\boldsymbol{\mathcal{A}_{dynamic}}$**-Update**, and $\boldsymbol{\mathcal{A}_{dynamic}}$**-Query**, where $\boldsymbol{\mathcal{A}_{dynamic}}$**-Preprocess** is applied at the beginning of the stream, $\boldsymbol{\mathcal{A}_{dynamic}}$**-Update** is applied after each operation, and $\boldsymbol{\mathcal{A}_{dynamic}}$**-Query** is applied whenever the algorithm is queried for a solution after some update operation. Without loss of generality, and for convenience, we will assume that the algorithm is queried at the end of the stream $\mathcal{S}$, even though the query could take place after any arbitrary operation.

▶ **Lemma 11.** *Let M' be the matching obtained by applying the algorithm $\mathcal{A}_{dynamic}$ with $\boldsymbol{\mathcal{A}_{dynamic}}$**-Query** invoked at the end of $\mathcal{S}$. If G contains a k-matching then, with probability at least $1 - \frac{11}{20k^3 \ln(2k)}$, M' is a maximum weighted k-matching of G.*

▶ **Theorem 12.** *The algorithm $\mathcal{A}_{dynamic}$ outputs a matching M' such that (1) if G contains a k-matching then, with probability at least $1 - \frac{11}{20k^3 \ln(2k)}$, M' is a maximum weighted k-matching of G; and (2) if G does not contain a k-matching then $M' = \emptyset$. Moreover, the algorithm $\mathcal{A}_{dynamic}$ runs in $\tilde{O}(Wk^2)$ space and has $\tilde{O}(1)$ update time.*

Proof. First, observe that G' is a subgraph of G, since it consists of edges sampled from subsets of edges in G. Therefore, statement (2) in the theorem clearly holds true. Statement (1) follows from Lemma 11. Next, we analyze the update time of algorithm $\mathcal{A}_{dynamic}$.

From **Algorithm 1** and **Algorithm 2**, we have $d_1 = O(\frac{k}{\ln k})$, $d_2 = O(\ln k)$, $d_3 = O(\ln^2 k)$ and $|F_i| = O(\ln k)$ for $i \in [d_1]^-$. Thus, $|\mathcal{G}(u)| = O(\ln k)$ holds for all $u \in V$. For the update time, it suffices to examine Steps 1–6 of $\boldsymbol{\mathcal{A}_{dynamic}}$**-Update** By Theorem 10, Steps 1–2 take time polynomial in $\log n$, which is $\tilde{O}(1)$. For Step 4, we can index $\mathcal{C}$ using a sorted sequence of triplets (i, j, w), where $i, j \in [d_1 \cdot d_2 \cdot d_3]^-$ and w ranges over all possible weights. Since $d_1 = \mathcal{O}(\frac{k}{\ln k})$, $d_2 = \mathcal{O}(\ln k)$ and $d_3 = \mathcal{O}(\ln^2 k)$, we have $|\mathcal{C}| = \mathcal{O}((d_1 \cdot d_2 \cdot d_3)^2 \cdot W) = \mathcal{O}(Wk^2 \ln^4 k) = \tilde{O}(Wk^2)$. Using binary search on $\mathcal{C}$, one execution of Step 4 takes time $\mathcal{O}(\log W + \log k)$. Since $|\mathcal{G}(u)| = O(\ln k)$ for every $u \in V$, and since by Lemma 1 an ℓ_0-sampler algorithm has $\tilde{O}(1)$ update time, Steps 3–6 take time $\mathcal{O}(\ln^2 k) \cdot (\mathcal{O}(\log W + \log k) + \tilde{O}(1)) = \tilde{O}(1)$. Therefore, the overall update time is $\tilde{O}(1)$.

Now, we analyze the space complexity of the algorithm. First, consider $\boldsymbol{\mathcal{A}_{dynamic}}$**-Preprocess**. Obviously, Step 1 uses $\mathcal{O}(1)$ space. Steps 1–2 use space $\mathcal{O}(k+(\log k)(\log n))$ (including the space used to store $\{f, F_0, \ldots, F_{d_1-1}, \mathcal{C}\}$) by Theorem 10. Altogether, $\boldsymbol{\mathcal{A}_{dynamic}}$**-Preprocess** runs in space $\mathcal{O}(k+(\log k)(\log n))$. Next, we discuss $\boldsymbol{\mathcal{A}_{dynamic}}$**-Update**. Steps 1–2 take space $O(\ln k)$ by Theorem 10. Observe that the space used in Steps 3–6 is dominated by the space used by the set $\mathcal{C}$ of ℓ_0-samplers. Since $\delta = \frac{1}{20k^4 \ln(2k)}$, an ℓ_0-sampler algorithm uses space $\mathcal{O}(\log^2 n \cdot \log k)$, by Lemma 1. Since $|\mathcal{C}| = \mathcal{O}(Wk^2 \ln^4 k)$, Steps 3–6 use space $\mathcal{O}(Wk^2 \log^2 n \log^5 k) = \tilde{\mathcal{O}}(Wk^2)$. Finally, consider $\boldsymbol{\mathcal{A}_{dynamic}}$**-Query** The space in Steps 1 – 4 is dominated by the space used by $\mathcal{C}$ and the space needed to store the graph G', and hence E'. By the above discussion, $\mathcal{C}$ takes space $\tilde{\mathcal{O}}(Wk^2)$. Since at most one edge is sampled from each ℓ_0-sampler instance and $|\mathcal{C}| = \mathcal{O}(Wk^2 \ln^4 k)$, we have $|E'| = |\mathcal{C}| = \tilde{O}(Wk^2)$. Step 5 utilizes space $\mathcal{O}(|E'|)$ [17, 18]. Therefore, $\boldsymbol{\mathcal{A}_{dynamic}}$**-Query** runs in space $\tilde{O}(Wk^2)$. It follows that the space complexity of $\mathcal{A}_{dynamic}$ is $\tilde{O}(Wk^2)$. ◀

Using Theorem 12, and following the same approach in [7], we obtain the following:

▶ **Theorem 13.** *Let $0 < \epsilon < 1$. There exists an algorithm for p-*WT-Matching *that computes a matching M' such that (1) if G contains a maximum weighted k-matching M, then with probability at least $1 - \frac{11}{20k^3 \ln(2k)}$, $wt(M') > (1-\epsilon)wt(M)$; and (2) if G does not contain a k-matching then $M' = \emptyset$. Moreover, the algorithm runs in $\tilde{O}(k^2\epsilon^{-1} \log W')$ space and has $\tilde{O}(1)$ update time, where W' is the ratio of the max weight to min weight.*

Proof. For each edge $e \in E$, round $wt(e)$ and assign it a new weight of $(1+\epsilon)^i$ such that $(1+\epsilon)^{i-1} < wt(e) \leq (1+\epsilon)^i$. Thus, there are $\mathcal{O}(\epsilon^{-1} \log W')$ distinct weights after rounding. By Theorem 12, the space and update time are $\tilde{\mathcal{O}}(k^2\epsilon^{-1} \log W')$ and $\tilde{\mathcal{O}}(1)$ respectively, and the success probability is at least $1 - \frac{11}{20k^3 \ln(2k)}$. Now we prove that $wt(M') > (1-\epsilon)wt(M)$.

Let $e \in M$ and let e' be the edge sampled from the ℓ_0-sampler that e is fed to. It suffices to prove that $wt(e') > (1-\epsilon)wt(e)$. Assume that $wt(e)$ is rounded to $(1+\epsilon)^i$. Then, $wt(e')$ is rounded to $(1+\epsilon)^i$ as well. If $wt(e') \geq wt(e)$, we are done; otherwise, $(1+\epsilon)^{i-1} < wt(e') < wt(e) \leq (1+\epsilon)^i$. It follows that $wt(e') > (1+\epsilon)^{i-1} \geq wt(e)/(1+\epsilon) > (1-\epsilon)wt(e)$. ◀

The following theorem is a consequence of Theorem 12 (applied with $W = 1$):

▶ **Theorem 14.** *There is an algorithm for p-*Matching *that computes a matching M' satisfying that (1) if G contains a k-matching then, with probability at least $1 - \frac{11}{20k^3 \ln(2k)}$, M' is a k-matching of G; and (2) if G does not contain a k-matching then $M' = \emptyset$. Moreover, the algorithm runs in $\tilde{O}(k^2)$ space and has $\tilde{O}(1)$ update time.*

6 Lower Bound

In this section, we discuss lower bounds on the space complexity of randomized streaming algorithms for p-MATCHING in the insert-only model (hence also in the dynamic model), and for p-WT-MATCHING in the dynamic model. These lower bound results, in conjunction with the algorithms given in the previous sections, show that the space complexity achieved by our algorithms is optimal (modulo a poly-logarithmic factor in the input size).

We will use the *one-way communication model* to prove lower bounds on the space complexity of randomized streaming algorithms for p-MATCHING and p-WT-MATCHING. In this model, there are two parties, Alice and Bob, each receiving x and y respectively, who wish to compute $f(x, y)$. Alice is permitted to send Bob a single message M, which only depends on x and Alice's random coins. Then Bob outputs b, which is his guess of $f(x, y)$. Here, b only depends on y, M, and Bob's random coins. We say the protocol computing f with success probability $1 - \delta$ if $\Pr(b = f(x, y)) \geq 1 - \delta$ for every x and y.

For p-MATCHING, we have the following theorem, which is implied from the space complexity lower-bound proof given in [7] for maximum matching.

▶ **Theorem 15** ([7]). *Any randomized streaming algorithm for p-MATCHING that, with probability at least 2/3, computing a k-matching uses $\Omega(k^2)$ bits.*

For p-WT-MATCHING, we start by defining the following problem:

> PARTIAL MAXIMIZATION: Alice has a sequence $a = \langle a_1, a_2, \ldots, a_n \rangle$ of numbers, where each $a_i \in [1, n^{1+\epsilon}]$, and Bob has a subset $T \subset [n]$. Compute $\max_{i \in [n] \setminus T} a_i$.

Let X, Y, Z, Z_1, Z_2 be random variables. Define the *Shannon entropy* of X as $H(X) = \sum_x \Pr(X = x) \log(\frac{1}{\Pr(X=x)})$. Define the *conditional entropy* of Z_1 given Z as $H(Z_1 \mid Z) = \sum_z H(Z_1|Z = z) \Pr(Z = z)$, and the *mutual information* as $I(Z_1; Z) = H(Z) - H(Z \mid Z_1)$. Define the *conditional mutual information* of Z_1, Z_2 given Z as $I(Z_1; Z_2 \mid Z) = H(Z_1 \mid Z) - H(Z_1 \mid Z_2, Z)$. $X \to Y \to Z$ is said to form a *Markov chain* if the conditional distribution of Z depends only on Y and is conditionally independent of X.

▶ **Theorem 16.** *For any constant $0 \leq \delta < 1$, any randomized one-way communication protocol for* PARTIAL MAXIMIZATION *with success probability at least $1 - \delta$ has communication complexity $\Omega(n \log n)$ bits.*

Proof. The proof has a similar fashion as Augmented Indexing problem [29, 10]. Consider the case where each X_j, for $j \in [n]$, is picked uniformly at random from $[(j-1) \cdot n^\epsilon + 1, j \cdot n^\epsilon]$. Note that $X_1 < X_2 < \cdots < X_n$ and that $H(X_j) = \epsilon \log n$ for each $j \in [n]$. For each $j \in [n]$, let $T_j = [j + 1, n]$ and let X'_j be Bob's guess of $\max_{i \in [n] \setminus T_j} X_i = X_j$. Let M be the message sent from Alice to Bob. Since $\Pr(X'_j = X_j) \geq 1 - \delta$ and $X_j \to (M, X_{j+1}, X_{j+2}, \ldots, X_n) \to X'_j$ is a Markov chain, by Fano's Inequality [13], for all $j \in [n]$, we have $H(X_j \mid M, X_{j+1}, \ldots, X_n) \leq \delta \cdot \epsilon \log n + 1$, and hence,

$$\begin{aligned} I(X_j; M \mid X_{j+1}, \ldots, X_n) &= H(X_j \mid X_{j+1}, \ldots, X_n) - H(X_j \mid M, X_{j+1}, \ldots, X_n) \\ &= H(X_j) - H(X_j \mid M, X_{j+1}, \ldots, X_n) \\ &\geq (1 - \delta)\epsilon \log n - 1, \end{aligned}$$

where the second equality holds because $X_j, X_{j+1}, \ldots, X_n$ are mutually independent. By Theorem 2.5.2 of [13], $H(M) \geq I(X_1, X_2, \ldots, X_n; M) = \sum_{j=1}^{n} I(X_j; M | X_{j+1}, \ldots, X_n) = \Omega(n \log n)$. Finally, by Theorem 2.6.4 of [13] the message M has at least $2^{\Omega(n \log n)}$ possibilities, hence the length of the longest possible M is $\Omega(n \log n)$, completing the proof. ◀

Theorem 16, plus a reduction from the Partial Maximization problem to the p-WT-Matching problem with parameter value $k = 1$, gives directly the following result.

▶ **Theorem 17.** *Any randomized streaming algorithm for p-*WT-Matching *that has success probability at least* $2/3$ *requires space* $k^2 \cdot \Omega(W(\log W + 1))$.

7 Concluding Remarks

In this paper, we presented streaming algorithms for the fundamental k-matching problem, for both unweighted and weighted graphs, and in both the insert-only and dynamic streaming models. While matching the best space complexity of known algorithms, which has been proved to be either optimal or near-optimal, our algorithms have much faster update times. For the insert-only model, our algorithm is optimal in both space and update time complexities. For the dynamic model, according to the new lower bounds we developed, our algorithms are near-optimal (i.e., optimal up to a poly-logarithmic factor) in both space and update time complexities. Our result for the weighted k-matching problem was achieved using a newly-developed structural result that is of independent interest. We believe that our results and techniques can have wider applicability for other fundamental graph problems.

Most work on weighted graph streams, including ours, assumes that the weight of an edge is unchanged in the stream [1, 2, 7, 20, 23]. We give an interesting observation below to justify this assumption and show that, if this assumption is lifted, then the space complexity of the k-matching problem can be much larger than the desirable space complexity for streaming algorithms. This lower bound is derived by a reduction from the following problem:

> Given a data stream $\mathcal{S}' = x_1, x_2, \ldots, x_m$, where $x_i \in \{1, \ldots, n'\}$, let $c_i = |\{j \mid x_j = i\}|$ be the number of occurrences of i in the stream $\mathcal{S}'$. Compute $F_\infty = \max_{1 \le i \le n'} c_i$.

▶ **Theorem 18** ([32])**.** *For data streams of length* m*, any randomized streaming algorithm computing* F_∞ *to within a* (1 ± 0.2) *factor with probability* $2/3$ *requires space* $\Omega(\min\{m, n'\})$.

Now we consider the p-WT-Matching problem in the more generalized dynamic streaming model, in which an instance of p-WT-Matching is given by a parameter k and a stream $\mathcal{S} = (e_{i_1}, \Delta_1(e_{i_1})), \ldots, (e_{i_j}, \Delta_j(e_{i_j})), \ldots$ of updates of edge weights in the underlying graph G, where the update $(e_{i_j}, \Delta_j(e_{i_j}))$ changes the current weight $wt(e_{i_j})$ of edge e_{i_j} to $wt(e_{i_j}) = wt(e_{i_j}) + \Delta_j(e_{i_j})$, assuming $wt(\cdot) = 0$ initially and $wt(\cdot) \ge 0$ for all updates. This model generalizes the dynamic graph streaming model in [7].

▶ **Theorem 19.** *Under the more generalized dynamic streaming model, any randomized streaming algorithm that, with probability at least* $2/3$*, approximates the maximum weighted* 1*-matching of the graph to a factor of* $6/5$ *uses space* $\Omega(\min\{m, \frac{(n-1)(n-2)}{2}\})$.

Proof. Given a data stream $\mathcal{S}' = x_1, x_2, \ldots, x_m$, where each $x_i \in \{1, \ldots, n'\}$, we define a graph stream $\mathcal{S}$ for a weighted graph G on n vertices, where n satisfies $(n-1)(n-2)/2 < n' \le n(n-1)/2$. Let $V = \{0 \ldots, n-1\}$ be the vertex-set of G. We first define a bijective function $\chi : \{(i,j) \mid i < j \in [n]^-\} \longrightarrow [\frac{n(n-1)}{2}]$. Let χ^{-1} be the inverse function of χ. Then, we can translate $\mathcal{S}'$ to a general dynamic graph streaming $\mathcal{S}$ of underlying weighted graph G by corresponding with x_i the i-th element $(\chi^{-1}(x_i), 1)$ of $\mathcal{S}$, for $i \in [m]$. Observe that computing F_∞ of $\mathcal{S}'$ is equivalent to computing a maximum weighted 1-matching for the graph stream $\mathcal{S}$ of G. Let uv be a maximum weighted 1-matching of $\mathcal{S}$, then $\chi(uv)$ is F_∞ of $\mathcal{S}'$. By Theorem 18, it follows that any randomized approximation streaming algorithm that approximates the maximum weighted 1-matching of G to a $\frac{6}{5}$-factor with probability at least $2/3$ uses space $\Omega(\{m, \frac{(n-1)(n-2)}{2}\})$, thus completing the proof. ◀

References

1 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Proceedings of the 31st ACM Symposium on Principles of Database Systems (PODS '12)*, pages 5–14, 2012.

2 KookJin Ahn, Graham Cormode, Sudipto Guha, Andrew McGregor, and Anthony Wirth. Correlation clustering in data streams. In *Proceedings of the 32nd International Conference on Machine Learning (ICML '15)*, pages 2237–2246, 2015.

3 J. Alman and H. Yu. Faster update time for turnstile streaming algorithms. In *Proceedings of the 31st annual ACM-SIAM symposium on Discrete algorithms (SODA '20)*, pages 1803–1813, 2020.

4 Sepehr Assadi, Sanjeev Khanna, and Yang Li. On estimating maximum matching size in graph streams. In *Proceedings of the 2017 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '17)*, pages 1723–1742, 2017.

5 Sepehr Assadi, Sanjeev Khanna, Yang Li, and Grigory Yaroslavtsev. Maximum matchings in dynamic graph streams and the simultaneous communication model. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '16)*, pages 1345–1364, 2016.

6 Jianer Chen, Ying Guo, and Qin Huang. Linear-time parameterized algorithms with limited local resources. *arXiv preprint*, 2020. `arXiv:2003.02866`.

7 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 *Proceedings of the 27th annual ACM-SIAM symposium on Discrete algorithms (SODA '16)*, pages 1326–1344, 2016.

8 Rajesh Chitnis, Graham Cormode, Hossein Esfandiari, MohammadTaghi Hajiaghayi, and Morteza Monemizadeh. New streaming algorithms for parameterized maximal matching and beyond. In *Proceedings of the 27th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '15)*, pages 56–58, 2015.

9 Rajesh Chitnis, Graham Cormode, Mohammad Taghi Hajiaghayi, and Morteza Monemizadeh. Parameterized streaming: Maximal matching and vertex cover. In *Proceedings of the 2015 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '15)*, pages 1234–1251, 2015.

10 Kenneth L. Clarkson and David P. Woodruff. Numerical linear algebra in the streaming model. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 205–214. ACM, 2009.

11 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

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

13 Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, USA, 2006.

14 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

15 Reinhard Diestel. *Graph Theory (Graduate Texts in Mathematics)*. Springer, 2005.

16 Stefan Fafianie and Stefan Kratsch. Streaming kernelization. In *Proceedings of the 39th International Symposium on Mathematical Foundations of Computer Science 2014 (MFCS '14)*, pages 275–286, 2014.

17 Harold N. Gabow. Data structures for weighted matching and nearest common ancestors with linking. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '90)*, pages 434–443. SIAM, 1990.

18 Harold N. Gabow. Data structures for weighted matching and extensions to b-matching and f-factors. *ACM Transactions on Algorithms*, 14(3), 2018.

19 Ashish Goel, Michael Kapralov, and Sanjeev Khanna. On the communication and streaming complexity of maximum bipartite matching. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '12)*, pages 468–485, 2012.

20 Ashish Goel, Michael Kapralov, and Ian Post. Single pass sparsification in the streaming model with edge deletions. *arXiv preprint*, 2012. `arXiv:1203.4900`.

21 Michael Kapralov. Better bounds for matchings in the streaming model. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '13)*, pages 1679–1697, 2013.

22 Michael Kapralov, Sanjeev Khanna, and Madhu Sudan. Approximating matching size from random streams. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '14)*, pages 734–751, 2014.

23 Michael Kapralov and David Woodruff. Spanners and sparsifiers in dynamic streams. In *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing (PODC '14)*, pages 272–281, 2014.

24 Christian Konrad and Adi Rosén. Approximating semi-matchings in streaming and in two-party communication. In *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming (ICALP '13)*, pages 637–649, 2013.

25 K. Larsen, J. Nelson, and H. Nguyen. Time lower bounds for nonadaptive turnstile streaming algorithms. In *Proceedings of the 47st Annual ACM Symposium on Theory of Computing STOC'15*, pages 803–812. ACM, 2015.

26 Roie Levin and David Wajc. Streaming submodular matching meets the primal-dual method. *arXiv preprint*, to appear in *SODA '21*, 2020. `arXiv:2008.10062`.

27 George B. Mertzios, André Nichterlein, and Rolf Niedermeier. A linear-time algorithm for maximum-cardinality matching on cocomparability graphs. *SIAM Journal on Discrete Mathematics*, 32(4):2820–2835, 2018.

28 George B. Mertzios, André Nichterlein, and Rolf Niedermeier. The power of linear-time data reduction for maximum matching. *Algorithmica*, 82(12):3521–3565, 2020.

29 Peter B. Miltersen, Noam Nisan, Shmuel Safra, and Avi Wigderson. On data structures and asymmetric communication complexity. *Journal of Computer and System Sciences*, 57(1):37–49, 1998.

30 Michael Mitzenmacher and Eli Upfal. *Probability and Computing*. Cambridge University Press, 2nd edition, 2017.

31 Ami Paz and Gregory Schwartzman. A $(2+\epsilon)$-approximation for maximum weight matching in the semi-streaming model. *ACM Transaction on Algorithms*, 15(2):18:1–18:15, 2019.

32 Tim Roughgarden. Communication complexity (for algorithm designers). *arXiv preprint*, 2015. `arXiv:1509.06257`.

33 Jeanette P. Schmidt, Alan Siegel, and Aravind Srinivasan. Chernoff-hoeffding bounds for applications with limited independence. *SIAM Journal on Discrete Mathematics*, 8(2):223–250, 1995.

34 M. Thorup and Y. Zhang. Tabulation-based 5-independent hashing with applications to linear probing and second moment estimation. *SIAM J. Comput.*, 41(2):293–331, 2012.

35 Salil P. Vadhan. Pseudorandomness. *Foundations and Trends in Theoretical Computer Science*, 7(1-3):1–336, 2012.