Approximating matching size from random streams

Michael Kapralov* Sanjeev Khanna† Madhu Sudan‡

January 3, 2014

Abstract

We present a streaming algorithm that makes one pass over the edges of an unweighted graph presented in *random* order, and produces a polylogarithmic approximation to the size of the maximum matching in the graph, while using only polylogarithmic space. Prior to this work the only approximations known were a folklore $\tilde{O}(\sqrt{n})$ approximation with polylogarithmic space in an n vertex graph and a constant approximation with $\Omega(n)$ space. Our work thus gives the first algorithm where both the space and approximation factors are smaller than any polynomial in n.

Our algorithm is obtained by effecting a streaming implementation of a simple "local" algorithm that we design for this problem. The local algorithm produces a $O(k \cdot n^{1/k})$ approximation to the size of a maximum matching by exploring the radius k neighborhoods of vertices, for any parameter k. We show, somewhat surprisingly, that our local algorithm can be implemented in the streaming setting even for $k = \Omega(\log n / \log\log n)$. Our analysis exposes some of the problems that arise in such conversions of local algorithms into streaming ones, and gives techniques to overcome such problems.

*MIT CSAIL, 32 Vassar Street Cambridge, MA 02139 USA. Email: kapralov@mit.edu We acknowledge financial support from grant #FA9550-12-1-0411 from the U.S. Air Force Office of Scientific Research (AFOSR) and the Defense Advanced Research Projects Agency (DARPA).

†Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104. Email: sanjeev@cis.upenn.edu. Supported in part by National Science Foundation grants CCF-1116961 and IIS-0904314. This work was done while the author was a visiting faculty at Microsoft Research New England, Cambridge, MA 02142.

‡Microsoft Research New England, One Memorial Drive, Cambridge, MA 02142, USA. madhu@mit.edu

1 Introduction

In this work we consider the task of approximating the size of a maximum matching in an undirected graph in the setting of streaming algorithms. The past two decades have seen surprisingly space-efficient streaming algorithms for an impressive variety of algorithmic problems. However, thus far no such algorithms are known for the problem of estimating the size of a maximum matching. It is easy to obtain a constant factor approximation to the size of the maximum matching when $O(n)$ space is allowed – simply maintain a maximal matching as edges arrive. When the approximation factor is allowed to be as large as $\Theta(\sqrt{n})$, it is possible to obtain an estimate in poly-logarithmic space by computing a simple sketch. Our work is motivated by the question if one can simultaneously achieve poly-logarithmic space and poly-logarithmic approximation. We answer this is in the affirmative by proving the following result:

THEOREM 1.1. *There exists a streaming algorithm that makes one pass over the edges of a graph presented in random order and outputs* poly-logarithmic approximation *to the size of the maximum matching using* poly-logarithmic space.

1.1 Techniques We start by designing a new local algorithm for estimating the size of a maximum matching in a graph, and then develop it into a streaming algorithm. We give a brief overview below by focusing on the following important special case: how does one efficiently distinguish graphs with an $\Omega(n)$ size matching from graphs which have no matchings of size $n/\mathrm{poly}\log n$.

We start by recalling a simple observation that a graph $G(V, E)$ has a matching of size $\Omega(n)$ if and only if for some integer d, it contains a subgraph H (not necessarily induced) with $\Omega(nd)$ edges such that all vertices in H have degree $\Theta(d)$. Of course, the algorithmic challenge is in identifying the right

subset of vertices and the right subset of edges. We attempt to find such a "right" subset by a simple filtering process.

Specifically, we find a sequence of graphs $G = G_0, G_1, G_2, \ldots$ as follows. We define a sequence of nested sets of vertices $V = V_0 \supseteq V_1 \supseteq V_2 \cdots$ appropriately and let G_i be the subgraph induced by G on V_i. We refer to i as the level of the graph G_i. The set V_i is chosen to be all vertices whose degree is sufficiently small (where this threshold depends on the index i) in G_{i-1}. We show that if any of the graphs is sufficiently dense (has many edges, relative to its level), then the graph has a large matching. We also give a converse, and this suggests a natural local algorithm for finding large matchings: Estimate the density of edges in G_i and use this estimate to approximate the matching size. This idea, it turns out, is easy to implement as a local algorithm. In the ith stage, vertices determine their membership in V_i, using information collected in the $(i-1)$th round by its neighbors. Density of the edges can then be measured by random sampling.

The main contribution of this work is to convert the above simple algorithm (strictly speaking, a variant of the above) into a streaming one. This conversion leads to two challenges. The first is an algorithmic one — namely, how can we estimate the number of edges at a given level? We provide a solution based on sampling — which crucially uses the fact that the edges are available in random order. Roughly to determine if a vertex $u \in V_i$, we sample neighbors of u (and the stream provides us with a sample) and test (recursively) that not too many of these neighbors are at level $i-1$. Estimating the number of samples needed, and proving that the required number of samples are available in a random stream requires some careful choices in the algorithms (including some variations on the local algorithm), but fortunately the parameters work out just right enabling the algorithm to work effectively.

This leads us to the second challenge which is in the analysis of our algorithm. We would have liked to prove statements of the form that the streaming algorithm correctly identifies the level of a vertex, but such statements are simply not true. The sampling based algorithms give probabilities with which vertices are identified at different levels, and these probabilities can be far from 0 or 1. Analyzing these probabilities, especially given that the tests are constructed recursively turns out to be hard. We manage to get around the task of determining these probabilities explicitly by defining random variables that behave roughly like the outcome of the tests, but are otherwise independent. We then show that our algorithm effectively ends up simulating these random variables even though the tests performed by our algorithm have huge dependencies due to the fact that the edges are effectively being sampled without replacement from the graph (to create the random stream). Specifically, we first analyze our algorithm in a hypothetical "i.i.d." case where the stream consists of edges of the graph are sampled uniformly and independently (with replacement). Later we reduce the real case, where the stream is a random permutation of the edges, to the i.i.d. case and show that this replacement can be carried out with relatively little loss in parameters. To finish the argument, we also show that our initial observation saying that some level i is dense if and only if the graph has a large matching, also extends to the case where the expected density of G_i is large. Putting the above ingredients together leads to our algorithm and analysis for the case when the graph has a sufficiently large matching.

1.2 Related work The problem of designing streaming algorithms to find approximately maximum matchings in bipartite graphs has received significant attention recently.

Single pass algorithms: Two natural variants have been considered in the literature: (1) the edge arrival setting, where edges arrive in the stream and (2) the vertex arrival setting, when vertices on one side of the graph arrive in the stream together with all their incident edges. The latter setting has also been studied extensively in the context of *online algorithms*, where each arriving vertex has to either be matched irrevocably or discarded upon arrival.

In a single pass, the best known approximation in the edge arrival setting in adversarial streams is $1/2$. However, a better algorithm is known under the assumption of random edge arrivals by [13], who achieve a $1/2+\epsilon$ approximation for a constant $\epsilon > 0$. On the lower bound side, it is known that no $\tilde{O}(n)$ space algorithm can achieve a better than $1 - 1/e$ approximation [7, 10].

In the vertex arrival setting, the best known algorithms achieve an approximation of $1 - 1/e$. The assumption of vertex arrivals allows one to

leverage results from online algorithms [12, 14, 11]. In the online model vertices on one side of the graph are known, and vertices on the other side arrive in an adversarial order. The algorithm has to either match a vertex irrevocably or discard upon arrival. The celebrated algorithm of Karp-Vazirani-Vazirani achieves a $1 - 1/e$ approximation for the online problem, which immediately implies an $\tilde{O}(n)$ space streaming algorithm. The $1 - 1/e$ impossibility result of [10] applies to the vertex arrival setting, implying that $1 - 1/e$ approximation is optimal for this setting.

Multiple-pass algorithms: Several small-space algorithms with strong approximation guarantees are known when multiple passes are allowed[6, 16, 5, 1, 13]. The best known algorithm [1] achieves a $1 - O(\sqrt{\log\log k/k})$ in k passes for the weighted as well as the unweighted version of the problem using $\tilde{O}(kn)$ space. A slightly better $1 - e^{-k}k^k/k! = 1 - O(1/\sqrt{k})$ approximation is known for the maximum cardinality version of the problem in the vertex arrival model[10]. Recently, [2] showed that the number of passes can be reduced substantially if slightly more than $\tilde{O}(n)$ space is available. In particular, $1-\epsilon$ approximation can be achieved in $O(p/\epsilon)$ passes using $O(n^{1+1/p})$ space. On the lower bound side, it is known that computing maximum matching size *exactly* requires $n^{1+\Omega(1/p)}/p^{O(1)}$ space in p passes [8]. Recently, [9] proved strong lower bounds for the communication complexity of approximate distributed maximum matching. For example, they showed that the amount of communication needs to be $\Omega(kn)$ when the graph is stored across k sites and a constant factor approximation is desired, matching the communication cost of a direct simulation of known streaming algorithms.

Local algorithms: The problem of obtaining sublinear time local algorithms for matching problems has also received significant attention recently. Very efficient solutions are known, some of which yield constant factor approximation to maximum matching size in poly(d) time, where d is the maximum degree of the graph (see, e.g. [17, 19, 4, 18, 15] and references therein). None of these algorithms, however, seem directly amenable to the streaming model when the underlying graph has unbounded degree.

1.3 Organization We present our deterministic local exploration algorithm for estimating the maximum matching size in Section 2. Then in Section 3, we give an overview of our approach for implementing the local algorithm in the streaming setting. We present in Section 4, a detailed implementation of the streaming algorithm. Our analysis consists of two steps. We first show in Section 5 that our streaming algorithm correctly estimates the matching size when given a stream of m i.i.d. samples of the edges of G as input. We then show in Section 6 that our streaming algorithm behaves similarly when the edges are presented as a random permutation (i.e. the two executions can be statistically coupled). This completes the proof of our main result. Finally, we conclude with some future directions in Section 7.

2 Estimating Matching Size via Local Explorations

We design here a simple local algorithm to estimate the size of a maximum matching. The algorithm is based on an iterative peeling process whereby in each iteration we remove the vertices with the highest residual degree. The goal of the process is to identify an iteration where the number of edges in the residual graph is large with respect to the maximum residual degree. Such a graph naturally certifies existence of a large matching, and as we show, if there is a large matching, one must necessarily encounter such a graph in one of the iterations of the algorithm. This iterative process lends itself naturally to a sampling-based local algorithm where we estimate the size of the largest matching by estimating the number of edges in the residual graph after each iteration, and determine membership of an edge in a residual graph by a local exploration of its end-points.

In the rest of the paper, we will find it more convenient to work with following decision version of the estimation problem that we refer to as the Gap-Matching problem: given a threshold U and gap parameter $g \geq 1$, distinguish between the following two cases:

YES if G contains a matching of size at least U;

NO if G does not contain a matching of size larger than U/g.

In particular, we will interested in designing algorithms that work for $g = \Theta(\text{poly}(\log n))$. We

note that Gap-Matching can be used to obtain an $O(g)$-approximation to matching size by running the Gap-Matching algorithm for a geometrically decreasing sequence of U's, and outputting the matching size to be at least U^*/g where U^* is the highest value of U for which the algorithm returns YES..

In what follows we will denote the degree of a node $u \in V$ in G by $\deg(u)$. For a set $S \subseteq V$ we will write $\deg_S(u)$ to denote the degree of $u \in V$ to the vertices in S. Let d_{max} denote an upper bound on maximum degree in G (our main result holds without any degree assumptions, but our techniques also have interesting consequences for bounded degree graphs, so it is useful to introduce d_{max}). We start by analyzing the following simple algorithm for Gap-Matching :

Algorithm 1 Gap matching via distance-k neighborhood exploration

```
1: procedure GapMatching(U, d_0, ..., d_k)
Require: d_0 = d_max ≥ d_1 ≥ ... ≥ d_{k-1} ≥ d_k = 1
2:     τ_i ← (3/2)(U d_i)/g for 0 ≤ i ≤ k − 1.
3:     V_0 ← V, E_0 ← E
4:     for i = 1 to k − 1 do
5:         V_i ← {u ∈ V_{i−1} : deg_{V_{i−1}}(u) ≤ d_i}
6:         E_i ← E ∩ (V_i × V_i)
7:     end for
8:     for i = 0 to k − 1 do
9:         if |E_i| > τ_i then
10:            return YES
11:        end if
12:    end for
13:    return NO
14: end procedure
```

Roughly, the algorithm partitions the vertices of the input graph into distinct *levels* by an iterative process. For $i \in [1 : k]$, let V_i denote the set of vertices with level at least i, and let E_i denote the set of edges in the graph induced by vertices in V_i. Then V_0 is simply V, and V_{i+1} is the set of all vertices of "sufficiently low" degree (specifically degree at most d_{i+1}) in $G_i = (V_i, E_i)$. The algorithm simply computes these sets (we argue later that this can be done locally) and checks if some level i has sufficiently many edges (relative to d_i, the degree threshold for that level). It outputs yes if such a level exists and no otherwise. The following lemma argues correctness of this algorithm.

LEMMA 2.1. *Let $d_0 = d_{max} \geq d_1 \geq \ldots \geq d_{k-1} \geq d_k = 1$ be any sequence of degrees such that $k\eta < 2g/3$ where $\eta = \max_{i \in [1:k]}(d_{i-1}/d_i)$. Then Algorithm 1 always outputs YES if G is a Yes-instance, and NO if G is a No-instance.*

Proof. We first show that whenever Algorithm 1 outputs YES, then G is a Yes-instance. Let i be the index for which $|E_i| > \tau_i$. Then by assigning a fractional weight of $1/d_i$ to each edge in E_i, we obtain a feasible fractional solution to the matching polytope with blossom constraints removed (since vertex degrees are bounded by d_i in the graph induced by E_i). The value of this solution is at least $|E_i|/d_i$. It is well-known that any feasible fractional solution with blossom constraints removed can be converted to an integral solution whose value is at least 2/3 times the value of the fractional solution. Thus G contains an integral matching of size strictly greater than

$$\frac{2}{3} \cdot \tau_i = \frac{2}{3} \cdot \frac{1}{d_i} \cdot \frac{3}{2} \frac{U d_i}{g} = \frac{U}{g},$$

and hence is a Yes-instance.

We now prove the converse, that is, whenever G is a Yes-instance, Algorithm 1 outputs YES. For $0 \leq i \leq k-2$, let $Z_i = V_i \setminus V_{i+1}$, and let $Z_{k-1} = V_{k-1}$. Clearly, Z_i's induce a partition of the vertex set V. We observe that for $0 \leq i \leq k-2$,

$$|E_i| \geq |V_i \setminus V_{i+1}| \cdot d_{i+1} = |Z_i| \cdot d_{i+1} \geq |Z_i| \cdot \frac{d_i}{\eta}.$$

We consider two cases. First, suppose that $|Z_i| > U/k$ for some $i \leq k-2$. Then

$$|E_i| \geq |Z_i| \frac{d_i}{\eta} \geq \frac{U}{k} \frac{d_i}{\eta} > \frac{3U d_i}{2g} \geq \tau_i,$$

since $k\eta < 2g/3$ by our assumption. Otherwise, removing vertices in $Z_0 \cup Z_1 \cup \ldots \cup Z_{k-2}$ leaves a matching of size at least $|U| - (k-1)|U|/k = |U|/k$, so

$$|E_{k-1}| \geq \frac{U}{k} \geq \frac{U}{k} \cdot \frac{d_{k-1}}{\eta} > \frac{3U d_{k-1}}{2g} = \tau_{k-1},$$

since $k\eta < 2g/3$ by our assumption. Thus the algorithm necessarily outputs YES whenever G is a Yes-instance.

Note that by choosing $d_i = n^{1-i/k}$, we can use the algorithm above to solve the gap matching problem for $g = \Theta(kn^{1/k})$. In particular, by setting $k = \Theta(\log n/\log\log n)$, we conclude that gap matching can be solved for $g = \log^{O(1)} n$. We state below two easy corollaries of Algorithm 1. The first shows that the algorithm above can be implemented "locally" to obtain an $O(\log d)$ approximation to maximum matching size in G in $d^{O(\log d)}$ time, and the second shows that it can be implemented in the streaming setting to obtain an $O(\log d)$ approximation to maximum matching size in G in $d^{O(\log d)}$ space, using $O(\log d)$ *passes* over the edges of G. Note that this latter bound is much weaker than the main result of this paper: a single-pass streaming algorithm that obtains a poly-logarithmic approximation to the matching size using poly-logarithmic space. An important distinction, however, is that our main result assumes a random order of arrival of edges, while the bound shown below holds for an adversarial order of arrival of edges in the graph.

LEMMA 2.2. *There is a randomized local algorithm that, given the ability to sample uniformly random edges of a graph G with vertex degrees bounded by d, outputs an $O(\log d)$ approximation to maximum matching size in G in $d^{O(\log d)}$ time.*

Proof. Our starting point is the simple observation that if in Algorithm 1, for $0 \leq i \leq k-1$, we replace the exact quantities $|E_i|$ with estimates $\tilde{m}_i$ such that $\tilde{m}_i > \tau_i$ whenever $|E_i| \geq 2\tau_i$ and $\tilde{m}_i \leq \tau_i$ whenever $|E_i| \leq \tau_i/2$, we get an algorithm that always outputs yes if there is a matching of size at least $2U$, and outputs no, whenever the maximum matching size is at most $U/2g$. Thus this approximate variant of Algorithm 1 suffices to obtain an $O(g)$-approximation. In what follows, we show that a randomized local algorithm can be used to obtain estimates $\tilde{m}_i$, $0 \leq i \leq k-1$ to within a factor of 2 in $d^{O(\log d)}$ time.

Let k be the smallest integer such that $d \leq 2^k$. We will sample the behavior of Algorithm 1 with paramaters $k = \lceil \log d \rceil$, $d_i = 2^{k-i}$, $g = 4k$, and by running the algorithm for each $U \in \{m/4d, m/2d, 2m/d, ..., n/2\}$. Clearly, in a degree d-bounded graph, the maximum matching size is at least $m/2d$ and is at most $n/2$. Thus the highest value U for which Algorithm 1 returns a Yes, the quantity U/g gives us an $O(g)$ approximation to the size of the maximum matching. If the algorithm returns No for all values of U, then we return m/d as the answer.

We now describe a simple sampling based local algorithm that mimics the behavior of Algorithm 1 for any value of U. Fix a value of U. The local algorithm samples a set $\tilde{E} \subseteq E$ of $\Theta(d\log^2 d)$ edges uniformly at random, and determines sets $\tilde{E}_i = \tilde{E} \cap E_i$. Since U is lower bounded by $m/4d$, and $g = \Theta(\log d)$, we have $\min_{0\leq i\leq k-1} \tau_i = \Omega(m/(d\log d))$. A standard application of Chernoff bounds now gives us that $\Theta(d\log^2 d)$ samples suffices to obtain estimates $\tilde{m}_i$ as described above, with probability at least $1 - 1/\text{poly}(d)$.

Finally, to complete the proof, we describe how a local algorithm can test whether or not an edge $e = (u, v) \in E_i$. To do this test, the algorithm first grows a distance i breadth-first search (BFS) tree from u – let S_u denotes this set of vertices. Note that a vertex that is more than distance i away from u plays no role in determining if $u \in V_i$. It thus suffices to consider the graph G_u induced by S_u and apply the iterative process of Algorithm 1 to G_u, and check if $u \in V_i$ at the end of i iterations of the process. Since $|S_u| \leq d^i$, and $i \leq \lceil \log d \rceil$, this can be implemented in $d^{O(\log d)}$ time by a local algorithm. We repeat the same computation to determine if $v \in V_i$. The edge $(u, v) \in E_i$ iff both u and v are determined to be in V_i. Total work done is thus bounded by $O(\log d) \times O(\log d) \times \Theta(d\log^2 d) \times d^{O(\log d)} = d^{O(\log d)}$; where the first $O(\log d)$ term corresponds to number of distinct values of U for which the algorithm is run, and the second $O(\log d)$ term corresponds to the k distinct values of i for which the estimate $\tilde{m}_i$ is computed inside each iteration. Any single computation of the estimate $\tilde{m}_i$ fails with probability at most $1/\text{poly}(d)$. Hence using the union bound, the overall probability of error can be bounded by $O(\log d) \times O(\log d) \times 1/\text{poly}(d) = 1/\text{poly}(d)$.

LEMMA 2.3. *For any $k = O(\log d)$, there is a randomized $O(d^{k+1}\log^4 d)$ space algorithm that obtains an $O(kd^{\frac{1}{k}})$-approximation to maximum matching size in k passes over edges of G presented in any (adversarial) order in graphs with vertex degrees bounded by d.*

Proof. The proof is similar to the proof of Lemma 2.2 with the only modification being that the local computation step in the proof is implemented in the streaming setting. We will invoke Algorithm 1 with $d_i = d^{1-\frac{i}{k}}$, and $g = \Theta(kd^{\frac{1}{k}})$. The local algorithm in the proof above needs to com-

pute distance k breadth-first search trees from the end-points of sampled edges. Total number of edge samples for which this computation is performed is bounded by $O(\log d) \times O(\log d) \times \Theta(d \log^2 d) = O(d \log^4 d)$. The streaming algorithm will grow in parallel the BFS trees for vertices in all the samples, expanding the BFS tree by one level in each streaming pass. Note that this step can be implemented even when the edge arrival order in each pass is adversarially chosen. Thus after k passes, the streaming algorithm has gathered precisely the information used by the local algorithm in the proof of Lemma 2.2. Total space used in this process is bounded by $O(d \log^4 d) \times d^k = O(d^{k+1} \log^4 d)$, and hence the assertion of the lemma follows.

One can show a family of instances on which Algorithm 1 does not give a better than logarithmic approximation for a natural choice of parameters.

3 An Overview of the Streaming Algorithm

In this section, we give an overview of the ideas needed to design a single-pass streaming implementation of Gap-Matching (Algorithm 1) that uses only poly-logarithmic space and distinguishes between instances whose matching size is separated by a poly-logarithmic factor. To motivate the design and analysis of our algorithm, it is helpful to think of the stream of edges of G as a collection of i.i.d. (independent and identically distributed) uniform samples of edges of G and analyzing various events under this assumption. Over a stream of length m, the set of edges seen by the algorithm in the i.i.d. model is clearly distinct from the set of edges seen in the random permutation model — i.i.d. stream will have many repeated edges, with some edges never appearing in the stream. Nevertheless, our algorithm behaves similarly in both cases (when the edges are sampled i.i.d., or streamed as a random permutation). Indeed our formal analysis works first with the i.i.d. case (Section 5), and then shows that the key events in the i.i.d. analysis have roughly the same probability even when the edges are taken from a random permutation stream (Section 6). In the remainder of this section, we give an overview of the main ideas assuming that the stream contains m i.i.d samples of edges of G.

A small-space implementation of Algorithm 1 needs to handle the following key steps in the algorithm:

(A) determine the sets $V_i, i = 0, \ldots, k-1$;

(B) compare the size of $|E_i| = |E \cap (V_i \times V_i)|$ to the threshold τ_i, $i = 0, \ldots, k-1$.

(A) Testing membership in V_i. Algorithm 1 performs residual degree tests for nodes $u \in V$. In particular, given $u \in V_i$ the algorithm decides whether or not u belongs to V_{i+1} by comparing u's degree in V_i to d_{i+1}. The main idea behind our implementation is to perform this test approximately by sampling: instead of computing the degree of u in V_i exactly and comparing it to d_{i+1}, we sample m/d_{i+1} edges of G uniformly at random, and for each edge (u, w) in the sample we test "recursively" if w belongs to V_i. Note that if $\deg_{V_i}(u) > 2d_{i+1}$, say, we expect at least two such w's to appear in the sample, and if $\deg_{V_i}(u) < d_{i+1}/2$, then we expect to see no such neighbors in this sample with probability at least $1/2$. This gives us the following test for membership of u in V_{i+1}, which we denote V_{i+1}-Test(u):

> Take a sample S of m/d_{i+1} edges of G. If S contains at least one edge (u, w) with $w \in V_i$,
>
> output **NO**, otherwise **YES**.

Note that V_0-Test(u) is trivial in that it outputs **YES** without examining the stream. For higher values of i, we must show that the number of samples needed to determine whether $w \in V_i$ for some $(u, w) \in S$ is not too large. Let ℓ_i denote the number of samples needed by V_i-Test(w), maximized over $w \in V_{i-1}$. We now derive a recurrence for ℓ_{i+1} in terms of ℓ_j's for $j \leq i$. A naive bound would suggest that it takes ℓ_i samples to test membership of $w \in V_i$ for every $(u, w) \in S$ and potentially every edge in S may be incident on u, yielding a bound of $\ell_{i+1} \approx m/d_{i+1} \cdot \ell_i$, but this would be too large an overcount. To improve on this we need to use the fact that $u \in V_1$ (and so has not too large a degree), and further it is in V_2 and so its degree in V_1 is even less (so most of its neighbors will fail the V_1 test) etc. More formally, we have that a V_i-node can have at most d_{j+1} neighbors that belong to V_j for all $j \in [0 : i-1]$, we get the recursive relation

$$\ell_{i+1} = \frac{m}{d_{i+1}} + \ell_1(d_1/d_{i+1}) + \ldots + \ell_i(d_i/d_{i+1}). \quad (3.1)$$

Indeed, the algorithm samples m/d_{i+1} edges, and among these edges one expects to find up to

d_{j+1}/d_{i+1} neighbors of u that belong to V_j, and hence need to be tested for membership in V_{j+1}, for each $j \in [0 : i-1]$ (note that the number of neighbors in V_i is the deciding factor for membership in V_{i+1}). The base case is provided by $\ell_1 = m/d_1$, since this is the number of edges that need to be sampled in order to determine if the degree of a node u is above d_1. In order to understand the growth of ℓ_{i+1} in (3.1), we rewrite it in the form

$$d_{i+1}\ell_{i+1} = m + \ell_1 d_1 + \ell_2 d_2 + \ldots + \ell_{i-1}d_{i-1} + \ell_i d_i.$$

Thus, we have $d_{i+1}\ell_{i+1} = 2^i m$ for $i = 0, \ldots, k-1$, suggesting that one cannot get a $\tilde{O}(1)$ approximation via this recursion, since it precludes choosing $k = \Theta(\log n / \log\log n)$ which is necessary for obtaining a poly-logarithmic approximation [1]. To remedy this, we modify the algorithm to ensure that its sampling complexity follows a recursion with a milder growth, allowing us to obtain a $\tilde{O}(1)$ approximation. For a parameter $\gamma > 0$ that will be chosen to be $O(1/\text{poly}(k))$, we will ensure that the sampling complexity of the V_{i+1}-TEST satisfies

(3.2)
$$\ell_{i+1} = \frac{m}{d_{i+1}} + \gamma\left(\ell_1(d_1/d_{i+1}) + \ldots + \ell_i(d_i/d_{i+1})\right),$$

which implies that $d_{i+1}\ell_{i+1} \leq (1+\gamma)^i m$, $i = 0, \ldots, k-2$. Thus

$$\ell_{i+1} \leq (1+\gamma)^i m/d_{i+1} \leq 2m/d_{i+1} \leq m \tag{3.3}$$

whenever $\gamma < 1/k$ and d_{k-1} is larger than a constant, suggesting that a random stream of m edges should be sufficient. We will later show that the recursion specified by (3.2) can indeed be achieved by adding an extra condition for membership in V_i for $i \geq 2$. In particular, we insist that only nodes u that satisfy $\deg_{V_{i-1}}(u) \leq \gamma d_i$ are included. This condition dampens the growth of ℓ_i, allowing us to achieve a poly-logarithmic approximation.

(B) Comparing $|E_i|$ to τ_i. We take the following natural approach for comparing the size of E_i to the threshold τ_i. Choose a uniformly random set of edges $E^* \subseteq E$ of size about m/τ_i. For each edge $e = (u, v) \in E^*$, run a V_i-TEST(u) and a V_i-TEST(v), and count the number of edges in this sample for which both endpoints passed the V_i-TEST (since we can only use poly-logarithmic space, we only sample a new edge from the stream once our current test finishes). If this count is positive, we conclude that $|E_i| \geq \tau_i$ and accept (in our implementation, we take a sample of size $\Theta(\frac{m \log n}{\tau_i})$ to get concentration).

The main question that needs to be resolved for this approach to be feasible is the number of samples that the invocations of V_i-TEST will consume – we need it to be bounded by m, the length of the stream. A direct bound using (3.3) gives

$$|E^*| \cdot \ell_i \approx |E^*| \cdot \left(\frac{m}{d_i}\right) = \left(\frac{m}{\tau_i}\right) \cdot \left(\frac{m}{d_i}\right) = \left(\frac{2}{3}\frac{g}{Ud_i}m\right) \cdot \left(\frac{m}{d_i}\right),$$

where the last equality uses the value of the parameter τ_i from Algorithm 1. Note that to obtain a poly-logarithmic approximation, we need the gap parameter g as well as the smallest allowed degree d_i to be bounded by a poly-logarithmic function. Thus the bound above behaves as $\tilde{\Omega}(m^2/U)$ which is much larger than m in general. However, the following observation makes the approach feasible: since the V_i-TESTs take longer as i gets larger, for each edge $(u, v) \in E^*$ we first run a V_1-TEST on its endpoints, then proceed to V_2-TEST only if both endpoints pass etc. In other words, we start with the lowest level tests and keep running them on both endpoints as long as both of them pass previous tests, hoping that most edges will fail the shorter lower-level tests. Then for an edge $e \in E_j, j < k-1$ we will run all tests up to V_{j+1}-TEST, observe that one of the end-points of e fails the V_{j+1}-TEST, and declare the edge e to be in E_j (for an edge $e \in E_{k-1}$ it would be natural to end with the V_{k-1}-TEST since there is no V_k-TEST to run; however, in our actual implementation we will stop at E_{k-2}, see section 4). Thus the number of samples needed to test edges in E^* can be bounded by

$$\sum_{j=0}^{i-1} |E^* \cap E_j| \cdot \ell_{j+1} + \ell_i,$$

where the first term accounts for edges in E^* that fail after running a V_{j+1}-TEST, and the second term accounts for a possibly successful V_i-TEST. This is already much better than the previous bound but still does not quite get the job done. Our final observation is that we can first run our tests for E_1, only proceed to E_2 if we discover that $|E_1| \leq \tau_1$

[1] We note, however, that this analysis can already be used to obtain an $n^{o(1)}$ approximation.

(since we can already accept otherwise), and then only proceed to E_3 if $|E_2| \leq \tau_2$ etc. Thus, when running the tests for E_i, we can assume that $|E_j| \leq \tau_j$ for $j \in [0:i-1]$. Since E^* is a sample of m/τ_i edges of E, we expect to have $|E^* \cap E_j| \leq \tau_j/\tau_i$ in this case. Assume for concreteness that $\tau_i \approx U \cdot d_i/g$ as in Algorithm 1. Then the number of samples taken by our tests is bounded by

$$\sum_{j=0}^{i-1}(\tau_j/\tau_i)\cdot \ell_{j+1} + \ell_i \approx \sum_{j=0}^{i-1}(\tau_j/\tau_i)\cdot m/d_{j+1} + m/d_i$$
$$= \sum_{j=0}^{i-1} m(d_j/d_i)\cdot \frac{1}{d_{j+1}} + m/d_i$$
$$\leq i \cdot \max_{j\in[0:i-1]}(d_j/d_{j+1})\cdot m/d_i + m/d_i$$

By choosing the ratio d_j/d_{j+1} to be at most $\log^{O(1)} n$, and terminating the process once d_i becomes poly-logarithmic, the expression above can be bounded by m. This is exactly what our algorithm will do: we only test whether sets E_j, $j < k-1$ are sufficiently large, slightly trading off the quality of approximation for the efficiency of the testing process. In the next section we give full details of our algorithm.

4 The Streaming Algorithm

In this section, we present in detail our streaming algorithm for solving the Gap-Matching problem. As explained earlier, the main idea behind our algorithm is to mirror the behavior of Algorithm 1. Recall that Algorithm 1 defines k near-regular induced subgraphs $G_i = (V_i, E_i), i = 0, \ldots, k-1$, via an adaptive exploration process. It outputs **YES** if at least one of these subgraphs is sufficiently dense, and **NO** otherwise. In particular, Algorithm 1 outputs **YES** iff one has $|E_i| \geq \tau_i$ for at least one $i \in [0:k-1]$, where τ_i's are suitably chosen thresholds.

For each $L = 0, \ldots, k-2$, our algorithm samples $(C \log n)m/\tau_L$ uniformly random edges of G, and for every sampled edge $e = (u, v)$ runs a V_j-Test on each endpoint of e, which is a randomized equivalent of a test for membership in the sets V_j from Algorithm 1. The algorithm counts the number of edges in this sample whose both endpoints pass the V_j-Test. It then outputs **NO** if the number of such edges exceeds $C \log n$ and **YES** otherwise (see Algorithm 2 below). Since our algorithm cannot use more than polylogarithmic space, a new edge is taken from the stream for testing only once testing of the previous edge finishes. The V_j-Test itself is a recursive procedure similar to Algorithm 1. V_1-Test(u) samples a fixed number of edges in the stream, runs V_{j-1}-Test on the neighbors of u found in this way, and outputs **YES** if and only if the number of neighbors that pass the V_{j-1}-Test is smaller than a threshold. The pseudocode of V_j-Test is given in Algorithm 4. Since V_j-Test takes more time for large j, we introduce an additional primitive GetLevel(u, j) (Algorithm 3), which runs V_j-Test for increasing j starting with $j = 0$ while it gets **YES** answers, crucially reducing the sample complexity.

4.1 The algorithm

We now give the pseudocode for the main algorithm and various auxiliary subroutines invoked by it. We assume that the algorithm knows m and n. This assumption is not restrictive, since running copies of the algorithm for a geometrically increasing sequence of m and n only increases the space requirement by a poly$(\log n)$ factor.

Algorithm 2 A streaming implementation of the Gap-Matching algorithm

```
1:  procedure GapMatching(U, g)
2:      α_j ← 0 for all j = 0, ..., k − 2
3:      ▷ Counters for number of successful experiments for ν_j(E)
4:      for L = 1 to k − 2 do
5:          ▷ Run the loop iterations in parallel
6:          Q_L ← (C log n)m/τ_L
7:          for t = 0 to Q_L do
8:              (u, v) ← next edge in the stream
9:              for s = 1 to L do
10:                 If V_s-Test(u)==0 then break
11:                 If V_s-Test(v)==0 then break
12:             end for
13:             If s = L then α_s ← α_s + 1
14:         end for
15:         If α_L > (9/10)C log n return YES
16:     end for
17:     return NO
18: end procedure
```

The V_j-tests are recursive, with V_1-test being a simple degree test, and V_j-test calling V_{j-1}-tests recursively. The function GetLevel(u, r) given below runs V_j-tests starting from the lowest values of j and keeps running these tests while it gets

positive answers. The motivation for this is to determine the largest j such that V_i-Test(u), $i = 0, \ldots, j$ return 1 while using few samples.

Algorithm 3 Determining if sampled level of a vertex u is at least r

```
1: procedure GetLevel(u, r)                          ▷ r ≥ 0
2:     if r ≥ 2 then
3:         L ← GetLevel(u, r − 1)
4:         If L < r − 1 then return L
5:     end if
6:     If V_r-Test(u)==0 then return r − 1
7:     else return r
8: end procedure
```

Algorithm 4 V_j-Test(u), $j \geq 2$

```
1: procedure V_j-Test(u)
2:     α ← 0
3:     R_j ← m/d_j
4:     for i = 1 to R_j do
5:         e = (u, w) ← next edge in the stream
6:         If e ∉ δ(u) continue
7:         L ←GetLevel(v, j − 2)
8:         If L == j − 2 then α ← α + 1
9:         If α > C log n then return 0
10:    end for
11:    return 1
12: end procedure
```

Algorithm 5 V_1-Test(u)

```
1: procedure V_1-Test(u)
2:     α ← 0
3:     R_1 ← m/d_1
4:     for i = 1 to R_1 do
5:         e ← next edge in the stream
6:         If e ∉ δ(u) continue
7:         w ← other endpoint of e
8:         L ←GetLevel(w, 0)
9:         If L == j − 1 then α ← α + 1
10:        If α > C log n then return 0
11:    end for
12:    return 1
13: end procedure
```

Algorithm 6 V_0-Test(u)

```
1: procedure V_0-Test(u)
2:     return 1
3: end procedure
```

We now note that GapMatching(U, g) can be implemented using $O(\text{poly}(\log n))$ space as long as $k = O(\log n)$. Indeed, first observe that GetLevel can be implemented to use $O(\text{poly}(\log n))$ space. This is because GetLevel(u, r) only calls GetLevel(w, j) for $j < r$ and V_j-Test(w) for $j \leq r$ for various vertices w. Thus, the depth of the call stack can not exceed $O(k)$. At the same time, each call to GetLevel(w, j) and V_j-Test(w) only uses $O(\log n)$ space for local variables (since both functions use at most one counter). These two facts together imply that $O(\text{poly}(\log n))$ space is sufficient. Finally, note that GapMatching(U, g) runs at most $L = O(k) = O(\log n)$ loops in parallel, which completes the argument.

5 Analysis for the IID Model

In this section, we prove that the streaming algorithm presented in Section 4 solves the Gap-Matching in $\text{poly}(\log n)$ space using $O(m/\log^2 n)$ i.i.d. samples. Specifically, we show the following result:

Theorem 5.1. *Algorithm 2 solves* Gap-Matching$_{U,g}$ *with high probability in poly*$(\log n)$ *space using* $O(m/\log^2 n)$ *i.i.d. samples of edges of* G *when* $U, g \geq$ *poly*$(\log n)$.

At a high level, our proof relies on defining a sequence of probability distributions, namely $\{\nu_j\}_{j=0}^{k-1}$, where $\nu_j(u) \in [0, 1]$ for each $u \in V$, such that the weight $\nu_j(E)$ will serve as a proxy for the set E_j used in Algorithm 1. We will refer to these functions as *level distributions* (see definitions below). We then show that Algorithm 2 correctly estimates the value $\nu_j(E)$ using only $O(m/\log^2 n)$ edge samples. The remainder of this section is organized as follows. Section 5.1 summarizes various parameters used in our analysis; Sections 5.2 and 5.3, formally define the notion of level distributions, and establish a tight connection between level distributions and the size of the maximum matching. In Section 5.4 we show that our streaming algorithm distinguishes between the Yes and No instances with high probability, without bounding the

number of samples used, and then finally in Sections 5.5, 5.6, and 5.7, we complete the analysis by bounding the number of i.i.d. samples used to be $O(m/\log^2 n)$.

5.1 Parameters We start with a glossary of parameters that are used in the algorithm and analysis, and list them here for reference:

- U, g – upper bound and gap for $\mathsf{Gap\text{-}Matching}_{U,g}$
- $n = d_0 \geq d_1 \geq \ldots \geq d_{k-1} \geq d_k = 1$ – degree thresholds for the algorithm. We will use a geometric sequence of degree thresholds $d_i = n^{1-1/k}$, where $n^{1/k}$ will be chosen to be polylogarithmic in n.
- $C > 0$ – constant such that sum of Bernoulli random variables with expectation at least $(1/2)C \log n$ deviates from expectation by more than a $1 \pm 1/100$ factor is bounded by n^{-100};
- τ_j – thresholds such that $\nu_j(E) \geq \tau_j$ implies that we are in the **YES** case. These parameters will be defined precisely in the parameter setting lemma (Lemma 5.16). We will set
$$\tau_j \approx (C \log n) U d_{j+2}/k$$
for $j \in [0 : k-3]$, and let $\tau_{k-2} \approx U$.
- R_j – the number of edges sampled for testing in V_j-Test, which we choose as $R_j = m/d_j$;
- Q_L – the number of edges sampled for testing whether $\nu_L(E) \geq \tau_L$ by GapMatching. We use $Q_L = (C \log n)m/\tau_L$.

Setting parameters to the proper values is a technical task that we do at the end of this section in Lemma 5.16. All lemmas in this section will assume that the following relations **p1-p5** hold for our choice of parameters. We list them here for reference:

p1 $d_j = (d_{k-1})^{k-j}$ for $j \in [0 : k-2]$, and $d_{k-1} \geq C' \log^4 n$ for a sufficiently large constant $C' > 0$.

p2 $\gamma \leq C'' \log^{-4} n$ for a sufficiently large constant $C' > 0$. The parameter γ does not appear in our algorithm, but will be useful for analyzing the sampling complexity of our subroutines in section 5.5. It is analogous to the parameter γ in section 3.

p3 $\tau_j/((C' \log^4 n)d_j) \geq C \log n$ for all $j \in [0 : k-2]$ and a sufficiently large constant $C' > 0$.

p4 $\frac{1}{6}\tau_j/d_j^{max} \geq U/g$ for all $j \in [0 : k-2]$.

p5 $\gamma d_{j-1}/d_j = C \log n$ for all $j \in [1 : k-1]$.

We note that here and below the constant $C > 0$ is the constant defined above that ensures good concentration for sums of Bernoulli rv's with expectation at least $C \log n$.

5.2 Level distributions We now define a collection of distributions associated with nodes of G that captures the behavior of V_j-Tests. We refer to these distributions as *level distributions*, where we think of a node u for which the V_j-test accepts as a j-level node. This will be useful for analyzing the approximation guarantees provided by Algorithm 2.

We first define Bernoulli 0/1 random variables $A_i(u)$ for each node $u \in V$ such that the set of nodes for which $\prod_{i=0}^{j} A_i(u) = 1$ resembles the set V_j in Algorithm 1.[2] We will later show how these distributions can be used to approximate matching size in G.

Distributions $A_j(u)$. Fix $u \in V$. First, define $A_0(u) \equiv 1$. The distributions $A_j(u), j = 1, \ldots, k-1$ are defined as follows. Let $e_1, \ldots, e_{R_j}$ denote i.i.d. samples of edges of G, where

$$R_j = m/d_j. \tag{5.4}$$

Let $W_j^u := \bigcup_{e_r=(u,w), r\in[1:R_j]} \{w\}$ denote the set of neighbors w of u such that (u, w) appeared in the sample (note that W_j^u is in general a multiset). Let

$$A_1(u) := \begin{cases} 0, & \text{if } \sum_{w\in W_1^u} A_0(w) > C \log n \\ 1 & \text{o.w.,} \end{cases} \tag{5.5}$$

and for $j = 2, \ldots, k-1$

$$A_j(u) := \begin{cases} 0, & \text{if } \sum_{w\in W_j^u} \prod_{i=0}^{j-2} A_i(w) > C \log n \\ 1 & \text{o.w.,} \end{cases} \tag{5.6}$$

where $A_i(w)$ in (5.6) are independent random variables and $C > 0$ is a sufficiently large constant. Note that in order to sample $A_j(u)$, one needs to

[2] It should be noted that we do not prove a formal correspondence between these random variables and the actual sets V_i in Algorithm 1, but instead prove directly that observing a sufficiently large number of such random variables is enough to approximate matching size.

sample $A_i(w), i = 0, \ldots, j-1$ for $w \in N(u)$. We will write $A \sim \mathcal{A}$ to indicate that random variable A is sampled from distribution $\mathcal{A}$. We denote the distribution of $A_j(u)$ by $\mathcal{A}_j(u)$ for $j = 0, \ldots, k-1$ and $u \in V$. It will be convenient to have the following notation for the marginals of $\prod_{i=0}^{j} A_i(u)$. For each $u \in V$ let

$$\nu_j(u) := \mathbf{Pr}\left[\prod_{i=0}^{j} A_i(u) = 1\right]. \tag{5.7}$$

We extend the definition of ν_j to edges $e \in E$ by letting $\nu_j(e) := \nu_j(u)\nu_j(v)$. For a function $f : U \to \mathbb{R}$, where $U = V$ or $U = E$, and a set $S \subseteq U$ we write $f(S) := \sum_{x \in S} f(u)$.

We have

Claim 5.1. *For all $u \in V, j \in [0 : k-1]$*

$$\nu_j(u) = \mathbf{Pr}[\textsc{GetLevel}(u, j) == j]$$

Proof. The claim follows by noting that the sampling process defined above exactly corresponds to the execution of GetLevel (in particular, we chose all cutoffs to match those in algorithms in the previous section exactly).

In what follows we will repeatedly use the following sampling process to reason about the distributions ν_j. For each node $u \in V$ sample random variables $A_i(u), i = 0, \ldots, k-1$ independently. For $j = 0, \ldots, k-1$ let $\hat{V}_j$ contain nodes u such that $\prod_{i=0}^{j} A_i(u) = 1$. Let $\hat{E}_j := E \cap (\hat{V}_j \times \hat{V}_j)$.

We start by proving bounds on the mass assigned to $N(u)$ by $\nu_i, i = 0, \ldots, j-1$ as a function of the sets $\hat{V}_j$ that u belongs to. These bounds will be crucial for relating level distributions ν_j to the size of the matching in the graph below.

Lemma 5.2. *Suppose that $\gamma d_{i-1}/d_i \geq C \log n$ for a sufficiently large constant $C > 0$ and all $i = 1, \ldots, k-1$ (**p5**), as guaranteed by Lemma 5.16. Let $A_i(u) \sim \mathcal{A}_i(u), i = 0, \ldots, k-1$ for $u \in V$, and let $\hat{V}_j, j \in [0 : k-1]$ be defined as above. Then the following assertions hold with probability at least $1 - n^{-10}$:*

(Upper bounds)

- **U1** $\nu_0(N(u)) \leq 2(C \log n)d_1$ *for all* $u \in \hat{V}_1$;
- **U2** $\nu_{i-1}(N(u)) \leq 2(C \log n)d_{i+1}$ *for all* $u \in \hat{V}_j, j \geq 2$ *and all* $i = 1, \ldots, j-1$.

(Lower bounds)

- **L1** $\nu_0(N(u)) \geq \frac{1}{2}(C \log n)d_1$ *for all* $u \in \hat{V}_0 \setminus \hat{V}_1$.
- **L2** $\nu_{j-1}(N(u)) \geq \frac{1}{2}(C \log n)d_{j+1}$ *for* $u \in \hat{V}_j \setminus \hat{V}_{j+1}$, *for all* $j \in [1 : k-2]$.

Remark 5.3. *Estimates from Lemma 5.2 will be useful in three distinct lemmas in what follows. Upper bounds* **U1** *and* **U2** *instantiated for $i = j-1$ will be used in Lemma 5.4 and subsequently in Lemma 5.6 to exhibit a large fractional matching in G in terms of ν_j's. Lower bounds* **L1-L2** *will be used to prove the converse, i.e. conclude that existence of a large matching implies that at least one of the ν_j's is large (see Lemma 5.7). Finally, upper bounds* **U2** *for all $i \in [0 : j-1]$ will be used to bound sampling complexity of our algorithm in Lemma 5.12.*

Proof of Lemma 5.2:

U1 and **L1** follow from the definition (5.5) of $A_1(u)$, as we now show. Recall that by (5.6) we have for a node $u \in \hat{V}_0$ that $u \in \hat{V}_1$ iff $A_1(u) = 1$, i.e. iff

$$\sum_{w \in W_1^u} A_0(w) \leq C \log n. \tag{5.8}$$

We denote the multiplicity of $w \in N(u)$ in W_1^u by λ_w. Since

$$\begin{aligned}\mathbf{E}\left[\sum_{w \in W_1^u} A_0(w)\right] &= \mathbf{E}\left[\sum_{w \in N(u)} \lambda_w A_0(w)\right] \\ &= \sum_{w \in N(u)} \mathbf{E}\left[\lambda_w\right] \nu_0(w) \\ &= \frac{R_1}{m} \cdot \nu_0(N(u)) = \nu_0(N(u))/d_1,\end{aligned}$$

where we used the fact that $A(w)$ is sampled independently of λ_w. Since $\nu_0(w) = A_0(w) \equiv 1$ for all w, **U1** and **L1** now follow by Chernoff bounds.

U2 and **L2** follow from the definition (5.6) of $A_j(u), j \geq 2$, as we now show. For a node $u \in \hat{V}_{j-1}$ one has that $u \in \hat{V}_j$ iff $A_j(u) = 1$, i.e. iff

$$\sum_{w \in N(u)} \lambda_w \cdot \prod_{i=0}^{j-2} A_i(w) \leq C \log n. \tag{5.9}$$

We have that λ_w are independent of $A_i(w)$ and satisfy

$$\mathbf{E}[\lambda_w] = R_j/m \tag{5.10}$$

for each $w \in N(V)$. Taking the expectation of (5.9) with respect to the choice of W_j^u and using (5.10) yields

$$\begin{aligned}
&\mathbf{E}\left[\sum_{w\in N(u)} \lambda_w \cdot \prod_{i=0}^{j-2} A_i(w)\right] \\
&= (R_j/m)\cdot \sum_{w\in N(u)} \mathbf{E}\left[\prod_{i=0}^{j-2} A_i(w)\right] \\
&= (R_j/m)\nu_{j-2}(N(u)) = \nu_{j-2}(N(u))/d_j
\end{aligned} \tag{5.11}$$

Now suppose for contradiction that $u \in \hat{V}_j$ and $\nu_{j-2}(N(u)) \geq 2(C\log n)d_j$. Then by the choice of the constant C we have using Chernoff bounds that $\sum_{w\in N(u)} \prod_{i=0}^{j-2} A_i(w) > C\log n$ with probability at least $1-n^{-10}$. Conditioned on this event, we have $u \notin \hat{V}_j$, leading to a contradiction. This yields **U2** for $i = j-1$. The claim for other values of i follows since the sets $\hat{V}_i$ are nested. Reversing the inequalities (with appropriately smaller constants on the rhs) yields **L2** instantiated for $u \in \hat{V}_{j-1}$. ■

In what follows we will need

LEMMA 5.4. (MAXIMUM DEGREE OF NODES IN $\hat{E}_j$) *Consider $j \in [0:k-1]$. Let $\hat{V}_j$ contain each $u \in V$ independently with probability $\nu_j(u)$. Then whp for each $u \in \hat{V}_j$ one has*

$$deg_{\hat{V}_j}(u) \leq d_j^{max},$$

where $d_0^{max} = d_0, d_j^{max} = 4(C\log n)d_j$ for $j \in [1:k-1]$.

Proof. Apply Chernoff bounds to **U1-U2** from Lemma 5.2. Note that we are only using the setting $i = j-1$ for **U2** here.

5.3 Level distributions and matching size

We now derive a connection between the distributions $\nu_j, j = 0,\ldots,k-1$ and the size of the maximum matching in G (recall that ν_j are defined by (5.7)).

Our proof proceeds as follows. We first sample a sequence of nested sets $\hat{V}_j, j = 0,\ldots,k-1$ using the distributions ν_j defined in the previous section. In particular, each node $u \in V$ samples $A_i(u)$ independently for $i = 0,\ldots,k-1$. A node u is then added to all sets $\hat{V}_j$ such that $\prod_{i=0}^{j} A_i(u) = 1$. Thus, for each $u \in V, j \in [0:k-1]$ we have $\mathbf{Pr}[u \in \hat{V}_j] = \nu_j$. Let $\hat{E}_j = E \cap (\hat{V}_j \times \hat{V}_j)$. We first show that the size of $\hat{E}_j$ is tightly concentrated around $\nu_j(E)$.

We will need the following simple

LEMMA 5.5. *Assume* **(p3)**, *as guaranteed by Lemma 5.16. For all $j = 0,\ldots,k-1$ if $\nu_j(E) \geq \tau_j$, then $|\hat{E}_j| \in (1\pm 1/4)\nu_j(E)$ with probability at least $1-n^{-7}$.*

The proof uses

THEOREM 5.2. (THEOREM 7.8.1 IN [3]) *Consider the polynomial*

$$Y = \sum_{e\in E}\prod_{i\in e} t_i,$$

where we have $\prod_{i\in e} t_i = 1$ when $e = \emptyset$. For a set $A \subseteq V$ let Y_A denote the derivative of Y with respect to the variables in A. Let $E_j := \max_{A\subseteq V, |A|\geq j} \mathbf{E}[Y_A]$. Let $t_i \in \{0,1\}$ be independent random variables. Then there exist constants $a_2, b_2 > 0$.

$$\mathbf{Pr}[|Y - \mathbf{E}[Y]| > a_2\lambda^2\sqrt{E_0E_1}] < b_2 e^{-\lambda/4+\log n}$$

Proof of Lemma 5.5: Recall that by Lemma 5.2 one has

$$\nu_j(u) \leq O(\log n)d_j$$

as long as $\nu_j(u) > n^{-10}$. Let $V^* = \{u \in V : \nu_j(u) > n^{-10}\}$, and let $\mathcal{E}$ denote the event that none of $u \in V\setminus V^*$ are sampled. One has $\mathbf{Pr}[\mathcal{E}] \geq 1-n^{-8}$ by the union bound.

Let $t_u, u \in V^*$ denote independent Bernoulli 0/1 variables with $\mathbf{E}[t_u] = \nu_j(u)$. Let

$$Y = \sum_{e=(u,v)\in E\cap V^*\times V^*} t_u t_v.$$

Note that $Y = |\hat{E}_j|$ and $\mathbf{E}[Y] = \nu_j(E)$, and we would like to prove that Y is tightly concentrated around its mean. We verify the conditions of Theorem 5.2. For a set $A \subseteq V^*$ let Y_A denote the derivative of Y with respect to the variables in A. Note that we can restrict our attention to singleton sets A since $Y_A = 0$ for $|A| > 2$ and $Y_A \leq 1$ when $|A| = 2$. It is easily seen that when $A = \{u\}$, we have $Y_A = \sum_{v\in N(u)} \nu_j(v)$, so that $\mathbf{E}[Y_A] = \nu_j(N(u)) = O(\log n)d_j$. Thus, in terms of the parameters of Theorem 5.2 we have $E_1 = O(\log d)d_j$ and $E_0 \geq \tau_j$, so $E_1 \leq (c^2\log^{-4} n)E_0$ for

a sufficiently small constant $c > 0$ by **p3**, so we have $\sqrt{E_0 E_1} \le c \log^{-2} n \nu_j(E)$. Finally, we have

$$\begin{aligned}
&\mathbf{Pr}[|\hat{E}_j| - \nu_j(E)| > (1/4)\nu_j(E)] \\
&\le \mathbf{Pr}[|\hat{E}_j| - \nu_j(E)| > a_2 \lambda^2 \sqrt{E_0 E_1}] \\
&\le \mathbf{Pr}[|\hat{E}_j| - \nu_j(E)| > a_2 \lambda^2 c \log^{-2} n \nu_j(E)] \\
&< b_2 e^{-\lambda/4 + \log n} \le n^{-8},
\end{aligned}$$

where we set $\lambda = c' \log n$ with a sufficiently large constant $c' > 0$. A union bound now gives the result. ∎

LEMMA 5.6. *Assume* **p4**, *as guaranteed by Lemma 5.16. If* $\nu_j(E) \ge (2/3)\tau_j$ *for some* $j \in [0 : k-1]$, *then* $\mathrm{OPT}(G) \ge U/g$ *with probability at least* $1 - n^{-10}$.

Proof. Let j be such that $\nu_j(E) \ge (2/3)\tau_j$. To prove the statement of the lemma, we exhibit a matching of size at least

$$\begin{aligned}
&(1/2)|\hat{E}_i|/d_i^{max} \ge (1/4)\nu_i(E)/d_i^{max} \\
&\ge (1/6)\tau_i(E)/d_i^{max} \ge U/g
\end{aligned}$$

in G, where we used Lemma 5.5 for the first inequality and property **p4** for the last inequality.

We exhibit a fractional matching of size at least $|\hat{E}_i|/d_i^{max}$ by placing weight $\frac{1}{d_i^{max}}$ on edges of $\hat{E}_i$. We need to verify feasibility. For bipartite graphs it is sufficient to check that the total weight incident on every node is at most 1. For $u \in \hat{V}_i$ we have $\deg_{\hat{V}_i}(u) \le d_i^{max}$ by Lemma 5.4 whp, so the total weight incident on a node is at most 1. This verifies feasibility for bipartite graphs. For non-bipartite graphs, it is sufficient to consider the restriction of the fractional matching we constructed to the edges going across a uniformly random bipartition of G. Such a matching will be of size at least $(1/2)|\hat{E}_i|/d_i^{max}$ in expectation over the bipartition. Thus, there exists at least one bipartition that yields a matching of size at least $(1/2)|\hat{E}_i|/d_i^{max}$, completing the proof.

LEMMA 5.7. *Let parameters be set as in Lemma 5.16. Suppose that* $G = (V, E)$ *contains a matching of size at least* U. *Then either*

[A] *there exists* $i = 0, \ldots, k-3$ *such that* $\nu_i(E) \ge \tau_i$ *or*

[B] $\nu_{k-2}(E) \ge \tau_{k-2}$.

Proof. Sample random sets $\hat{V}_j$ as described above. By Lemma 5.2 one has whp

1. $\nu_0(N(u)) \ge d_1$ for $u \in \hat{V}_0 \setminus \hat{V}_1$ by Lemma 5.2, **L1**
2. $\nu_{i-1}(N(u)) \ge \frac{1}{2}(C \log n) d_{i+1}$ for $u \in \hat{V}_i \setminus \hat{V}_{i+1}$ for $i = 1, \ldots, k-2$ by Lemma 5.2, **L2**;

Recall that by definition of $\hat{V}_i$ one has $\mathbf{E}[\deg_{\hat{V}_i}(u)] = \nu_i(N(u))$ for all $u \in V$ and $i \in [0 : k-2]$. We now have whp by Chernoff bounds (1) $\deg_{\hat{V}_0}(u) \ge d_1/2 \ge \frac{1}{4}(C \log n) d_2$ for $u \in \hat{V}_0 \setminus \hat{V}_1$ (since $d_1 \ge (\log^4 n) d_2$ by **(p1)**) and (2) $\deg_{\hat{V}_{i-1}}(u) \ge \frac{1}{4}(C \log n) d_{i+1}$ for $u \in \hat{V}_i \setminus \hat{V}_{i+1}$ for $i \in [1 : k-2]$. Thus,

$$|\hat{E}_{i-1}| \ge \frac{1}{4}(C \log n) d_{i+1} \cdot |\hat{V}_i \setminus \hat{V}_{i+1}| \tag{5.12}$$

for $i \in [1 : k-2]$ whp. Now if

$$\sum_{i=0}^{k-2} |\hat{V}_i \setminus \hat{V}_{i+1}| = |V| - |\hat{V}_{k-1}| \ge U/2, \tag{5.13}$$

we get using (5.12) that

$$|\hat{E}_i| \ge \frac{U/2}{k} \frac{1}{4}(C \log n) d_{i+2} \ge \tau_i$$

for some $i \in [0 : k-3]$, where we used the definition of τ_i (see Lemma 5.16).

Now suppose that (5.13) does not hold. Then since G has a matching of size at least U by assumption, removing at most $U/2$ nodes with all incident edges leaves at least $U/2$ edges in $\hat{E}_{k-1}$, yielding $|\hat{E}_{k-2}| \ge |\hat{E}_{k-1}| \ge \tau_{k-2}$ by the choice of τ_{k-2}(see Lemma 5.16), as required.

5.4 Correctness of GapMatching with access to unlimited samples We assume for the purposes of this section that Algorithm 2 has access to a stream of i.i.d. samples of edges of G of unlimited length. We will show later in sections 5.5 and 5.6 that Algorithm 2 consumes $O(m/\log^2 n)$ samples whp. We first show that if Algorithm 2 outputs **YES** , then we are not in the **NO** case with high probability. We then show that if the answer is **YES** , then the algorithm outputs **YES** whp (Lemma 5.10) The following claim will be useful:

CLAIM 5.8. *If* $\alpha_L \ge (9/10) C \log n$, *then* $\alpha_L \cdot \frac{m}{Q_L} \in (1 \pm 1/10)\nu_L(E)$ *with probability at least* $1 - n^{-10}$.

Proof. This follows by the choice of C and Chernoff bounds.

We now have

LEMMA 5.9. *Assuming* **p1-p5**, *if Algorithm 2 outputs* **YES** , *then whp* $\mathrm{OPT}(G) \geq U/g$.

Proof. Recall that we have $Q_L = (C \log n)m/\tau_L$. Algorithm 2 only outputs **YES** if $\alpha_L > (9/10)C \log n$. By Claim 5.8 we have $\alpha_L \cdot m/Q_L \leq (10/9)\nu_L(E)$ whp, which implies that

$$\nu_L(E) \geq (9/10)(\alpha_L/(C \log n)) \cdot \tau_L \geq (9/10)^2 \tau_L \geq (2/3)\tau_L$$

We then have $\mathrm{OPT}(G) \geq U/g$ by Lemma 5.6, completing the proof.

It remains to show that if there exists a matching of size at least U, then Algorithm 2 outputs **YES** .

LEMMA 5.10. *Suppose that $G = (V, E)$ contains a matching of size at least U. Then with probability at least $1 - n^{-10}$ one has at the end of execution of Algorithm 2 that $\alpha_L \geq (9/10)C \log n$, so the algorithm outputs* **YES** .

Proof. Let $L \in [0 : k-2]$ be such that $\nu_L(E) \geq \tau_L$ (existence of such L is guaranteed by Lemma 5.7). Consider the parallel execution of the **for** loop in Algorithm 2 that has $L = j$. Let E^* denote the set of edges that is tested in this parallel execution. For $j = 0, \ldots, L$ let $\tilde{E}_j \subseteq E^*$ denote the number of edges $e = (u, v) \in E^*$ that are determined to be at level at least j. Since each such edge e is uniformly random edge of G, one has

$$\mathbf{E}[|\tilde{E}_j|] = \frac{1}{m}\nu_j(E) \cdot Q. \tag{5.14}$$

Using this in (5.14), we get

$$\mathbf{E}[|\tilde{E}_L|] \geq \frac{1}{m} \cdot Q \cdot \nu_L(E) \geq \frac{1}{m} \cdot Q \cdot \tau_L = C \log n. \tag{5.15}$$

Thus, with probability $1 - n^{-10}$ one has that $\alpha_L \geq (9/10)(C \log n)$ at the end of the run of the algorithm, and hence the algorithm outputs **YES**.

LEMMA 5.11. *Assuming* **p1-p5**, *as guaranteed by Lemma 5.16. Then Algorithm 2 solves the* $\mathsf{Gap\text{-}Matching}_{U,g}$ *problem whp, assuming access to an unlimited number of samples, for any U, g greater than a poly*$(\log n)$.

Proof. Lemma 5.9 shows that the algorithm outputs **YES** in the **NO** case only with polynomially small probability. Lemma 5.10 shows that the algorithm outputs **YES** in the **YES** case whp.

5.5 Sampling complexity of GetLevel In this section we analyze the sampling complexity of GETLEVEL. Recall that GETLEVEL(u, r) runs V_j-tests on u for $j \leq r-2$ and stops as soon as at least one test fails. Thus, the number of samples used by the call depends on the answer returned as well as the parameter r passed. To this end, we let ℓ_j denote the number of edges of G sampled by GETLEVEL(u, r) for $r \geq j$ *if no* $V_{j'}$-TEST *is run during the execution for any* $j' > j$. Note that this happens, for example, whenever the returned answer is at most $j-1$ (and, of course, when $r \leq j$). We will exhibit functions $T_j, f_j, j = 1, \ldots, k-1$ such that

$$\mathbf{Pr}[\ell_j > T_j] < f_j.$$

In what follows we show that any T_j, f_j that satisfy

$$\begin{aligned} &T_1 \geq m/d_1 \\ &f_1 \geq 0 \\ &T_j \geq m/d_j + \sum_{i=1}^{j-1}(4\gamma d_i/d_j + 1) \cdot T_i, \\ &\quad j \in [2 : k-1] \\ &f_j \geq O(n^{-10}) + \sum_{i=1}^{j-1}(4\gamma d_i/d_j + 1) \cdot f_i, \\ &\quad j \in [2 : k-1] \end{aligned} \tag{5.16}$$

give a valid bound.

LEMMA 5.12. *Suppose that $\gamma d_{j-1}/d_j = C \log n$, $j \in [1 : k-1]$ (***p5***), as guaranteed by Lemma 5.16. Let ℓ_j be defined as above, and let T_j, f_j satisfy* (5.16). *Then* $\mathbf{Pr}[\ell_j > T_j] < f_j$.

Proof. The proof is by induction on j.

Base: $j = 1$ The test is non-recursive and examines exactly $R_1 = m/d_1$ edges.

Inductive step: $j - 1 \to j$ We will bound T_j in terms of $T_1, \ldots, T_{j-1}$ and $f_1, \ldots, f_{j-1}$. Note that we have $j \geq 2$ here.

Recall (see Algorithm 3) that a call to GETLEVEL(u, j) starts by calling

GetLevel$(u, j-1)$. Suppose that this invocation returned $j-1$. Then Algorithm 3 calls the function $V_{j'}$-Test(u) for $j'=j$ but not for any larger j'. The latter function samples R_j edges in the stream and invokes GetLevel$(w, j-2)$ on the other endpoints of edges in this sample that are incident on u. We will bound the sampling complexity of V_j-Test in terms of $T_{j'}, j'<j$. This will in turn allow us to bound the sampling complexity of GetLevel(u,j). The crux of the proof is bounding the number of calls to GetLevel$(w, j-2)$ from a call to V_j-Test that return j' for $j' \le j-2$. Here we use two bounds: for $j'<j-2$ the number of such calls is bounded using Lemma 5.2, and for $j'=j-2$ the number of such calls is bounded by $C\log n$ by the definition of the test. We start with the first bound.

By Lemma 5.2, **U2** and the assumption $\gamma d_{s-1}/d_s = C\log n, s\in[1:k-1]$ one has

$$\nu_{i-1}(N(u)) \le 2(C\log n)d_{i+1} = 2\gamma d_i$$

for all $i=1,\dots,j-2$ with probability at least $1-n^{-10}$ over the randomness in the invocation of GetLevel (note that this is only non-trivial for $j\ge 3$). For each $j'\in[0:j-1]$ let $q_{j'}$ denote the number of neighbors of u whose invocation of GetLevel returned at least $j'-1$ (i.e. involved a call to $V_{j'}$-Test). For $w\in N(u)$ let λ_w denote the number of times w appears in the set of nodes tested in V_j-Test(u,r).

For each $i=1,\dots,j-2$ one has

$$\begin{aligned}\mathbf{E}[q_i] &= R_j\cdot\mathbf{E}\left[\sum_{w\in N(u)}\lambda_w\cdot\nu_{i-1}(w)\right]\\ &\le \frac{1}{m}R_j\cdot 2\gamma d_i = 2\gamma d_i/d_j\end{aligned} \tag{5.17}$$

where we used the definition of R_j in (5.4). We have that $\nu_{j-1}(w)\in[0,1]$ for all $w\in V$ by definition, and the rhs of (5.17) is at least $C\log n$, so Chernoff bounds imply that for $i=1,\dots,j-2$

$$q_i \le 4\gamma d_i/d_j \tag{5.18}$$

with probability at least $1-n^{-10}$. Furthermore, recall that the algorithm terminates and outputs 0 if more than $C\log n = \gamma d_{j-1}/d_j$ nodes w that have been tested are placed at level at least $j-2$, so the number of calls to GetLevel$(w,j-1)$ that return $j-2$ is bounded by $\gamma d_{j-1}/d_j$. Combining this with (5.18), we get by the inductive hypothesis that the number of edges used by V_j-Test is bounded by

$$\begin{aligned}&R_j + \sum_{i=1}^{j-2} q_i\cdot T_i + (\gamma d_{j-1}/d_j)|T_{j-1}|\\ &\le R_j + \sum_{i=1}^{j-1}(4\gamma d_i/d_j)\cdot T_i\end{aligned}$$

with probability at least

$$1-\left(O(n^{-10}) + \sum_{i=1}^{j-1}(4\gamma d_i/d_j)\cdot f_i\right).$$

where the probability is taken over the randomness used in recursive calls. Further, note that the call to GetLevel$(u,j-1)$ takes at most T_{j-1} time since no $V_{j'}$-Test, $j'\ge j$ is run. Taking this into account, we conclude that the sampling complexity is bounded by

$$\begin{aligned}&T_{j-1} + \left(R_j + \sum_{i=1}^{j-1}(4\gamma d_i/d_j)\cdot T_i\right)\\ &\le m/d_j + \sum_{i=1}^{j-1}(4\gamma d_i/d_j + 1)\cdot T_i.\end{aligned}$$

with probability at least

$$1-\left(O(n^{-10}) + \sum_{i=1}^{j-1}(4\gamma d_i/d_j + 1)\cdot f_i\right),$$

as required.

Finally, we show

Lemma 5.13. *Suppose that $\gamma d_{j-1}/d_j \ge C\log n$ for all $j\in[1:k-1]$ (**p5**), as guaranteed by Lemma 5.16. Then $d_jT_j = m\cdot(1+8k\gamma)^{j-1}$ and $d_jf_j = (1+8k\gamma)^{j-1}d_0n^{-10}$ yield a valid solution to (5.16) for any d_i as long as $k=O(\log n)$.*

Proof. This follows directly from (5.16). Indeed, multiplying both sides by d_j and replacing inequal-

ities with equalities, we get

$$
\begin{aligned}
d_j T_j &= m + \sum_{i=1}^{j-1} (4\gamma d_i + d_j) \cdot T_i \\
d_j f_j &= d_j n^{-10} + \sum_{i=1}^{j-1} (4\gamma d_i + d_j) \cdot f_i \\
T_1 &= m/d_1, f_1 \geq 0
\end{aligned} \tag{5.19}
$$

We now upper bound the growth of the solution to (5.19). In particular, we get

$$
\begin{aligned}
d_j T_j &\leq m + \sum_{i=1}^{j-1} (4\gamma d_i + d_j) \cdot T_i \\
&\leq m + \sum_{i=1}^{j-1} 8\gamma d_i T_i,
\end{aligned}
$$

where we used $\gamma d_{j-1}/d_j \geq C \log n$ to upper bound d_j with $4\gamma d_{j-1}$. Substituting $d_j T_j = m \cdot (1+8k\gamma)^{j-1}$ into the rhs yields

$$
\begin{aligned}
& m + m \sum_{i=1}^{j-1} 8\gamma (1+8k\gamma)^{i-1} \\
\leq & m + m 8k\gamma (1+8k\gamma)^{j-2} \\
\leq & m(1+8k\gamma)^{j-1}
\end{aligned}
$$

as required. A similar calculation yields the bound on f_j.

REMARK 5.14. *Lemma 5.13 suggests that sampling complexity can be reduced by increasing d_j, at the expense of a loss in the approximation factor.*

5.6 Sampling complexity of GapMatching

LEMMA 5.15. *Assume* **p1, p2**, *as guaranteed by Lemma 5.16. Let $L \in [0 : k-2]$ be such that for each $j \in [0 : L]$ one has $\nu_j(E) \leq \tau_j$. Then the total number of edges sampled by V_j-tests run in loop L in Algorithm 2 is $O(m/\log^2 n)$.*

Proof. Consider any parallel loop in Algorithm 2. Let E^* denote the set of edges that is tested in such a parallel loop. For $j = 0, \ldots, L$ let $\tilde{E}_j \subseteq E^*$ denote the set of edges $e = (u, v) \in E^*$ such that are determined to be at level at least j.

First note that for each of u, v the algorithm runs V_i-tests starting with $i = 1$ until one of the tests fails or until $i = L$. Let j be the largest such that a V_j-test is run on either u or v. Note that the sequence of calls to V_j-TEST corresponds exactly to a call to GETLEVEL(u, L) (resp. GETLEVEL(v, L)) conditioned on outputting a level no larger than j. By Lemma 5.12 the number of samples that GETLEVEL(u, L) consumes conditioned on outputting a level no larger than j is bounded by T_{j+1} with probability $1 - f_{j+1}$, where T_j, f_j are given in Lemma 5.13. Thus, the number of samples it takes to test an edge in $|\tilde{E}_j \setminus \tilde{E}_{j+1}|$ is bounded by $2T_{j+1}$, for all $j \in [1 : k-1]$. We now upper bound the number of samples used.

First suppose that $L < k-2$. Then $\tau_j = \frac{U/2}{k} \frac{1}{4} \gamma d_{j+1}$ for all $j \in [0 : L]$. Note that $|\tilde{E}_j| = O(\tau_j Q_L/m)$ with probability at least $1 - 1/\text{poly}(n)$ by Chernoff bounds, so we get

$$
\begin{aligned}
\sum_{j=0}^{L} |\tilde{E}_j|(2T_{j+1}) &= \sum_{j=0}^{L} O(\tau_j Q_L/m) T_{j+1} \\
&= (C \log n) \sum_{j=0}^{L} (\tau_j/\tau_L) T_{j+1} \\
&= (C \log n) \sum_{j=0}^{L} (d_{j+1}/d_{L+1}) \frac{m}{d_{j+1}} \\
&= (C \log n) L m / d_{L+1},
\end{aligned} \tag{5.20}
$$

except with probability at most $\sum_{j=0}^{L} |\tilde{E}_j|(2f_{j+1}) = 1/\text{poly}(n)$. Furthermore, one has $(C \log n) L m / d_{L+1} = O(m/\log^2 n)$ by **p1**, giving the desired result for $L < k-2$.

Now suppose that $L = k-2$. Then $\tau_L = U/2$, and we have

$$
\begin{aligned}
\sum_{j=0}^{L} |\tilde{E}_j|(2T_{j+1}) &= \sum_{j=0}^{L} O(\tau_j Q_L/m) T_{j+1} \\
&= (C \log n) \sum_{j=0}^{L} (\tau_j/\tau_L) T_{j+1} \\
&= (C \log n) \sum_{j=0}^{L} (4\gamma d_{j+1})(m/d_{j+1}) \\
&= \gamma (C \log n) L m,
\end{aligned} \tag{5.21}
$$

which is $O(m/\log^2 n)$ by **p2**. The upper bound on failure probability in this case is analogous to the above.

5.7 Putting It Together We now argue that whp Algorithm 2 correctly solves Gap-Matching and consumes only $O(m/\log^2 n)$ i.i.d. edge samples. Our first step towards this goal is to exhibit a setting of parameters that satisfies **p1-p5**:

LEMMA 5.16. *Let $U \geq \log^c n$ for a sufficiently large constant $c > 0$, and let $g \geq \log^{c'} n$ constant $c' > 0$. Then there exists a setting of parameters that satisfies* **p1-p5**.

Proof. First, let $d_0 = U$, let $d_{k-1} = \log^A n$, and let $d_i = (d_{k-1})^{k-i}, i = k-2, \ldots, 1$, where $k = \lfloor \log_{\log^A n} U \rfloor$. We also have $\gamma d_{j-1}/d_j = C \log n$ for all $j \in [2 : k-1]$ when $\gamma = C/\log^{A-1} n$.

This allows us to derive a convenient expression for d_j^{max}. Recall that by Lemma 5.4 one has $d_0^{max} = d_0 = n, d_j^{max} = 4(C \log n) d_j$ for $j \in [1 : k-1]$. We also let

$$\tau_j = \frac{U/2}{k}\frac{1}{4}\gamma d_{j+1} = \frac{U/2}{k}\frac{1}{4}(C \log n) d_{j+2}$$

for $j \in [0 : k-3]$, and let $\tau_{k-2} = U/2$.

We now prove that the setting of parameters above satisfies **p1-p5**. We have

p1 $d_{k-1} \geq C' \log^4 n$ for a sufficiently large constant $C' > 0$ by choice of d_{k-1} above.

p2 $\gamma \leq C'' \log^{-4} n$ for a sufficiently large constant $C' > 0$ by choice of γ above.

p3 $\tau_j/(C' d_j \log^4 n) \geq C \log n$ for all $j \in [0 : k-2]$ is satisfied since

$$\begin{aligned}\tau_j/(C' d_j \log^4 n) &= (U/(8k))\gamma d_{j+1}/(C' d_j \log^4 n)) \\ &\geq C \log n\end{aligned}$$

as long as $U \geq 8k(\gamma d_{j+1}/(C' d_j \log^4 n))^{-1} = \Omega(\text{poly}(\log n))$

p4 $\frac{1}{6}\tau_j/d_j^{max} \geq U/g$ is satisfied for all $j \in [0 : k-2]$ since

$$\frac{1}{6}\tau_j/d_j^{max} = \Omega(U\gamma d_{j+1}/(4\gamma d_{j-1})) \geq U/g$$

as long as g is greater than a $\text{poly}(\log n)$.

p5 $\gamma d_{j-1}/d_j = C \log n$ for all $j \in [1 : k-1]$ by our choice of γ.

We are now ready to complete the proof of the main result of this section.

Proof of Theorem 5.1: We assume **p1-p5**, as guaranteed by Lemma 5.16. If $\nu_j(E) \leq \tau_j$ for all $j \in [0 : k-2]$, then by Lemma 5.15 each parallel loop in Algorithm 2 consumes $O(m/\log^2 n)$ samples. In this case correctness follows by Lemma 5.11.

Now suppose that $\nu_L(E) \geq \tau_L$. Let L be the smallest index with this property. Consider the L-th parallel loop in Algorithm 2. Then the number of samples consumed by tests run on $\hat{E}_j, j \leq L-1$ is bounded by $O(m/\log^2 n)$ by Lemma 5.15. On the other hand, we get by the same Lemma 5.15 that $O(m/\log^2 n)$ samples are sufficient to determine that $\nu_j(E) \geq \tau_j$, so the algorithm outputs **YES** . ■

6 Analysis for the Random Permutation Model

We now show that Algorithm 2 correctly solves the $\text{Gap-Matching}_{U,g}$ problem with access to a random stream of the edges of graph G, for $g = \text{poly}(\log n)$. We show this by establishing the following theorem which asserts a coupling between the behavior of Algorithm 2 in the i.i.d. case and the random permutation case.

THEOREM 6.1. *Let $ALG^{IID}(G)$ denote the answer output by Algorithm 2 when run on an stream of m i.i.d. samples of edges of G. Let $ALG^{\pi}(G)$ denote the answer output by Algorithm 2 when run on a stream of randomly permuted edges of G. Then*

$$\mathbf{Pr}[ALG^{IID}(G) \neq ALG^{\pi}(G)] \leq 1/poly(\log n),$$

where n is the number of vertices in G.

We defer the proof of the above theorem to the full version of the paper.

We can now wrap up the proof of the main result of our paper:

Proof of Theorem 1.1: Recall that the problem of approximating the maximum matching size to within a poly-logarithmic factor can be solved by $O(\log n)$ instances of Gap-Matching for geometrically decreasing values of U (see Section 2).

Combining Theorem 6.1 with Theorem 5.1, we get that Algorithm 2 correctly solves the Gap-Matching with probability at least $1 - 1/\text{poly}(\log n)$, in poly-logarithmic space, when edges of the graph are presented in random order. Since we need $O(\log n)$ invocations of Gap-Matching we get the desired result by applying a union bound over all invocations. ■

7 Concluding Remarks

Our algorithm relies crucially on the ability to sample the edges in a random order. An interesting direction for future work is to explore if one can design an algorithm with similar parameters (polylogarithmic space and approximation) when the edges are presented in an arbitrary (adversarial) order.

In order to reduce the space complexity of estimating the matching size to being below linear, we were forced to weaken the approximation guarantee from constant to polylogarithmic. It would be interesting to see if one can derive a constant factor approximation algorithm using polylogarithmic (or even n^{ϵ}) space.

Finally, the possibility of obtaining a better than $1 - 1/e$ approximation in polylogarithmic space in adversarially ordered streams is precluded by lower bounds based on Ruzsa-Szemerédi graph constructions [7, 10]. It is conceivable that the adversarial setting also precludes achieving a weaker guarantee of polylogarithmic approximation in polylogarithmic space. However, existing techniques do not seem to lend easily to answer this question and it will be very useful (quite possibly for other related problems) to develop tools needed to make progress on this front.

8 Acknowledgements

We would like to thank Ryan O'Donnell and Rocco Servedio for pointers to the literature on concentration of multi-variate polynomials.

References

[1] K. Ahn and S. Guha. Linear programming in the semi-streaming model with application to the maximum matching problem. *ICALP*, pages 526–538, 2011.

[2] K. Ahn and S. Guha. Access to data and number of iterations: Dual primal algorithms for maximum matching under resource constraints. *CoRR*, abs/1307.4359, 2013.

[3] N. Alon and J. Spencer. *The probabilistic method*. Wiley, 2008.

[4] Noga Alon, Ronitt Rubinfeld, Shai Vardi, and Ning Xie. Space-efficient local computation algorithms. *SODA*, pages 1132–1139, 2012.

[5] Sebastian Eggert, Lasse Kliemann, and Anand Srivastav. Bipartite graph matchings in the semi-streaming model. In Amos Fiat and Peter Sanders, editors, *Algorithms - ESA 2009*, volume 5757 of *Lecture Notes in Computer Science*, pages 492–503. Springer Berlin / Heidelberg, 2009.

[6] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348:207–216, 2005.

[7] A. Goel, M. Kapralov, and S. Khanna. On the communication and streaming complexity of maximum bipartite matching. *SODA*, 2012.

[8] Venkatesan Guruswami and Krzysztof Onak. Superlinear lower bounds for multipass graph processing. *CCC*, 2012.

[9] Zengfeng Huang, Božidar Radunović, Milan Vojnović, and Qin Zhang. Communication complexity of approximate maximum matching in distributed graph data. *MSR Technical Report*, 2013.

[10] Michael Kapralov. Better bounds for matchings in the streaming model. *SODA*, 2013.

[11] Chinmay Karande, Aranyak Mehta, and Pushkar Tripathi. Online bipartite matching with unknown distributions. *STOC*, pages 587–596, 2011.

[12] R. Karp, U. Vazirani, and V. Vazirani. An optimal algorithm for online bipartite matching. *STOC*, 1990.

[13] Christian Konrad, Frédéric Magniez, and Claire Mathieu. Maximum matching in semi-streaming with few passes. *APPROX-RANDOM*, 2012.

[14] Mohammad Mahdian and Qiqi Yan. Online bipartite matching with random arrivals: an approach based on strongly factor-revealing lps. *STOC*, pages 597–606, 2011.

[15] Yishay Mansour and Shai Vardi. A local computation approximation scheme to maximum matching. *CoRR*, abs/1306.5003, 2013.

[16] A. McGregor. Finding graph matchings in data streams. *APPROX-RANDOM*, pages 170–181, 2005.

[17] Huy N. Nguyen and Krzysztof Onak. Constant-time approximation algorithms via local improvements. *FOCS*, pages 327–336, 2008.

[18] Krzysztof Onak, Dana Ron, Michal Rosen, and Ronitt Rubinfeld. A near-optimal sublinear-time algorithm for approximating the minimum vertex cover size. *SODA*, pages 1123–1131, 2012.

[19] Yuichi Yoshida, Masaki Yamamoto, and Hiro Ito. An improved constant-time approximation algorithm for maximum matchings. *STOC*, pages 225–234, 2009.