Kernelization via sampling with applications to finding matchings and related problems in dynamic graph streams
Chitnis, Rajesh; Cormode, Graham; Esfandiari, Hossein; Hajiaaghayi, MohammadTaghi; McGregor, Andrew; Monemizadeh, Morteza; Vorotnikova, Sofya

DOI:
10.1137/1.9781611974331.ch92

License:
None: All rights reserved

Document Version
Peer reviewed version

Citation for published version (Harvard):

Link to publication on Research at Birmingham portal

General rights
Unless a licence is specified above, all rights (including copyright and moral rights) in this document are retained by the authors and/or the copyright holders. The express permission of the copyright holder must be obtained for any use of this material other than for purposes permitted by law.

• Users may freely distribute the URL that is used to identify this publication.
• Users may download and/or print one copy of the publication from the University of Birmingham research portal for the purpose of private study or non-commercial research.
• Users may use extracts from the document in line with the concept of ‘fair dealing’ under the Copyright, Designs and Patents Act 1988 (?)
• Users may not further distribute the material nor use it for the purposes of commercial gain.

Where a licence is displayed above, please note the terms and conditions of the licence govern your use of this document.

When citing, please reference the published version.

Take down policy
While the University of Birmingham exercises care and attention in making items available there are rare occasions when an item has been uploaded in error or has been deemed to be commercially or otherwise sensitive.

If you believe that this is the case for this document, please contact UBJRA@lists.bham.ac.uk providing details and we will remove access to the work immediately and investigate.
Kernelization via Sampling with Applications to Finding Matchings and Related Problems in Dynamic Graph Streams

Rajesh Chitnis‡, Graham Cormode¶, Hossein Esfandiari†, MohammadTaghi Hajiaghayi‡, Andrew McGregor§, Morteza Monemizadeh¶, Sofya Vorotnikova§

Abstract
In this paper we present a simple but powerful subgraph sampling primitive that is applicable in a variety of computational models including dynamic graph streams (where the input graph is defined by a sequence of edge/hyperedge insertions and deletions) and distributed systems such as MapReduce. In the case of dynamic graph streams, we use this primitive to prove the following results:

- **Matching:** Our main result for matchings is that there exists an \(O(k^d)\) space algorithm that returns the edges of a maximum matching on the assumption the cardinality is at most \(k\). The best previous algorithm used \(O(kn)\) space where \(n\) is the number of vertices in the graph and we prove our result is optimal up to logarithmic factors. Our algorithm has \(O(1)\) update time. We also show that there exists an \(O(n^2/\alpha^3)\) space algorithm that returns an \(\alpha\)-approximation for matchings of arbitrary size. In independent work, Assadi et al. (SODA 2016) proved this approximation algorithm is optimal and provided an alternative algorithm. We generalize our exact and approximate algorithms to weighted matching. For graphs with low arboricity such as planar graphs, the space required for constant approximation can be further reduced. While there has been a substantial amount of work on approximate matching in insert-only graph streams, these are the first non-trivial results in the dynamic setting.

- **Vertex Cover and Hitting Set:** There exists an \(O(k^d)\) space algorithm that solves the minimum hitting set problem where \(d\) is the cardinality of the input sets and \(k\) is an upper bound on the size of the minimum hitting set. We prove this is optimal up to logarithmic factors. Our algorithm has \(O(1)\) update time. The case \(d = 2\) corresponds to minimum vertex cover.

Finally, we consider a larger family of parameterized problems (including \(b\)-matching, disjoint paths, vertex coloring among others) for which our subgraph sampling primitive yields fast, small-space dynamic graph stream algorithms. We then show lower bounds for natural problems outside this family.

1 Introduction
Over the last decade, a growing body of work has considered solving graph problems in the data stream model. Most of the early work considered the insert-only variant of the model where the stream consists of edges being added to the graph and the goal is to compute properties of the graph using limited memory. Recently, however, there has been a significant amount of interest in being able to process dynamic graph streams where edges are both added and deleted from the graph \([3,6,8,10,27,28,33,34,40]\). These algorithms are all based on the surprising efficacy of using random linear projections, aka linear sketching, for solving combinatorial problems. Results include testing edge connectivity \([6]\) and vertex connectivity \([28]\), constructing sparsifiers \([7,8,33]\), approximating the densest subgraph \([10,20,43]\), correlation clustering \([3]\), and estimating the number of triangles \([40]\).

For a recent survey of the area, see \([42]\).

The concept of **parameterized stream algorithms** was explored by Chitnis et al. \([13]\) and Fafianie and Kratsch \([22]\). Their work investigated a natural connection between data streams and parameterized complexity. In parameterized complexity, the time cost of a problem is analyzed in terms of not only the input size but also other parameters of the input. For example, while the classic vertex cover problem is NP complete, it can be solved via a simple branching algorithm in time \(2^k \cdot \text{poly}(n)\) where \(k\) is the size of the optimal vertex cover. An important concept in parameterized complexity is **kernelization** in which the goal is to efficiently transform an instance of a problem into a smaller instance such that the smaller instance is a “yes” instance (e.g., has a solution...
of at least a certain size) iff the original instance was also a “yes” instance. For more background on parameterized complexity and kernelization, see [16,24]. Parameterizing the space complexity of a problem in terms of the size of the output is a particularly appealing notion in the context of data stream computation. In particular, the space used by any algorithm that returns an actual solution (as opposed to an estimate of the size of the solution) is necessarily at least the size of the solution.

Our Results and Related Work. In this paper we present a simple but powerful subgraph sampling primitive that is applicable in a variety of computational models including dynamic graph streams (where the input graph is defined by a sequence of edge/hyperedge insertions and deletions) and distributed systems such as MapReduce. This primitive will be useful for both those parameterized problems whose output has bounded size and for those where the optimal solution need not be bound. In the case where the output has bounded size, our results can be thought of as kernelization via sampling, i.e., we sample a relatively small set of edges according to a simple (but not uniform) sampling procedure and can show that the resulting graph has a solution of size at most \(k\) iff the original graph has an optimal solution of size at most \(k\). We present the subgraph sampling primitive and implementation details in Section 2.

Graph Matchings. Finding a large matching is the most well-studied graph problem in the data stream model [4,5,15,18,23,26,31,32,37,38,41,48]. However, all of the existing single-pass stream algorithms are restricted to the insert-only case, i.e., edges may be inserted but will never be deleted. This restriction is significant: for example, the simple greedy algorithm using \(\tilde{O}(n)\) space returns a 2-approximation if there are no deletions. In contrast, prior to this paper no \(o(n)\)-approximation was known in the dynamic case when there are both insertions and deletions. Finding an algorithm for the dynamic case of this fundamental graph problem was posed as an open problem in the Bertinoro Data Streams Open Problem List [1, Problem 64].

We prove the following results for computing a matching in the dynamic model. Our first result is an \(\tilde{O}(k^2)\) space algorithm that returns the edges of a maximum matching on the assumption that its cardinality is at most \(k\). Our algorithm has \(\tilde{O}(1)\) update time. The best previous algorithm [13] collects \(\min(\deg(u), 2k)\) edges incident to each vertex \(u\) and finds the optimal matching amongst these edges. This algorithm can be implemented in \(\tilde{O}(kn)\) space where \(n\) is the number of vertices in the graph. Indeed obtaining an algorithm with \(f(k)\) space, for any function \(f\), in the dynamic graph stream case was left as an important open problem [13]. We can also extend our approach to maximum weighted matching. Our second result is an optimal \(\tilde{O}(n^2/\alpha^2)\) space algorithm that returns an \(\alpha\)-approximation for matchings of arbitrary size.

For example, this implies an \(n^{1/3}\) approximation using \(\tilde{O}(n)\) space, commonly known as the semi-streaming space restriction [23,44]. We present our second result and an algorithm for graphs with bounded arboricity, along with a discussion of very recent related work [9,11,36], in Section 4.

Vertex Cover and Hitting Set. We next consider the problem of finding the minimum vertex cover and its generalization, minimum hitting set. The hitting set problem can be defined in terms of hypergraphs: given a set of hyperedges, select the minimum set of vertices such that every hyperedge contains at least one of the selected vertices. If all hyperedges have cardinality two, this is the vertex cover problem.

There is a growing body of work analyzing hypergraphs in the data stream model [17,28,35,45,47]. For example, Emek and Rosén [17] studied the following set-cover problem which is closely related to the hitting set problem: given a stream of hyperedges (without deletions), find the minimum subset of these hyperedges such that every vertex is included in at least one of the hyperedges. They present an \(O(\sqrt{n})\) approximation streaming algorithm using \(\tilde{O}(n)\) space along with results for covering all but a small fraction of the vertices. Another related problem is independent set since the minimum vertex cover is the complement of the maximum independent set. Halldórsson et al. [29] presented streaming algorithms for finding large independent sets but these do not imply a result for vertex cover in either the insert-only or dynamic setting.

In Section 3.2 we present a \(\tilde{O}(k^d)\) space algorithm that finds the minimum hitting set where \(d\) is the cardinality of the input sets and \(k\) is an upper bound on the cardinality of the minimum hitting set. We prove the space use is optimal and matches the space used by previous algorithms in the insert-only model [13,22]. Our algorithms can be implemented with \(\tilde{O}(1)\) update time. The only previous results in the dynamic model were by Chitnis et al. [13] and included a \(\tilde{O}(kn)\) space algorithm for the vertex cover problem. They also provide a \(\tilde{O}(k^2)\) space algorithm under a much stronger “promise” that the vertex cover of the graph defined by any prefix of the stream may never exceed \(k\). Relaxing this promise remained as the main open problem of Chitnis et al. [13]. In Section 3.2 we also generalize our exact matching result to hypergraphs. In Section 6 we show our result is also optimal.

General Family of Results. We consider a larger family of parameterized problems for which our subgraph sampling primitive yields fast, small-space dynamic graph stream algorithms. This result is presented in Section 5, while lower bounds for various problems outside this family are proved in Section 6.

2 Basic Subgraph Sampling Technique

Basic Approach and Intuition. The inspiration for our subgraph sampling primitive is the following simple procedure
for edge sampling. Given a graph $G = (V, E)$ and probability $p \in [0, 1]$, let $\mu_{G,p}$ be the distribution over $E \cup \{\perp\}$ defined by the following process:

1. Sample each vertex independently with probability $p$ and let $V'$ denote the set of sampled vertices.
2. Return an edge chosen uniformly at random from the edges in the induced graph on $V'$. If no such edge exists, return $\perp$.

The distribution $\mu_{G,p}$ has some surprisingly useful properties. For example, suppose that the optimal matching in a graph $G$ has size at most $k$. It is possible to show that this matching has the same size as the optimal matching in the graph formed by taking $O(k^2)$ independent samples from $\mu_{G,1/k}$. It is not hard to show that such a result would not hold if the edges were sampled uniformly at random.

The intuition is that when we sample from $\mu_{G,p}$ we are less likely to sample an edge incident to a high degree vertex than if we sampled uniformly at random from the edge set. For a large family of problems including matching, it will be advantageous to avoid bias towards edges whose endpoints have high degree.

Our subgraph sampling primitive essentially parallelizes the process of sampling from $\mu_{G,p}$. This will lead to more efficient algorithms in the dynamic graph stream model. The basic idea is rather than select a subset of vertices $V'$, we randomly partition $V$ into $V_1 \cup V_2 \cup \ldots \cup V_{1/p}$. Selecting a random edge from the induced graph on any $V_i$ results in an edge distributed as in $\mu_{G,p}$. Sampling an edge on each $V_i$ results in $1/p$ samples from $\mu_{G,p}$ although note that the samples are no longer independent. This lack of independence will not be an issue and will sometimes be to our advantage.

In many applications it will make sense to parallelize the sampling further and select a random edge between each pair, $V_i$ and $V_j$, of vertex subsets. For applications involving hypergraphs we select random edges between larger subsets of $\{V_i, V_2, \ldots, V_{1/p}\}$.

**Sampling Data Structure:** We now present the subgraph sampling primitive formally. Given an unweighted (hyper)graph $G = (V, E)$, consider a “coloring” defined by a function $c : V \rightarrow [b]$. It will be convenient to introduce the notation: for each $i \in [b]$

$$V_i = \{v \in V : c(v) = i\}$$

and say that every vertex in $V_i$ has color $i$. For a (hyper)edge $e \in E$, we define $c(e) = \{c(v) : v \in e\}$, i.e., $c(e)$ is exactly the set of colors seen on the vertices of $e$. For $S \subseteq [b]$, we say that an (hyper)edge $e$ of $G$ is $S$-colored if $c(e) = S$, i.e., each color from $S$ is used to color the vertices in $e$ and no other colors are used. Given a constant $q \geq 1$ which denotes the “size restriction”, for each $S \subseteq [b]$ of cardinality at most $q$, $E_S$ contains a single edge chosen uniformly at random from the set of all $S$-colored edges. If there are no $S$-colored edges, then $E_S = \emptyset$. The union of these sets defines the random graph $G' = (V, E')$, i.e.,

$$E' = \bigcup_{S \subseteq [b], |S| \leq q} E_S.$$

For example, given a simple graph, if we have $q = 1$ then for each color $i \in [b]$ we choose an edge whose endpoints are both colored $i$. If $q = 2$, then for every $1 \leq i < j \leq b$ we choose an edge whose one endpoint has color $i$ and the other endpoint has color $j$: note that this includes the possibility that $i = j$. In the case of a weighted graph, for each distinct weight $w$ we choose a single edge $E_{S,w}$ uniformly at random from the set of $S$-colored edges with weight $w$.

**Definition 2.1.** We define $\text{Sample}_{b,q,1}$ to be the distribution over subgraphs generated as above where $c$ is chosen uniformly at random from a family of pairwise independent hash functions. $\text{Sample}_{b,q,r}$ is the distribution over graphs formed by taking the union of $r$ independent graphs sampled from $\text{Sample}_{b,q,1}$. Algorithm 1 gives pseudocode for sampling from $\text{Sample}_{b,q,r}$.

**Motivating Application.** As a first application to motivate the subgraph sampling primitive we again consider the problem of estimating matchings. We will use the following simple lemma that will also be useful in subsequent sections.

**Lemma 2.1.** Let $U \subseteq V$ be an arbitrary subset of $|U| = r$ vertices and let $c : V \rightarrow [4r^{-1}]$ be a pairwise independent hash function. Then with probability at least $3/4$, at least $(1 - \epsilon)r$ of the vertices in $U$ are hashed to distinct values. Setting $\epsilon < 1/r$ ensures all vertices are hashed to distinct values with this probability.

**Proof.** Let $b = 4r^{-1}$. For a vertex $u \in U$, let $I_u$ be the indicator random variable that equals one if there exists $u' \in U \setminus \{u\}$ such that $c(u) = c(u')$. Since $c$ is pairwise independent,

$$\Pr[I_u] \leq \sum_{u' \in U \setminus \{u\}} \Pr[c(u) = c(u')] = \sum_{u' \in U \setminus \{u\}} 1/b < r/b = \epsilon/4.$$

Let $I = \sum_{u \in U} I_u$ and note that $\mathbb{E}[I] \leq \epsilon r/4$. Then Markov’s inequality implies $\Pr[I \geq \epsilon r] \leq 1/4$. □
We now describe how the subgraph sampling primitive can be used to sample a subgraph \( G' = (V, E') \) from \( G = (V, E) \) in \( O(\log |V|) \) space. The sampling is performed by an \( \ell_0 \)-sampler algorithm. Given a dynamic graph stream, the behavior of an \( \ell_0 \)-sampler algorithm is defined as follows: It returns FAIL with probability at most \( \delta \) and otherwise, it returns an edge chosen uniformly at random amongst the edges that have been inserted and not deleted. If there are no such edges, the \( \ell_0 \)-sampler returns NULL. The \( \ell_0 \)-sampling primitive can be implemented using \( O(\log^2 n \log \delta^{-1}) \) bits of space and \( O(\text{polylog} n) \) update time. In some cases, we can make use of simpler deterministic data structures. For Theorem 3.1, we can replace the \( \ell_0 \) sampler with a counter and the exclusive-or of all the edge identifiers, since we only require to recover edges when they are unique within their color class. For Theorem 3.1, we only require a counter. In both cases, the space cost is reduced to \( O(\log n) \).

At the start of the stream we choose a pairwise independent hash function \( c : V \rightarrow [b] \). For each weight \( w \) and subset \( S \subseteq [b] \) of size \( q \), this hash function defines a sub-stream corresponding to the \( S \)-colored edges of weight \( w \). We then use \( \ell_0 \)-sampling on each sub-stream to select a random edge to be used as \( E_S \).

**MapReduce and Distributed Models.** The sampling distribution is naturally parallel, making it straightforward to implement in a variety of popular models. In MapReduce, the \( r \) hash functions can be shared state among all machines, allowing Map function to output each edge keyed by its color under each hash function. Then, these can be sampled from on the Reduce side to generate the graph \( G' \). Optimizations can do some data reduction on the Map side, so that only one edge per color class is emitted, reducing the communication cost. A similar outline holds for other parallel graph models such as Pregel.

### 3 Parameterized Matching, Vertex Cover, and Hitting Set

#### 3.1 Finding Maximum Matchings and Minimum Vertex Covers Exactly

In this section, we present results on finding edges in a maximum matching and the vertices in a minimum vertex covering. We describe an algorithm for sampling subgraphs according to \( \text{Sample}_{b,q,r} \).

**Algorithm 1.** Algorithm for Sampling Subgraphs According to \( \text{Sample}_{b,q,r} \).

**Input:** A (hyper)graph \( G = (V, E) \) and natural numbers \( b, q, r \).

**Output:** A subgraph \( G' = (V, E') \) where \( E' \subseteq E \).

1. Choose \( c_1, \ldots, c_r \) u.a.r. from a family of pairwise independent hash functions mapping \( V \) to \([b]\).
2. Set \( E' = \emptyset \).
3. for \( 1 \leq j \leq r \) do
   4. for each \( S \subseteq [b] \) such that \(|S| \leq q\) do
      5. Select an edge \( E_E^j \) u.a.r. from the set of \( S \)-colored edges \( \{ e \in E : \cup_{v \in e} c_j(v) = S \} \) if this set is non-empty.
      6. Otherwise let \( E_E^j = \emptyset \).
   7. \( E' \leftarrow E' \cup E_E^j \).
8. Report the graph \( G' = (V, E') \).

Suppose \( G \) is a graph with a matching \( M = \{ e_1, \ldots, e_k \} \) of size \( k \). Let \( G' \sim \text{Sample}_{b,2,1} \). By the above lemma, there exists \( b = O(k^2) \), such that all the \( 2k \) endpoints of edges in \( M \) are colored differently with constant probability. Suppose the endpoints of edge \( e_i \) received the colors \( a_i \) and \( b_i \). Then \( G' \) contains an edge in \( E_{\{a_i, b_i\}} \) for each \( i \in [k] \). Assuming all endpoints receive different colors, no edge in \( E_{\{a_i, b_i\}} \) shares an endpoint with an edge in \( E_{\{a_j, b_j\}} \) for \( j \neq i \). Hence, we can conclude that \( G' \) also has a matching of size \( k \). In Section 2.3 we show that a similar approach can be generalized to a range of problems. Using a similar argument, there exists \( b = O(k) \) such that \( G' \) contains a constant approximation to the optimum matching. However, in Section 2.3 we show that there exists \( b = O(k) \) such that with high probability graphs sampled from \( \text{Sample}_{b,2,O(\log k)} \) preserve the size of the optimal matching exactly.

#### 2.1 Application to Data Streams and MapReduce

We now describe how the subgraph sampling primitive can be implemented in various computational models.

**Dynamic Graph Streams.** Let \( S \) be a stream of insertions and deletions of edges of an underlying graph \( G(V, E) \). We assume that vertex set \( V = \{1, 2, \ldots, n\} \). We assume that the length of the stream is polynomially related to \( n \) and hence \( O(\log |S|) = O(\log n) \). We denote an undirected edge in \( E \) with two endpoints \( u, v \in V \) by \( uv \). For weighted graphs, we assume that the weight of an edge is specified when the edge is inserted and deleted and that the weight never changes. The following theorem establishes that the sampling primitive can be efficiently implemented in dynamic graph streams.

**Theorem 2.1.** Suppose \( G \) is a graph with \( w_0 \) distinct weights. It is possible to sample from \( \text{Sample}_{b,q,r} \) with probability at least \( 1 - \delta \) in the dynamic graph stream model using \( O(b^2 r w_0) \) space and \( O(r) \) update time.

**Proof.** To sample a graph from \( \text{Sample}_{b,q,r} \) we simply sample \( r \) graphs from \( \text{Sample}_{b,q,1} \) in parallel. To draw a sample from \( \text{Sample}_{b,q,1} \), we employ one instance of an \( \ell_0 \)-sampling primitive for each of the \( O(b^2) \) edge colorings [14].
cover of a graph $G$. We use $\text{match}(G)$ to denote the size of the maximum (weighted or unweighted as appropriate) matching in $G$ and use $\text{vc}(G)$ to denote the size of minimum vertex cover. The main theorem we prove in this section is that a maximum matching (or minimum vertex cover) in the sampled graph is also a maximum matching (or minimum vertex cover) in the original graph.

**Theorem 3.1.** *(Finding Exact Solutions)* Suppose $\text{match}(G) \leq k$. Then, with probability $1 - 1/\text{poly}(k)$,

$$\text{match}(G') = \text{match}(G) \text{ and } \text{vc}(G') = \text{vc}(G),$$

where $G' = (V, E') \sim \text{Sample}_{1000k, 2, \Theta(\log k)}$.

**Intuition and Preliminaries.** To argue that $G'$ has a matching of the optimal size, it suffices to show that for every edge $uv \in G$ that is not in $G'$, there is a large number of edges incident to one or both of $u$ and $v$ that are in $G'$. If this is the case, then it will still be possible to match at least one of these vertices in $G'$.

To make this precise, let $U$ be the subset of vertices with degree at least $10k$. Let $F$ be the set of edges in the induced subgraph on $V \setminus U$, i.e., the set of edges whose endpoints both have small degree. We will prove that with high probability,

$$(3.1) \quad (F \subseteq E') \text{ and } (\forall u \in U, \text{deg}_{G'}(u) \geq 5k),$$

where $E'$ is the set of edges in $G'$. Note that any sampled graph $G'$ that satisfies $(3.1)$ has the property that for all edges $uv \in G$ that are not in $G'$ we have $\text{deg}_{G'}(u) \geq 5k$ or $\text{deg}_{G'}(v) \geq 5k$.

**Analysis.** The first lemma establishes that it is sufficient to prove that $(3.1)$ holds with high probability.

**Lemma 3.1.** If $\text{match}(G) \leq k$ then $(3.1)$ implies $\text{match}(G') = \text{match}(G)$ and $\text{vc}(G') = \text{vc}(G)$.

**Proof.** We first argue that $\text{vc}(G') = \text{vc}(G)$. Since the vertex cover of $G$ is of size at most $2k$, every vertex in $U$ must be in the vertex cover of both $G$ and $G'$ since the degrees of such vertices in both graphs are strictly greater than $2k$. This follows because if a vertex in $U$ was not in the minimum vertex cover then all its neighbors need to be in the vertex cover. To complete the vertex cover requires consideration of only those edges not incident on $U$. This is exactly the set of edges $F$, which by the assumption are present in $G'$, leading to the same vertex cover.

We next argue that $\text{match}(G') = \text{match}(G)$. If property $(3.1)$ is satisfied then $G'$ contains a matching of size $\text{match}(F) + |U| \geq \text{match}(G)$ since we may choose the optimum matching in $F$ and then still be able to match every vertex in $U$. This follows because the optimum matching in $F$ “consumes” at most $2k$ potential endpoints, since $\text{match}(G) \leq k$. Hence, each of the (at most $2k$) vertices in $U$ can still be matched to $3k$ possible vertices.

The next lemma establishes that $(3.1)$ holds with the required probability.

**Lemma 3.2.** Property $(3.1)$ holds with probability at least $1 - 1/\text{poly}(k)$.

**Proof.** Let $\text{VC}(G)$ be a minimum vertex cover of $G$. Recall that $\text{match}(G) \leq k$ implies that $\text{vc}(G) = |\text{VC}(G)| \leq 2k$ because the endpoints of the edges in a maximum matching form a vertex cover. Next consider $H \sim \text{Sample}_{1000k, 2, \Theta(\log k)}$. We will show that for any $e \in F$ and $u \in U$,

$$\mathbb{P}[e \in H] > 1/2 \quad \text{and} \quad \mathbb{P}[\text{deg}_H(u) \geq 5k] \geq 1/2.$$
every \{c_i, c(u)\}-colored edge is incident to \(u\) and is distinct from every \{c_j, c(u)\}-colored edge.

Observe that \(P(c(u) \in c(A)) \leq 2k/b\). By appealing to Lemma 2.1 with probability at least 3/4, there are at least \(6k\) colors used to color the vertices \(N_u\). Of these colors, at least \(5k\) are colored differently from vertices in \(A\). Hence we find \(5k\) edges incident to \(u\) with probability at least \(3/4 - 2k/b \geq 1/2\). □

**Extension to Weighted Matching.** We now extend the result of the previous section to the weighted case. The following lemma shows that it is possible to remove an edge \(uv\) from a graph without changing the weight of the maximum weighted matching, if \(u\) and \(v\) satisfy certain properties.

**Lemma 3.3.** Let \(G = (V, E)\) be a weighted graph and let \(G' = (V, E')\) be a subgraph with the property:

\[
\forall uv \in E \setminus E', \deg^w_G(u) \geq 5k \text{ or } \deg^w_G(v) \geq 5k,
\]

where \(\deg^w_G(u)\) is the number of edges incident to \(u\) in \(G\) with weight \(w\). Then, \(match(G) = match(G')\).

**Proof.** Let \(E \setminus E' = \{e_1, e_2, \ldots, e_t\}\) and let \(G_i\) be the graph formed by removing \(\{e_1, \ldots, e_t\}\) from \(G\). So \(G_0' = G\) and \(G_i = G_i'\). For the sake of contradiction, suppose \(\text{match}(G) > \text{match}(G')\) and let \(r\) be the minimal value such that \(\text{match}(G) > \text{match}(G_r')\).

By the minimality of \(r\), \(\text{match}(G) = \text{match}(G_{r-1}')\). Consider the maximum weight matching \(M\) in \(G_0'\). If \(e_r \notin M\) then \(\text{match}(G) = \text{match}(G_{r-1}') = \text{match}(G_r')\) and we have a contradiction. If \(e_r \in M\), let \(u, v\) be the endpoints of \(e_r\) and the weight of \(e_r\) be \(w\). Without loss of generality \(\deg_{G_r'}^w(u) \geq \deg_{G_r'}^w(v) \geq 5k\). Hence, there exists edge \(ux\) of weight \(w\) in \(G_r'\) where \(x\) is not an endpoint in \(M\). Therefore, the matching \(\{M \setminus \{e_r\}\} \cup \{ux\}\) contains \(G_r'\) and has the same weight as \(M\). Hence, \(\text{match}(G) = \text{match}(G_{r-1}') = \text{match}(G'_r)\) and we again have a contradiction. □

Consider a weighted graph \(G\) and let \(G' \sim \text{Sample}_{100k, 2.5b \log k}\). For each weight \(w\), let \(G_w\) and \(G'_w\) denote the subgraphs consisting of edges with weight exactly \(w\). By applying the analysis of the previous section to \(G_w\) and \(G'_w\), we may conclude that \(G'\) satisfies the properties of the above lemma. Hence, \(\text{match}(G) = \text{match}(G')\). To reduce the dependence on the number of distinct weights in Theorem 2.1, we may first round each weight to the nearest power of \((1+\epsilon)\) at the cost of incurring a \((1+\epsilon)\) factor error. If \(W\) is the ratio of the max weight to min weight, there are \(O(\epsilon^{-1} \log W)\) distinct weights after the rounding.

### 3.2 Finding Minimum Hitting Set Exactly

In this section, we present exact results for computing hitting sets and hypergraph matchings. Throughout the section, let \(G\) be a hypergraph with \(\text{hs}(G) \leq k\) where \(\text{hs}(G)\) denotes the cardinality of the minimum hitting set of \(G\). We assume that each edge has size exactly \(d\) where \(d\) is a constant.

**Intuition and Preliminaries.** Given that the hitting set problem is a generalization of the vertex cover problem, naturally some of the ideas in this section build upon ideas from the previous section. However, the combinatorial structure we need to analyze for our sampling result goes beyond what is typically needed when extending vertex cover kernelization results to hitting sets. We first need to review a basic definition and result about “sunflower” set systems.

**Lemma 3.4.** (Sunflower Lemma \[19\]) Let \(F\) be a collection of subsets of \([n]\). Then \(A_1, \ldots, A_s \in F\) is an \(s\)-sunflower if \(A_i \cap A_j = \emptyset\) for all \(1 \leq i < j \leq s\). We refer to \(C\) as the core of the sunflower and \(A_1 \setminus C\) as the petals. If each set in \(F\) has size at most \(d\) and \(|F| > d!k^d\), then \(F\) contains a \((k+1)\)-sunflower.

Let \(s_G(C)\) denote the number of petals in a maximum sunflower in the graph \(G\) with core \(C\). We say a core is large if \(s_G(C) > ak\) for some large constant \(a\) and significant if \(s_G(C) > k\). Define the sets:

- \(U = \{C \subseteq V \mid s_G(C) > ak\}\) is the set of large cores.
- \(F = \{D \subseteq E \mid \forall C \subseteq U, C \not\subseteq D\}\) is the set of edges that do not include a large core.
- \(U' = \{C \subseteq U \mid s_G(C) \leq k\}\) is the set of large cores that do not contain significant cores.

**Lemma 3.5.** \(|F| = O(k^d)\) and \(|U'| = O(k^{d-1})\)

**Proof.** For the sake of contradiction assume \(|F| > d!(ak)^d\). Then, by the Sunflower Lemma, \(F\) contains a \((ak+1)\)-sunflower. If the core of this sunflower is empty, \(F\) has a matching of size \((ak+1)\) and therefore cannot have a hitting set of size at most \(k\). If the sunflower has a non-empty core \(C\), then some edge \(D \subseteq E\) contains \(C\), which contradicts the definition of \(F\). Therefore, \(|F| \leq d!(ak)^d\).

To prove \(|U'| \leq (d-1)!k^{d-1}\), first note that \(|C'| \leq d-1\) for all \(C' \in U'\). For the sake of contradiction assume that \(|U'| > (d-1)!k^{d-1}\). Then, by the Sunflower Lemma again, \(U'\) contains a \((k+1)\)-sunflower. Note that it is a sunflower of cores, not hypergraph edges. Let \(C_1, C_2, \ldots, C_{k+1}\) be the sets in the sunflower. Each of these sets has to contain at least one vertex of the minimum hitting set. Therefore, if \(C_1, C_2, \ldots, C_{k+1}\) are disjoint (i.e., the core of the sunflower is empty), \(U'\) has a matching of size \((k+1)\) and cannot have a hitting set of size at most \(k\). If the sunflower has a non-empty core \(C^*\), we will show that union of the maximum sunflowers with cores \(C_1, C_2, \ldots, C_{k+1}\) contains a sunflower with \(k+1\) edges with core \(C^* \subseteq C_1 \in U'\). This contradicts the definition of \(U'\) and therefore \(|U'| \leq (d-1)!k^{d-1}\).
Proof. For each significant core $C$ there has to be at least one vertex from the hitting set in $C$. Since all large cores are significant, $\text{hs}(G) = \text{hs}(U \cup F)$. If $C \in U$ has a subset $C'$ such that $\text{s}_G(C') > k$, then there is at least one vertex from the hitting set in $C'$ and this vertex also hits $C$. Thus, $\text{hs}(G) = \text{hs}(U \cup F)$. By Lemma 3.7, the set of significant cores in $G'$ is a superset of $U'$ with high probability. By Lemma 3.8, every edge in $F$ is in $G'$ with high probability. \( \square \)

Lemma 3.7. \( \mathbb{P}[\forall C \in U', \text{s}_G(C) > k] \geq 1 - 1/\text{poly}(k) \).

Proof. Fix an arbitrary core $C \in U'$. Consider $H \sim \text{Sample}_{b,d}$. And let $c : [n] \to [b]$ be the coloring that defined $H$. We need to identify $k + 1$ sets of colors $S_1, S_2, \ldots, S_{k+1} \subset [b]$ each of size $d$, such that any set of edges $D_1, D_2, \ldots, D_{k+1}$ where $D_i$ is $S_i$-colored forms a sunflower of size $k + 1$ on core $C$. In order for this to hold, the color sets have to satisfy the following three properties:

1. All edges that are $S_i$-colored contain $C$.
2. There is at least one $S_i$-colored edge.
3. If $D$ is $S_i$-colored and $D'$ is $S_j$-colored then $(D \setminus C) \cap (D' \setminus C) = \emptyset$.

In what follows, we first define a set $F = \{S_1, S_2, \ldots\}$ that satisfies the above properties. We then argue that $|F| \geq k + 1$ with probability at least $1/2$. By repeating the process $O(\log k)$ times will ensure that such a family exists with high probability. The lemma follows by taking the union bound over all $C \in U'$ since $|U'| = O(k^{d-1})$ by Lemma 3.5.

Property 1. We first define a set of vertices $A$ such that any edge that does not intersect $A$ must include $C$. Then, for any $S \subset [b]$ that is disjoint from $c(A)$, we may infer that all $S$-colored edges contain $C$. This follows since if $S = c(D)$ for some edge $D$, then $c(D) \cap c(A) = \emptyset$ which implies that $D \cap A = \emptyset$, and so $C \subseteq D$. Let

$$A = (\text{HS}(G) \setminus C) \cup \left( \bigcup_{C' \subset C} \text{HS}(M_{C', C}) \right).$$

All edges that do not intersect $\text{HS}(G) \setminus C$ must intersect with $C$. But all edges that intersect with only a subset of $C$, say $C'$, must intersect with $\text{HS}(M_{C', C})$. Hence $A$ has the claimed property. We will say that $C$ is a good core if $c(C) \cap c(A) = \emptyset$ and $|c(C)| = |C|$. 

Hitting Set. For the rest of this section we let $G' = (V, E') \sim \text{Sample}_{b,d,1}(G)$ where $b = O(k)$, $d$ is the cardinality of the hyperedges, and $r = O(\log k)$. It will also be convenient to use the notation $\text{HS}(S)$ to denote a minimum hitting set of a collection of sets $S$, i.e., $\text{hs}(S) = |\text{HS}(S)|$.
Property 2. Next, let $\mathcal{P}$ be a set of petals in a sunflower with core $C$ that do not intersect with $A$. We may choose a set of $|\mathcal{P}| = ak - |A|$ such petals. For each $P \in \mathcal{P}$, define the set:

$$A_P = A \cup C \cup \left( \bigcup_{Q \in \mathcal{P}\setminus P} Q \right).$$

If $C$ is a good core, let $\mathcal{P}'$ contain all $P \in \mathcal{P}$ such that $c(P) \cap c(A_P) = \emptyset$ and $|c(P)| = |P|$. If $C$ is not a good core, let $\mathcal{P}' = \emptyset$. Then the family $\mathcal{F} = \{c(P \cup C)\}_{P \in \mathcal{P}'}$ satisfies Properties 1 and 2. Note that no two petals in $\mathcal{P}'$ share the same color and hence $|\mathcal{F}| = |\mathcal{P}'|$ assuming $C$ is a good core.

Property 3. Assume $C$ is a good core since otherwise $\mathcal{F} = \emptyset$ and Property 3 is trivially satisfied. Let $S_1, S_2 \in \mathcal{F}$ and suppose $S_1 = c(C \cup P_1)$ and $S_2 = c(C \cup P_2)$ for some $P_1, P_2 \in \mathcal{P}'$. Suppose edges $C \cap Q_1$ and $C \cup Q_2$ are $S_1$-colored and $S_2$-colored respectively. Then $c(Q_1) = c(P_1)$ and $c(Q_2) = c(P_2)$ because $|c(C)| = |C|$, $|c(P_1)| = |P_1|$, $|c(P_2)| = |P_2|$, and all edges have the same cardinality. But $c(P_1) \cap c(P_2) = \emptyset$ implies $c(Q_1) \cap c(Q_2) = \emptyset$ and so $Q_1 \cap Q_2 = \emptyset$ as required.

Size of $\mathcal{F}$. We need to show that $|\mathcal{P}'| \geq (k + 1)^2$ with probability $1/2$. Recall that $c : V \to [b]$ is chosen randomly from a family of pairwise independent hash functions and suppose $b = 8\max(d|A| + d^2, d|A_P| + d^2)$. Note that $b = O(k)$ since, by appealing to Lemma 3.6,

$$|A| \leq |A_P| \leq |A| + |C| + d|P| \leq \hs(G) + \sum_{C' \subset C} \hs(M_{C,C'}) + |C| + d|P| \leq k + 2^d dk + d + dak = O(k).$$

Then,

$$\mathbb{P}[C \text{ is not a good core}] = \mathbb{P}[c(C) \cap c(A) \neq \emptyset \text{ or } |c(C)| \neq |C|] \leq (d|A| + d^2)/b \leq 1/8.$$

For each $P \in \mathcal{P}$, let $X_P = 1$ if $P \notin \mathcal{P}'$ or $C$ is not a good core. Let $X_P = 0$ otherwise. Then

$$\mathbb{E} \left[ \sum_{P \in \mathcal{P}} X_P \right] \leq |\mathcal{P}| \left( d|A_P| + d^2/b + 1/8 \right) \leq |\mathcal{P}|/4,$$

and so

$$\mathbb{P} \left[ \sum_{P \in \mathcal{P}} X_P \geq |\mathcal{P}|/2 \right] \leq 1/2$$

by the Markov inequality. Hence,

$$|\mathcal{P}'| = |\mathcal{P}| - \sum_{P \in \mathcal{P}} X_P \geq |\mathcal{P}|/2 = ak/2 - |A|/2 \geq k + 1$$

for sufficiently large $a$ with probability at least $1/2$.

**Lemma 3.8.** $\mathbb{P}[F \subseteq E'] \geq 1 - 1/\text{poly}(k)$.

**Proof.** Pick an arbitrary edge $D \in F$. Consider $H \sim \text{Sample}_{b,d,1}$ and let $c : [n] \to [b]$ be the coloring that defined $H$. It suffices to show that there is a unique edge that is $c(D)$-colored since then $D$ is necessarily an edge in $H$. It suffices to show that this is the case with probability at least $1/2$ because repeating the process $O(\log k)$ times will ensure that such a family exists with high probability. The result then follows by taking the union bound over all $D \in F$ since $|F| = O(k^d)$ by Lemma 3.5.

Let $S = c(D)$. We first define a set $A$ of vertices such that the only edge that is disjoint from $A$ is $D$. It follows that $D$ is the unique $S$-colored edge if $S \cap c(A) = \emptyset$, since every other edge intersects $A$ and hence must share a color with it.

We define $A$ as follows:

$$A = (HS(G) \setminus D) \cup \left( \bigcup_{C \subset D} HS(M_{C,D}) \right).$$

Note $D$ itself is disjoint from $A$ since each $HS(M_{C,D})$ does not include vertices from $D$. If an edge is disjoint from $(HS(G) \setminus D)$ then it must intersect $D$. Suppose there exists an edge $D'$ such that $D \cap D' = C \neq D$, then $D'$ intersects $HS(M_{C,D})$. Hence, the only edge that is disjoint from $A$ includes the vertices in $D$ and so is equal to $D$ on the assumption that all edges have the same number of vertices.

It remains to show that $S \cap c(A) = \emptyset$ with probability at least $1/2$. If $b \geq 2d|A|$ then we have

$$\mathbb{P}[S \cap c(A) = \emptyset] \geq 1 - d|A|/b \geq 1/2.$$

Finally, note that $b = O(k)$ since $|A| \leq \hs(G) + \sum_{C \subset D} \hs(M_{C,D}) \leq k + 2^d akd = O(k)$ by appealing to Lemma 3.6 and using the fact that $s_G(C) \leq ak$ for all $C \subset D$ since $D \in F$. □

A result for hypergraph matching follows along similar lines.

**Theorem 3.3.** Suppose $\text{match}(G) \leq k' = k/d$. With probability $1 - 1/\text{poly}(k)$, $\text{match}(G') = \text{match}(G)$.

**Proof.** $\hs(G) \leq dk' = k$. Let $M$ be the matching. $F \cap M$ is preserved in $G'$. Consider an edge $D \in M$ such that $C \subset D$ for some $C \in U$. Then in $G'$ we can find (by Lemma 3.7) at least $k + 1$ petals in a sunflower with core either $C$ itself or some $C' \subset C$. At most $k$ of those intersect $M \setminus \{D\}$. Therefore, there is still at least one edge we can pick for the matching. □

4 Approximating Large Matchings

The problem of approximating large matchings in the dynamic graph stream model has seen a flurry of recent activity. Four sets of related results were disclosed almost simultaneously (including a version of this paper). Assadi et al. [9] present a different $\alpha$-approximation algorithm for maximum matching that uses the same space as our algorithm.
(which they show is optimal). Konrad [36] proves slightly weaker bounds. Bury and Schwiegelshohn [11] present an algorithm for estimating the size of the maximum matching in graphs of bounded arboricity. Our second algorithm in this section is similar (the difference is that we can find the edges of an exact matching when it is small whereas they approximate the cardinality in this case by guessing and verifying the rank of a related matrix).

### 4.1 Approximating Matching in Arbitrary Graphs

**Intuition and Preliminaries.** Given a hash function \( c : V \to [b] \), we say an edge \( uv \) is colored \( i \) if \( c(u) = c(v) = i \). If the endpoints have different colors, we say the edge is uncolored. The basic idea behind our algorithm is to repeatedly sample the endpoints of an edge. Let \( G \) be the graph of arboricity \( \nu \), and let the algorithm used for Theorem 3.1, we can find the exact matching using \( O(\nu^2) \) space in the dynamic graph stream model. If \( r = O(2^{L/2} \log k) \) and \( b = 2^L + 1/\alpha \). These graphs can be generated in \( O(n^2 \log \alpha) \) space. For some \( i \),

\[ 2^i \leq \text{match}(G) < 2^{i+1} \]

and hence \( \text{match}(G') = \Omega(\text{match}(G)/\alpha) \).

This result generalizes to the weighted case using the Crouch-Stubbs technique [15]. They showed that if we can find a \( \beta \)-approximation to the maximum cardinality matching amongst all edges of weight greater than \( (1 + \epsilon)^i \) for each \( i \), then we can find a \( 2(1 + \epsilon) \)-approximation to the maximum weighted matching.

### 4.2 Matchings in Planar and Bounded-Arboricity Graphs

In this section, we present an algorithm for estimating the size of the matching in a graph of bounded arboricity. Recall that a graph has arboricity \( \nu \) if its edges can be partitioned into at most \( \nu \) forests. In particular, it can be shown that for any graph \( G \) with arboricity \( \nu \) the maximum matching in a graph of arboricity \( \nu \).

**Lemma 4.1.** (ESFANDIARI ET AL. [21]) For any graph \( G \) with arboricity \( \nu \) define a vertex to be heavy if its degree is at least \( 2\nu + 3 \) and define an edge to be shallow if it is not incident to a heavy vertex. Then,

\[ \max\{h, s\} \leq \text{match}(G) \leq 2 \max\{h, s\} \cdot \frac{2.5\nu + 4.5}{2.5\nu + 4.5} \]
where \( h \) is the number of heavy vertices and \( s \) is the number of shallow edges.

To estimate \( \max\{h, s\} \), Esfandiari et al. sampled a set of vertices \( Z \) and (a) computed the exact degree of these vertices, then (b) found the set of all edges in the induced subgraph on these vertices. The fraction of heavy vertices in \( Z \) and shallow edges in the induced graph are then used to estimate \( h \) and \( s \).

By choosing the size of \( Z \) appropriately, they showed that the resulting estimate was sufficiently accurate on the assumption that \( \max\{h, s\} \) is large. In the case where \( \max\{h, s\} \) is small, the maximum matching is also small and hence a maximal matching could be constructed in small space using a greedy algorithm.

**Algorithm for Dynamic Graph Streams.** In the dynamic graph stream model, it is not possible to construct a maximal matching. However, we may instead use the algorithm of Theorem 3.1 to find the exact size of the maximum matching. Furthermore, we can still recover the induced subgraph on sampled vertices \( Z \) via a sparse recovery sketch [25]. This can be done space-efficiently because the number of edges is at most \( 2\nu|Z| \). Lastly, rather than fixing the size of \( Z \), we consider sampling each vertex independently with a fixed probability as this simplifies the analysis significantly. The resulting algorithm is as follows:

1. Invoke algorithm of Theorem 3.1 for \( k = 2n^{2/5} \) and let \( r \) be the reported matching size.

2. In parallel, sample vertices with probability \( p = 8e^{-2n^{-1/5}} \) and let \( Z \) be the set of sampled vertices. Find the degrees of vertices in \( Z \) in \( G \) and maintain a \( 2\nu|Z| \)-sparse recovery sketch of the edges in the induced graph on \( Z \). Let \( s_Z \) be the number of shallow edges in the induced graph on \( Z \) and let \( s_Z \) be the number of heavy vertices in \( Z \). Return \( \max\{r, h_Z/p, s_Z/p^2\} \).

**Analysis.** Our analysis relies on the following lemma that shows that \( \max\{h_Z/p, s_Z/p^2\} \) is a \( 1 + \epsilon \) approximation for \( \max\{s, h\} \) on the assumption that \( \max\{s, h\} \geq n^{2/5} \).

**Lemma 4.2.** With probability at least \( 4/5 \),

\[ |\max\{h_Z/p, s_Z/p^2\} - \max\{s, h\}| \leq \epsilon \cdot \max\{n^{2/5}, s, h\} \, . \]

**Proof.** First we show \( s_Z/p^2 \) is a sufficiently good estimate for \( s \). Let \( S \) be the set of shallow edges in \( G \) and let \( E_Z \) be the set of edges in the induced graph on \( Z \). For each shallow edge \( e \in S \), define a random indicator variable \( X_e \) where \( X_e = 1 \) iff \( e \in E_Z \) and note that \( s_Z \geq \sum_{e \in S} X_e \). Then,

\[ \mathbb{E}[s_Z] = sp^2 \]

and

\[ \mathbb{V}[s_Z] = \sum_{e \in S} \sum_{e' \in S} \mathbb{E}[X_e X_{e'}] - \mathbb{E}[X_e] \mathbb{E}[X_{e'}] \, . \]

Note that

\[ \sum_{e' \in S} \mathbb{E}[X_e X_{e'}] - \mathbb{E}[X_e] \mathbb{E}[X_{e'}] = \begin{cases} p^2 - p^4 & \text{if } e = e' \\ p^3 - p^4 & \text{if } e \text{ and } e' \text{ share exactly one endpoint} \\ 0 & \text{if } e \text{ and } e' \text{ share no endpoints} \end{cases} \]

and since there are at most \( 2\nu + 3 \) edges that share an endpoint with a shallow edge,

\[ \mathbb{V}[s_Z] \leq s(p^2 - p^4 + (2\nu + 3)p^3 - p^4) \leq 2sp^2 \]

on the assumption that \( (2\nu + 3) \leq 1/p \). We then use Chebyshev’s inequality to obtain

\[ (4.2) \quad \mathbb{P}\left[|s_Z - sp^2| \leq p^2 \epsilon \cdot \max\{n^{2/5}, s\}\right] \leq \frac{2sp^2}{(p^2 \epsilon \cdot \max\{n^{2/5}, s\})^2} \leq 9/10 \, . \]

Next we show that \( h_Z/p \) is a sufficiently good estimate for \( h \). Let \( H \) denote the set of \( h \) heavy vertices in \( G \) and define an indicator random variable \( Y_v \) for each \( v \in H \), where \( Y_v = 1 \) iff \( v \in Z \). Note that \( h_Z = \sum_{v \in H} Y_v \) and \( \mathbb{E}[h_Z] = hp \). Then, by an application of the Chernoff-Hoeffding bound,

\[ (4.3) \quad \mathbb{P}\left[h_Z - hp \geq \epsilon p \max\{n^{2/5}, h\}\right] \leq \exp(-\epsilon^2pn^{2/5}/3) \leq 9/10 \, . \]

Therefore, it follows from Eq. 4.2 and 4.3 that with probability at least \( 4/5 \),

\[ |\max\{h_Z/p, s_Z/p^2\} - \max\{s, h\}| \leq \epsilon \cdot \max\{n^{2/5}, s, h\} \, . \]

**Theorem 4.2.** There exists a \( \tilde{O}(\nu e^{-2n^{4/5}/5} \log \delta) \)-space dynamic graph stream algorithm that returns a \((5\nu + 9)(1 + \epsilon)^2\) approximation of \( \text{match}(G) \) with probability at least \( 1 - \delta \) where \( \nu \) is the arboricity of \( G \).

**Proof.** To argue the approximation factor, first suppose \( \text{match}(G) \leq 2n^{2/5} \). In this case \( r = \text{match}(G) \) and \( \max\{s, h\} \leq (2.5\nu + 4.5) \text{match}(G) \) by appealing to Lemma 4.1. Hence,

\[ \text{match}(G) \leq \max\{r, h_Z/p, s_Z/p^2\} \leq (2.5\nu + 4.5) \text{match}(G) \, . \]

Next suppose \( \text{match}(G) \geq 2n^{2/5} \). In this case, \( \max\{s, h\} \geq n^{2/5} \) by Lemma 4.1. Therefore, by Lemma 4.2,

\[ \max\{h_Z/p, s_Z/p^2\} = (1 \pm \epsilon) \max\{s, h\} \, , \] and so

\[ \text{match}(G) \leq \max\{r, h_Z/p, s_Z/p^2\} \leq (1 + \epsilon) \max\{s, h\} \leq (1 + \epsilon)(2.5\nu + 4.5) \text{match}(G) \, . \]
To argue the space bound, recall that the algorithm used in Theorem 5.1 requires $\tilde{O}(n^{4/5})$ space. Note that $|Z| \leq 2np = O(\epsilon^{-n^{1/3}})$ with high probability. Hence, to sample the vertices $Z$ and maintain a $2\epsilon |Z|$-sparse recovery data structure requires $\tilde{O}(n^{4/5})$ space.

5 Sampling Kernels for Subgraph Search Problems

We extend our parameterized results to a class of problems where the objective is to search for a subgraph $H$ of $G(V,E)$ which satisfies some property $P$. In the parameterized setting, we typically search for the largest $H$ which satisfies this property, subject to the promise that the size of any $H$ satisfying $P$ is at most $k$. For concreteness, we assume the size is captured by the number of vertices in $H$, and our objective is to find a maximum cardinality satisfying subgraph. The sampling primitive $\text{Sample}_{b,2,1}$ can be used here when $P$ is preserved under vertex contraction: if $G'$ is a vertex contraction of $G$, then any subgraph $H$ of $G'$ satisfying $P$ also satisfies $P$ for $G$ (with vertices suitably remapped).

Let $\Gamma(\cdot)$ denote the vertex contraction property of $G$. Suppose that the number of vertices in $G$ is at most $n$. Hence, the data structure requires $\tilde{O}(n^{4/5})$ space. By the vertex contraction property of $P$, this means that a maximum cardinality solution for $P'$ in $G''$ is a maximum cardinality solution in $G$.

Note that for this application of the subgraph sampling primitive, it suffices to implement the sampling data structure with a counter for each pair of colors: any non-zero count corresponds to an edge in $G''$.

Note that the generality of the result comes at the cost of increasing the number of colors, and hence the space of the stream algorithms. To generalize the result to the weighted case (e.g., where the objective is to find the subgraph satisfying $P$ with the greatest total weight), we take the approach used in Section 5.1. We perform the sampling in parallel for each distinct weight value, and then round each edge weight to the closest power of $(1 + \epsilon)$ to reduce the number of weight classes to $O(\epsilon^{-1}\log W)$, with a loss factor of $(1 + \epsilon)$.

6 Lower Bounds

6.1 Matching and Hitting Set Lower Bounds

The following theorem establishes that the space-use of our matching, vertex cover, hitting set, and hyper matching algorithms is optimal up to logarithmic factors.

**Theorem 6.1.** Any (randomized) parameterized streaming algorithm for the minimum $d$-hitting set or maximum (hyper)matching problem with parameter $k$ requires $\Omega(k^d)$ space.

**Proof.** We reduce from the Membership problem in communication complexity:

<table>
<thead>
<tr>
<th>Membership</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> Alice has a set $X \subseteq [n]$, and Bob has an element $1 \leq x \leq n$.</td>
</tr>
<tr>
<td><strong>Question:</strong> Bob wants to check whether $x \in X$.</td>
</tr>
</tbody>
</table>

There is a lower bound of $\Omega(n)$ bits of communication from Alice to Bob, even allowing randomization [2].

Let $S = s_1, s_2, ..., s_n$ be the characteristic string of $X$, i.e., a binary string such that $s_i = 1$ iff $i \in X$. Let $k = \sqrt[3]{n}$. Fix a canonical mapping $h : [n] \rightarrow [k]^d$. This way we can view an $n$ bit string as an adjacency matrix of a $d$-partite graph. Construct the following graph $G$ with $d$ vertex partitions $V_1, V_2, ..., V_d$:

- Each partition $V_i$ has $dk$ vertices: for each $j \in [k]$ create vertices $v^1_{i,j}, v^2_{i,j}, ..., v^{d-1}_{i,j}$.
- Alice inserts a hyperedge $(v^1_{i,j_1}, v^2_{i,j_2}, ..., v^{d-1}_{i,j_d})$ iff the corresponding bit in the string $S$ is 1, i.e., $s_a = 1$ where $h(a) = (j_1, j_2, ..., j_d)$.
- Let $h(x) = (J_1, J_2, ..., J_d)$. Bob inserts edge $(v^1_{i,j_1}, v^1_{i,j_2}, ..., v^{d-1}_{i,j_d})$ iff $j \neq J_i$. 


Alice runs the (assumed) hitting set algorithm on the edges she is inserting using space \( f(k) \). Then she sends the memory contents of the algorithm to Bob, who finishes running the algorithm on his edges.

The minimum hitting set should include vertices \( v_{i,j}^* \) such that \( j \neq J_i \). If edge \((v_{1,j_1}^*, v_{2,j_2}^*, ..., v_{d,j_d}^*)\) is in the graph, we also need to include one of its vertices. Therefore,

\[
x \in X \iff s_x = 1
\]

\[
\iff (v_{1,j_1}^*, v_{2,j_2}^*, ..., v_{d,j_d}^*) \text{ is in } G
\]

\[
\iff \text{hs}(G) = dk - d + 1.
\]

On the other hand,

\[
x \notin X \iff s_x = 0
\]

\[
\iff (v_{1,j_1}^*, v_{2,j_2}^*, ..., v_{d,j_d}^*) \text{ is not in } G
\]

\[
\iff \text{hs}(G) = dk - d.
\]

Alice only sends \( f(k) \) bits to Bob. Therefore, \( f(k) = \Omega(n) = \Omega(k^d) \).

For the lower bound on matching we use the same construction. For each vertex \( v_{i,j}^* \) such that \( j \neq J_i \) maximum matching should include \((v_{1,j_1}^*, v_{1,j_1}^*, v_{2,j_2}^*, ..., v_{d,j_d}^*)\). If edge \((v_{1,j_1}^*, v_{2,j_2}^*, ..., v_{d,j_d}^*)\) is in the graph, we include it in the matching as well. Therefore,

\[
x \in X \iff s_x = 1
\]

\[
\iff (v_{1,j_1}^*, v_{2,j_2}^*, ..., v_{d,j_d}^*) \text{ is in } G
\]

\[
\iff \text{match}(G) = dk - d + 1.
\]

And

\[
x \notin X \iff s_x = 0
\]

\[
\iff (v_{1,j_1}^*, v_{2,j_2}^*, ..., v_{d,j_d}^*) \text{ is not in } G
\]

\[
\iff \text{match}(G) = dk - d.
\]

\[
\square
\]

### 6.2 Lower Bounds for Problems considered by Fafianie and Kratsch [22]

**Comparison with Lower Bounds for Streaming Kernels:**

Fafianie and Kratsch [22] introduced the notion of kernelization in the streaming setting as follows:

**Definition 6.1.** A 1-pass streaming kernelization algorithm receives an input \((x, k)\) and returns a kernel, with the restriction that the space usage of the algorithm is bounded by \(p(k) \cdot \log |x|\) for some polynomial \(p\).

Fafianie and Kratsch [22] gave deterministic lower bounds for several parameterized problems. In particular, they showed that:

- Any 1-pass kernel for Edge Dominating Set \((k)\) requires \(\Omega(m)\) bits, where \(m\) is the number of edges. However, there is a 2-pass kernel which uses \(O(k^3 \cdot \log n)\) bits of local memory and \(O(k^2)\) time in each step and returns an equivalent instance of size \(O(k^3 \cdot \log k)\).

- The lower bound of \(\Omega(m)\) bits for any 1-pass kernel also holds for several other problems such as Cluster Editing \((k)\), Cluster Deletion \((k)\), Cluster Vertex Deletion \((k)\), Cograph Vertex Deletion \((k)\), Minimum Fill-In \((k)\), Edge Bipartization \((k)\), Feedback Vertex Set \((k)\), Odd Cycle Transversal \((k)\), Triangle Edge Deletion \((k)\), Triangle Vertex Deletion \((k)\), Triangle Packing \((k)\), \(s\)-Star Packing \((k)\), Bipartite Colorful Neighborhood \((k)\).

- Any \(t\)-pass kernel for Cluster Editing \((k)\) and Minimum Fill-In \((k)\) requires \(\Omega(n/t)\) space.

In this section, we give \(\Omega(n)\) randomized lower bounds for the space complexity of all the problems considered by Fafianie and Kratsch. In addition, we also consider some other problems such as Path \((k)\) which were not considered by Fafianie and Kratsch. A simple observation shows that any lower bound for parameterized streaming kernels also transfers for the parameterized streaming algorithms. Thus the results of Fafianie and Kratsch [22] also give lower bounds for the parameterized streaming algorithms for these problems. However, our lower bounds have the following advantage over the results of [22]:

- All our lower bounds also hold for randomized algorithms, whereas the kernel lower bounds were for deterministic algorithms.

- With the exception of Edge Dominating Set \((k)\), all our lower bounds also hold for any constant number of passes.

#### 6.2.1 Lower Bound for Edge Dominating Set

We now show a lower bound for the Edge Dominating Set \((k)\) problem.

**Definition 6.2.** Given a graph \(G = (V, E)\) we say that a set of edges \(X \subseteq E\) is an edge dominating set if every edge in \(E \setminus X\) is incident on some edge of \(X\).

**Theorem 6.2.** For the Edge Dominating Set \((k)\) problem, any (randomized) streaming algorithm needs \(\Omega(n)\) space.
Proof. Given an instance of \textsc{Membership}, we create a graph 
\( G \) on \( n + 2 \) vertices as follows. For each \( i \in [n] \) we create a
vertex \( v_i \). Also add two special vertices \( a \) and \( b \). For every
\( y \in X \), add the edge \((a, y)\). Finally add the edge \((b, x)\).

Now we will show that \( G \) has an edge dominating set
of size 1 iff \textsc{Membership} answers YES. In the first direction
suppose that \( G \) has an edge dominating set of size 1. Then it
must be the case that \( x \in X \); otherwise for a minimum edge
dominating set we need one extra edge to dominate the star
incident on \( a \), in addition to the edge \((b, x)\) dominating itself.
Hence \textsc{Membership} answers YES. In reverse direction,
suppose that \textsc{Membership} answers YES. Then the edge
\((a, x)\) is clearly an edge dominating set of size 1.

Therefore, any (randomized) streaming algorithm that
can determine whether a graph has an edge dominating set
of size at most \( k = 1 \) gives a communication protocol for
\textsc{Membership}, and hence requires \( \Omega(n) \) space. \( \square \)

6.2.2 Lower Bound for \( \mathcal{G}\)-Free Deletion

\textsc{Definition 6.3.} Let \( \mathcal{G} \) be a set of graphs such that each
graph in \( \mathcal{G} \) is connected. We say that \( \mathcal{G} \) is bad if there is graph
\( H \in \mathcal{G} \) such that

- \( H \) is a minimal element of \( \mathcal{G} \) under the operation
  of taking subgraphs, i.e., no proper subgraph of \( H \) is in \( \mathcal{G} \)
- \( H \) has at least two distinct edges

Note that \( \mathcal{G} = \{P_2\} \) is not bad (where \( P_2 \) is the path
on two vertices) since the only minimal graph in \( \mathcal{G} \) is \( P_2 \) which
does not have two edges. On the other hand, the class of
graphs \( \mathcal{G} = \{P_3, P_4, P_5, \ldots\} \) is bad since \( P_3 \) is a minimal
graph (under operation of taking subgraphs) of \( \mathcal{G} \) and \( P_3 \)
contains two edges.

For any bad set of graphs \( \mathcal{G} \), we now show a lower bound
for the following general problem:

\[ \text{\textsc{G-Free Deletion}}(k) \]

\textbf{Input:} A bad set of graphs \( \mathcal{G} \), an undirected graph
\( G = (V, E) \) and an integer \( k \)

\textbf{Question:} Does there exist a set \( X \subseteq V \) such that \( G \setminus X \)
contains no graph from \( \mathcal{G} \)?

We reduce from the \textsc{Disjointness} problem in communica-
tion complexity.

\textsc{Disjointness}

\emph{Input:} Alice has a string \( x \in \{0, 1\}^n \) given by \( x_1 x_2 \ldots x_n \).
Bob has a string \( y \in \{0, 1\}^n \) given by \( y_1 y_2 \ldots y_n \).

\emph{Question:} Bob wants to check if \( \exists i \in [n] \) such that
\( x_i = y_i = 1 \).

There is a lower bound of \( \Omega(n/p) \) bits of communication
between Alice and Bob, allowing \( p \)-rounds and randomiza-
tion \cite{39}.

\textsc{Theorem 6.3.} For a bad set of graphs \( \mathcal{G} \), any \( p \)-pass (ran-
domized) streaming algorithm for the \( \mathcal{G}\)-\textsc{Free Deletion}
problem needs \( \Omega(n/p) \) space.

\textbf{Proof.} Since \( \mathcal{G} \) is a bad set of graphs, there is a minimal graph
\( H \in \mathcal{G} \) which has at least two distinct edges, say \( e_1 \) and \( e_2 \).
Let \( H' := H \setminus \{e_1, e_2\} \) . Given an instance of \textsc{Disjointness},
we create a graph \( G \) which consists of \( n \) disjoint copies say
\( G_1, G_2, \ldots, G_n \) of \( H' \). For each \( i \in [n] \), to the copy \( G_i \) of
\( H' \) we add the edge \( e_1 \) iff \( x_i = 1 \) and the edge \( e_2 \) iff \( y_i = 1 \).
We now show that the resulting graph \( G \) contains a copy of
\( H \) if and only if it is a YES instance of \textsc{Disjointness}.

Suppose that it is a YES instance of \textsc{Disjointness}. So
there is a \( j \in [n] \) such that \( x_j = 1 = y_j \). Therefore, to
the copy \( G_j \) of \( H' \) we would have added the edges \( e_1 \) and \( e_2 \) which
would produce an instance of \( H \). So \( G \) contains a copy of \( H \). In other direction, suppose that \( G \) contains a copy of \( H \). Note that since we add \( n \) disjoint copies of \( H' \)
and add at most two edges \((e_1 \text{ and } e_2)\) to each copy, it follows
that each connected component of \( G \) is in fact a subgraph of
\( H = H' \cup \{e_1 + e_2\} \). Since \( H \) is connected and \( G \) contains a
copy of \( H \), some connected component of \( G \) must exactly be
the graph \( H \), i.e., to some copy \( G_i \) of \( H' \) we must have added
both the edges \( e_1 \) and \( e_2 \). This implies \( x_i = 1 = y_i \), and so
\textsc{Disjointness} answers YES. \( \square \)

Since each connected component of \( G \) is a subgraph
of \( H \), the minimality of \( H \) implies that \( G \) contains a graph
from \( G \) iff \( G \) contains a copy of \( H \), which in turn is true
iff \textsc{Disjointness} answers YES. Therefore, any \( p \)-pass (randomized)
streaming algorithm that can determine whether
a graph is \( \mathcal{G}\)-free (i.e., answers the question with \( k = 0 \))
gives a communication protocol for \textsc{Disjointness}, and hence
requires \( \Omega(n/p) \) space.

This implies lower bounds for the following set of problems:

\textsc{Theorem 6.4.} For each of the following problems,
any \( p \)-pass (randomized) algorithm requires \( \Omega(n/p) \)
space: \textsc{Feedback Vertex Set}(\( k \)), \textsc{Odd Cycle Transversal}(\( k \)), \textsc{Even Cycle Transversal}(\( k \)) and
\textsc{Triangle Deletion}(\( k \)).

\textbf{Proof.} We first define the problems below:

\textsc{Feedback Vertex Set}(\( k \))

\textbf{Input:} An undirected graph \( G = (V, E) \) and an integer \( k \)

\textbf{Question:} Does there exist a set \( X \subseteq V \) of size at most \( k \)
such that \( G \setminus X \) has no cycles?

\textsc{Odd Cycle Transversal}(\( k \))

\textbf{Input:} An undirected graph \( G = (V, E) \) and an integer \( k \)

\textbf{Question:} Does there exist a set \( X \subseteq V \) of size at most \( k \)
such that \( G \setminus X \) has no odd cycles?
We verify the conditions for \( F \) under operation of taking subgraphs. Finally, finding a set \( X \subseteq V \) of size at most \( k \) such that \( G \setminus X \) has no even cycles?

**Theorem 6.5.** For a good set of graphs \( G \), any \( p \)-pass (randomized) streaming algorithm for the \( G\)-Editing\( (k) \) problem needs \( \Omega(n/p) \) space.

**Proof.** We reduce from the Disjointness problem in communication complexity. Since \( G \) is a good set of graphs, there is a minimal graph \( H \in G \) such that \( H \) is connected and has at least two distinct edges, say \( e_1 \) and \( e_2 \). Let \( H' := H \setminus \{e_1, e_2\} \). Given an instance of Disjointness, we create a graph \( G \) which consists of \( n \) disjoint copies say \( G_1, G_2, \ldots, G_n \) of \( H' \). By minimality of \( H \), it follows that \( H' \notin G \). For each \( i \in [n] \) we add to \( G_i \) the edge \( e_1 \) if \( i = 1 \) and the edge \( e_2 \) if \( y_i = 1 \). Let the resulting graph be \( G \).

We now show that \( G \) contains a copy of \( H \) if and only if Disjointness answers YES. Suppose that \( G \) contains a copy of \( H \). Note that since we add \( n \) disjoint copies of \( H' \) and add at most two edges \( e_1 \) and \( e_2 \) to each copy, it follows that each connected component of \( G \) is in fact a subgraph of \( G = H' \cup \{e_1, e_2\} \). Since \( H' \) is connected and \( G \) contains a copy of \( H \), some connected component of \( G \) must exactly be the graph \( H \), i.e., to some copy \( G_i \) of \( H' \) we must have added both the edges \( e_1 \) and \( e_2 \). This implies \( x_i = 1 = y_i \), and so Disjointness answers YES. Now suppose that Disjointness answers YES, i.e., there exists \( j \in [n] \) such that \( x_j = 1 = y_j \). Therefore, to the copy \( G_j \) of \( H' \) we would have added the edges \( e_1 \) and \( e_2 \) which would complete it into \( H \). So \( G \) contains a copy of \( H \).

Otherwise, due to minimality of \( H \), the graph \( G \) does not contain any graph from \( G \). Therefore, any \( p \)-pass (randomized) streaming algorithm that can determine whether a graph \( G \) contains a graph from \( G \) (i.e., answers the question with \( k = 0 \)) gives a communication protocol for Disjointness, and hence requires \( \Omega(n/p) \) space.

This implies lower bounds for the following set of problems:

**Theorem 6.6.** For each of the following problems, any \( p \)-pass (randomized) algorithm requires \( \Omega(n/p) \) space: Triangle Packing\( (k) \), s-Star Packing\( (k) \) and Path\( (k) \).

**Proof.** We first define the problems below:

**Triangle Packing\( (k) \)**

**Input:** An undirected graph \( G = (V, E) \) and an integer \( k \)

**Question:** Do there exist at least \( k \) vertex disjoint triangles in \( G \)?
We now show a lower bound for the \textsc{Clique} problem.

We verify the conditions for \textsc{Triangle Packing}:

- \textbf{Triangle Packing}(k) with k = 1: Take $G = \{C_3\}$ and $H = C_3$
- \textbf{s-Star Packing}(k) with k = 1: Take $G = \{K_{1,s}\}$ and $H = K_{1,s}$
- \textbf{Path}(k) with k = 3: Take $G = \{P_3, P_4, P_5, \ldots\}$ and $H = P_3$

We verify the conditions for \textsc{Triangle Packing}(k) with k = 1; the proofs for other problems are similar. Note that the choice of $G = \{C_3\}$ and $H = C_3$ implies that $G$ is good since $G$ only contains one graph $H$ which is connected and has at least two distinct edges. Finally, finding a set of edges $X$ such that the graph $(V, E \cup X)$ contains a graph from $G$ implies that it has at least one $C_3$, i.e., $X$ is a solution for \textsc{Triangle Packing}(k) with $k = 1$.

### 6.2.4 Lower Bound for \textsc{Cluster Vertex Deletion} problem

We now show a lower bound for the \textsc{Cluster Vertex Deletion}(k) problem.

**Definition 6.5.** We say that $G$ is a cluster graph if each connected component of $G$ is a clique.

$$\textsc{Cluster Vertex Deletion}(k) \quad \text{Parameter: } k$$

\textbf{Input:} An undirected graph $G = (V, E)$ and an integer $k$

\textbf{Question:} Does there exist a set $X \subseteq V$ of size at most $k$ such that $G \setminus X$ is a clique?

**Theorem 6.7.** For the \textsc{Cluster Vertex Deletion}(k) problem, any $p$-pass (randomized) streaming algorithm needs $\Omega(n/p)$ space.

**Proof.** Given an instance of \textsc{Disjointness}, we create a graph $G$ on $3n$ vertices as follows. For each $i \in [n]$ we create three vertices $a_i, b_i, c_i$. Insert the edge $(a_i, c_i)$ if $x_i = 1$ and the edge $(b_i, c_i)$ if $y_i = 1$. This is illustrated in Figure 3.

Now we will show that each connected component of $G$ is a clique iff \textsc{Disjointness} answers NO. In the first direction suppose that each connected component of $G$ is a clique. Then there cannot exist $i \in [n]$ such that $x_i = 1 = y_i$ because then the vertices $a_i, b_i, c_i$ will form a connected component which is a $P_3$; this contradicts the assumption that each connected component of $G$ is a clique. In reverse direction, suppose that \textsc{Disjointness} answers NO. Then it is easy to see that each connected component of $G$ is either $P_2$ or $P_3$, both of which are cliques.

Therefore, any $p$-pass (randomized) streaming algorithm that can determine whether a graph is a cluster graph (i.e., answers the question with $k = 0$) gives a communication protocol for \textsc{Disjointness}, and hence requires $\Omega(n/p)$ space.

### 6.2.5 Lower Bound for \textsc{Minimum Fill-In} problem

We now show a lower bound for the \textsc{Minimum Fill-In}(k) problem.

**Definition 6.6.** We say that $G$ is a chordal graph if it does not contain an induced cycle of length $\geq 4$.

$$\textsc{Minimum Fill-In}(k) \quad \text{Parameter: } k$$

\textbf{Input:} An undirected graph $G = (V, E)$ and an integer $k$

\textbf{Question:} Does there exist a set $X$ of at most $k$ edges such that $(V, E \cup X)$ is a chordal graph?

**Theorem 6.8.** For the \textsc{Minimum Fill-In}(k) problem, any $p$-pass (randomized) streaming algorithm needs $\Omega(n/p)$ space.

**Proof.** We reduce from the \textsc{Disjointness} problem in communication complexity. Given an instance of \textsc{Disjointness}, we create a graph $G$ on $4n$ vertices as follows. For each $i \in [n]$ we create vertices $a_i, b_i, c_i, d_i$ and insert edges $(a_i, b_i)$ and $(c_i, d_i)$. Insert the edge $(a_i, c_i)$ if $x_i = 1$ and the edge $(b_i, d_i)$ if $y_i = 1$. This is illustrated in Figure 3.

Now we will show that $G$ is chordal iff \textsc{Disjointness} answers NO. In the first direction suppose that $G$ is chordal. Then there cannot exist $i \in [n]$ such that $x_i = 1 = y_i$ because then the vertices $a_i, b_i, c_i, d_i$ will form an induced $C_4$; contradicting the assumption that $G$ is chordal. In reverse direction, suppose that \textsc{Disjointness} answers NO. Then it is easy to see that each connected component of $G$ is either $P_2$ or $P_3$. Hence, $G$ cannot have an induced cycle of length $\geq 4$, i.e., $G$ is chordal.

Therefore, any $p$-pass (randomized) streaming algorithm that can determine whether a graph is a chordal graph (i.e., answers the question with $k = 0$) gives a communication protocol for \textsc{Disjointness}, and hence requires $\Omega(n/p)$ space.

---

It is easy to see that the same proof also works for the problems of \textsc{Cluster Edge Deletion}(k) where we can delete at most $k$ edges and \textsc{Cluster Editing}(k) where we can delete/add at most $k$ edges.
6.2.6 Lower Bound for Cograph Vertex Deletion

We now show a lower bound for the Cograph Vertex Deletion\((k)\) problem.

**Definition 6.7.** We say that \(G\) is a cograph if it does not contain an induced \(P_4\).

**Cograph Vertex Deletion\((k)\)**

**Parameter:** \(k\)

**Input:** An undirected graph \(G = (V, E)\) and an integer \(k\)

**Question:** Does there exist a set \(X \subseteq V\) of size at most \(k\) such that \(G \setminus X\) is a cograph?

**Theorem 6.9.** For the Cograph Vertex Deletion\((k)\) problem, any \(p\)-pass (randomized) streaming algorithm needs \(\Omega(n/p)\) space.

**Proof.** We reduce from the Disjointness problem in communication complexity. Given an instance of Disjointness, we create a graph \(G\) on \(4n\) vertices as follows. For each \(i \in [n]\) we create vertices \(a_i, b_i, c_i, d_i\) and insert edges \((a_i, c_i)\) iff \(x_i = 1\) and the edge \((b_i, d_i)\) iff \(y_i = 1\). This is illustrated in Figure 4.

Now we will show that \(G\) has an induced \(P_4\) if and only if Disjointness answers YES. In the first direction suppose that \(G\) has an induced \(P_4\). Since each connected component of \(G\) can have at most 4 vertices, it follows that the \(P_4\) is indeed given by the path \(c_i - a_i - b_i - d_i\) for some \(i \in [n]\). By construction of \(G\), this implies that \(x_i = 1 = y_i\), i.e., Disjointness answers YES. In reverse direction, suppose that Disjointness answers YES. Then there exists \(j \in [n]\) such that the edges \((a_j, c_j)\) and \((b_j, d_j)\) belong to \(G\). Then \(G\) has the following induced \(P_4\) given by \(c_j - a_j - b_j - d_j\).

Therefore, any \(p\)-pass (randomized) streaming algorithm that can determine whether a graph is a cograph (i.e., answers the question with \(k = 0\)) gives a communication protocol for Disjointness, and hence requires \(\Omega(n/p)\) space.

6.2.7 Bipartite Colorful Neighborhood

We now show a lower bound for the Bipartite Colorful Neighborhood\((k)\) problem.

**Bipartite Colorful Neighborhood\((k)\)**

**Parameter:** \(k\)

**Input:** A bipartite graph \(G = (A, B, E)\) and an integer \(k\)

**Question:** Is there a 2-coloring of \(B\) such that there exists a set \(S \subseteq A\) of size at least \(k\) such that each element of \(S\) has at least one neighbor in \(B\) of either color?
Theorem 6.10. For the Bipartite Colorful Neighborhood\((k)\) problem, any \(p\)-pass (randomized) streaming algorithm needs \(\Omega\left(\frac{n}{p}\right)\) space.

Proof. We reduce from the Disjointness problem in communication complexity. Given an instance of Disjointness, we create a graph \(G\) on \(n + 2\) vertices as follows. For each \(i \in [n]\) we create a vertex \(v_i\). In addition, we have two special vertices \(a\) and \(b\). For each \(i \in [n]\), insert the edge \((a, v_i)\) iff \(x_i = 1\) and the edge \((b, v_i)\) iff \(y_i = 1\). Let \(A = \{v_1, v_2, \ldots, v_n\}\) and \(B = \{a, b\}\). This is illustrated in Figure 5.

Now we will show that \(G\) answers YES for Bipartite Colorful Neighborhood\((k)\) with \(k = 1\) iff Disjointness answers YES. In the first direction suppose that \(G\) answers YES for Bipartite Colorful Neighborhood\((k)\) with \(k = 1\). Let \(v_i\) be the element in \(A\) which has at least one neighbor in \(B\) of either color. Since \(|B| = 2\), this means that \(v_i\) is adjacent to both \(a\) and \(b\), i.e., \(x_i = 1 = y_i\) and hence Disjointness answers YES. In reverse direction, suppose that Disjointness answers YES. Hence, there exists \(j \in [n]\) such that \(x_j = 1 = y_j\). This implies that \(v_j\) is adjacent to both \(a\) and \(b\). Consider the 2-coloring of \(B\) by giving different colors to \(a\) and \(b\). Then \(S = \{v_j\}\) satisfies the condition of having a neighbor of each color in \(B\), and hence \(G\) answers YES for Bipartite Colorful Neighborhood\((k)\) with \(k = 1\).

Therefore, any \(p\)-pass (randomized) streaming algorithm that can solve Bipartite Colorful Neighborhood\((k)\) with \(k = 1\) gives a communication protocol for Disjointness, and hence requires \(\Omega\left(\frac{n}{p}\right)\) space. \(\square\)

References

[1] List of open problems in sublinear algorithms: Problem 64. [http://sublinear.info/64](http://sublinear.info/64)


