Space- and Time-Efficient Algorithm for Maintaining Dense Subgraphs on One-Pass Dynamic Streams

Sayan Bhattacharya* Monika Henzinger† Danupon Nanongkai‡
Charalampos E. Tsourakakis§

Abstract

While in many graph mining applications it is crucial to handle a stream of updates efficiently in terms of *both* time and space, not much was known about achieving such type of algorithm. In this paper we study this issue for a problem which lies at the core of many graph mining applications called *densest subgraph problem*. We develop an algorithm that achieves time- and space-efficiency for this problem simultaneously. It is one of the first of its kind for graph problems to the best of our knowledge.

Given an input graph, the densest subgraph is the subgraph that maximizes the ratio between the number of edges and the number of nodes. For any $\epsilon > 0$, our algorithm can, with high probability, maintain a $(4+\epsilon)$-approximate solution under edge insertions and deletions using $\tilde{O}(n)$ space and $\tilde{O}(1)$ amortized time per update; here, n is the number of nodes in the graph and $\tilde{O}$ hides the $O(\text{poly}\log_{1+\epsilon} n)$ term. The approximation ratio can be improved to $(2+\epsilon)$ with more time. It can be extended to a $(2+\epsilon)$-approximation sublinear-time algorithm and a distributed-streaming algorithm. Our algorithm is the first streaming algorithm that can maintain the densest subgraph in *one pass*. Prior to this, no algorithm could do so even in the special case of incremental stream and even when there is no time restriction. The previously best algorithm in this setting required $O(\log n)$ passes [Bahmani, Kumar and Vassilvitskii, VLDB'12]. The space required by our algorithm is tight up to a polylogarithmic factor.

*The Institute of Mathematical Sciences, Chennai, India. Part of this work was done while the author was in Faculty of Computer Science, University of Vienna, Austria.

†Faculty of Computer Science, University of Vienna, Austria. The research leading to these results has received funding from the European Unions Seventh Framework Programme (FP7/2007-2013) under grant agreement 317532 and from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC Grant Agreement number 340506.

‡KTH Royal Institute of Technology, Sweden. Part of this work was done while the author was in Faculty of Computer Science, University of Vienna, Austria.

§Harvard University, School of Engineering and Applied Sciences.

Contents

Part I
EXTENDED ABSTRACT

1 Introduction

In analyzing large-scale rapidly-changing graphs, it is crucial that algorithms must use small space and adapt to the change quickly. This is the main subject of interest in at least two areas, namely *data streams* and *dynamic algorithms*. In the context of graph problems, both areas are interested in maintaining some graph property, such as connectivity or distances, for graphs undergoing a stream of edge insertions and deletions. This is known as the (one-pass) *dynamic semi-streaming* model in the data streams community, and as the *fully-dynamic* model in the dynamic algorithm community.

The two areas have been actively studied since at least the early 80s (e.g. [15, 28]) and have produced several sophisticated techniques for achieving time and space efficiency. In dynamic algorithms, where the primary concern is *time*, the heavy use of *amortized analysis* has led to several extremely fast algorithms that can process updates and answer questions in a poly-logarithmic amortized time. In data streams, where the primary concern is *space*, the heavy use of *sampling* techniques to maintain small *sketches* has led to algorithms that require space significantly less than the input size; in particular, for dynamic graph streams the result by Ahn, Guha, and McGregor [1] has demonstrated the power of linear graph sketches in the dynamic model, and initiated an extensive study of dynamic graph streams (e.g. [1–3, 22, 23]). Despite numerous successes in these two areas, we are not aware of many results that combine techniques from *both* areas to achieve time- and space-efficiency *simultaneously* in dynamic graph streams. A notable exception we are aware of is the connectivity problem, where one can combine the space-efficient streaming algorithm of Ahn et al. [2] with the fully-dynamic algorithm of Kapron et al. [24][1].

In this paper, we study this issue for the *densest subgraph* problem. For any unweighted undirected graph G, the density of G is defined as $\rho(G) = |E(G)|/|V(G)|$. The densest subgraph of G is the subgraph H that maximizes $\rho(H)$, and we denote the density of such subgraph by $\rho^*(G) = \max_{H \subseteq G} \rho(H)$. For any $\gamma \geq 1$ and ρ', we say that ρ' is an *γ-approximate value* of $\rho^*(G)$ if $\rho^*(G)/\gamma \leq \rho' \leq \rho^*(G)$. The (static) densest subgraph problem is to compute or approximate ρ^* and the corresponding subgraph. Throughout, we use n and m to denote the number of nodes and edges in the input graph, respectively.

This problem and its variants have been intensively studied in practical areas as it is an important primitive in analyzing massive graphs. Its applications range from identifying dense communities in social networks (e.g. [13]), link spam detection (e.g. [16]) and finding stories and events (e.g. [4]); for many more applications of this problem see, e.g., [6, 26, 35, 36]. Goldberg [18] was one of the first to study this problem although the notion of graph density has been around much earlier (e.g. [25, Chapter 4]). His algorithm can solve this problem in polynomial time by using $O(\log n)$ flow computations. Later Gallo, Grigoriadis and Tarjan slightly improved the running time using parametric maximum flow computation. These algorithms are, however, not very practical, and an algorithm that is more popular in practice is an $O(m)$-time $O(m)$-space 2-approximation algorithm of Charikar [9]. However, as mentioned earlier, graphs arising in modern applications are huge and keep changing. This algorithm is not suitable to handle such graphs. Consider, for example, an application of detecting a dense community in social networks. Since people can make new friends as well as "unfriend" their old friends, the algorithm must be able to process these updates efficiently. With this motivation, it is natural to consider the dynamic version of this problem. To be precise, we define the problem following the dynamic algorithms literature as follows. We say that an algorithm is a *fully-dynamic γ-approximation* algorithm for the densest subgraph problem if it can process the following operations.

[1] We thank Valerie King (private communication) for pointing out this fact.

- INITIALIZE(n): Initialize the algorithm with an empty n-node graph.
- INSERT(u, v): Insert edge (u, v) to the graph.
- DELETE(u, v): Delete edge (u, v) from the graph.
- QUERYVALUE: Output a γ-approximate value of $\rho^*(G)$.[2]

The *space complexity* of an algorithm is defined to be the space needed in the worst case. We define *time complexity* separately for each type of operations: Time for the INITIALIZE operation is called *preprocessing time*, time to process each INSERT and DELETE operation is called *update time*, time for answering each QUERY operation is called *query time*. For any τ, we say that an algorithm has an *amortized* update time τ if the total time it needs to process any k insert and delete operations is at most $k\tau$.

Our Results. Our main result is an efficient $(4+\epsilon)$-approximation algorithm for this problem, formally stated as follows. For every integer $t \geq 0$, let $G^{(t)} = (V, E^{(t)})$ be the state of the input graph $G = (V, E)$ just after we have processed the first t updates in the dynamic stream, and define $m^{(t)} \leftarrow |E^{(t)}|$. We assume that $m^{(0)} = 0$ and $m^{(t)} > 0$ for all $t \geq 1$. Let $\text{OPT}^{(t)}$ denote the density of the densest subgraph in $G^{(t)}$.

Theorem 1.1. *Fix some small constant $\epsilon \in (0, 1)$, a constant $\lambda > 1$, and let $T = \lceil n^\lambda \rceil$. There is an algorithm that processes the first T updates in the dynamic stream using $\tilde{O}(n)$ space and maintains a value $\text{OUTPUT}^{(t)}$ at each $t \in [T]$. The algorithm gives the following guarantees with high probability: We have $\text{OPT}^{(t)}/(4+O(\epsilon)) \leq \text{OUTPUT}^{(t)} \leq \text{OPT}^{(t)}$ for all $t \in [T]$. Further, the total amount of computation performed while processing the first T updates in the dynamic stream is $O(T \; poly \log n)$.*

We note that our algorithm can be easily extended to output the set of nodes in the subgraph whose density $(4+\epsilon)$-approximates $\rho^*(G)$ using $O(1)$ time per node. As a by product of our techniques, we obtain some additional results.

• **A $(2+\epsilon)$-approximation one-pass dynamic semi-streaming algorithm:** This follows from the fact that with the same space, preprocessing time, and update time, and an additional $\tilde{O}(n)$ query time, our main algorithm can output a $(2+\epsilon)$-approximate solution. See Section 3.

• **Sublinear-time algorithm:** We show that Charikar's linear-time linear-space algorithm can be improved further! In particular, if the graph is represented by an incident list (this is a standard representation [10, 17]), our algorithm needs to read only $\tilde{O}(n)$ edges in the graph (even if the graph is dense) and requires $\tilde{O}(n)$ time to output a $(2+\epsilon)$-approximate solution. We also provide a lower bound that matches this running time up to a poly-logarithmic factor. See Appendix A.

• **Distributed streaming algorithm:** In the distributed streaming setting with k sites as defined in [11], we can compute a $(2+\epsilon)$-approximate solution with $\tilde{O}(k+n)$ communication by employing the algorithm of Cormode et al. [11]. See Appendix B.

To the best of our knowledge, our main algorithm is the first dynamic graph algorithm that requires $\tilde{O}(n)$ space (in other words, a dynamic semi-streaming algorithm) and at the same time can quickly process each update and answer each query. Previously, there was no space-efficient algorithm known for this problem, even when time efficiency is not a concern, and even in the conventional streaming model where there are only edge insertions. In this insertion-only model, Bahmani, Kumar, and Vassilvitskii [6] provided a deterministic $(2+\epsilon)$-approximation $O(n)$-space

[2] We note that we can also quickly return the subgraph whose density γ-approximates $\rho^*(G)$.

algorithm. Their algorithm needs $O(\log_{1+\epsilon} n)$ passes; i.e., it has to read through the sequence of edge insertions $O(\log_{1+\epsilon} n)$ times. (Their algorithm was also extended to a MapReduce algorithm, which was later improved by [5].) Our $(2+\epsilon)$-approximation dynamic streaming algorithm improves this algorithm in terms of the number of passes. The space usage of our dynamic algorithms matches the lower bound provided by [6, Lemma 7] up to a polylogarithmic factor.

We note that while in some settings it is reasonable to compute the solution at the end of the stream or even make multiple passes (e.g. when the graph is kept on an external memory), and thus our and Bahmani et al's $(2+\epsilon)$-approximation algorithms are sufficient in these settings, there are many natural settings where the stream keeps changing, e.g. social networks where users keep making new friends and disconnecting from old friends. In the latter case our main algorithm is necessary since it can quickly prepare to answer the densest subgraph query after every update.

Another related result in the streaming setting is by Ahn et al. [2] which approximates the fraction of some dense subgraphs such as a small clique in dynamic streams. This algorithm does not solve the densest subgraph problem but might be useful for similar applications.

Not much was known about time-efficient algorithm for this problem even when space efficiency is not a concern. One possibility is to adapt dynamic algorithms for the related problem called *dynamic arboricity*. The arboricity of a graph G is $\alpha(G) = \max_{U \subseteq V(G)} |E(U)|/(|U|-1)$ where $E(U)$ is the subgraph of G induced by U. Observe that $\rho^*(G) \leq \alpha(G) \leq 2\rho^*(G)$. Thus, a γ-approximation for the arboricity problem will be a (2γ)-approximation algorithm. In particular, we can use the 4-approximation algorithm of Brodal and Fagerberg [7] to maintain an 8-approximate solution to the densest subgraph problem in $\tilde{O}(1)$ amortized update time. (With a little more thought, one can in fact improve the approximation ratio to 6.) In the paper that appeared at about the same time as this paper, Epasto et al. [**?**] presented a $(2+\epsilon)$-approximation algorithm which can handle arbitrary edge insertions and random edge deletions.

Overview. An intuitive way to combine techniques from data streams and dynamic algorithms for any problem is to run the dynamic algorithm using the sketch produced by the streaming algorithm as an input. This idea does not work straightforwardly. The first obvious issue is that the streaming algorithm might take excessively long time to maintain its sketch and the dynamic algorithm might require an excessively large additional space. A more subtle issue is that the sketch might need to be processed in a specific way to recover a solution, and the dynamic algorithm might not be able to facilitate this. As an extreme example, imagine that the sketch for our problem is not even a graph; in this case, we cannot even feed this sketch to a dynamic algorithm as an input.

The key idea that allows us to get around this difficulty is to develop streaming and dynamic algorithms based on the same structure called *(α, d, L)-decomposition*. This structure is an extension of a concept called *d-core*, which was studied in graph theory since at least the 60s (e.g., [14, 27, 34]) and has played an important role in the studies of the densest subgraph problem (e.g., [6, 33]). The d-core of a graph is its (unique) largest induced subgraph with every node having degree at least d. It can be computed by repeatedly removing nodes of degree less than d from the graph, and can be used to 2-approximate the densest subgraph. Our (α, d, L)-decomposition with parameter $\alpha \geq 1$ is an approximate version of this process where we repeatedly remove nodes of degree "approximately" less than d: in this decomposition we must remove all nodes of degree less than d and are allowed to remove *some* nodes of degree between d and αd. We will repeat this process for L iterations. Note that the (α, d, L)-decomposition of a graph is not unique. However, for $L = O(\log_{1+\epsilon} n)$, an (α, d, L)-decomposition can be use to $2\alpha(1+\epsilon)^2$-approximate the densest subgraph. We explain this concept in detail in Section 2.

We show that this concept can be used to obtain an approximate solution to the densest subgraph problem and leads to both a streaming algorithm with a small sketch and a dynamic

algorithm with small amortized update time. In particular, it is intuitive that to check if a node has degree approximately d, it suffices to sample every edge with probability roughly $1/d$. The value of d that we are interested in approximately ρ^*, which can be shown to be roughly the same as the average degree of the graph. Using this fact, it follows almost immediately that we only have to sample $\tilde{O}(n)$ edges. Thus, to repeatedly remove nodes for L iterations, we will need to sample $\tilde{O}(Ln) = \tilde{O}(n)$ edges (we need to sample a new set of edges in every iteration to avoid dependencies).

We turn the (α, d, L)-decomposition concept into a dynamic algorithm by dynamically maintaining the sets of nodes removed in each of the L iterations, called *levels*. Since the (α, d, L)-decomposition gives us a choice whether to keep or remove each node of degree between d and αd, we can save time needed to maintain this decomposition by moving nodes between levels *only when it is necessary*. If we allow α to be large enough, nodes will not be moved often and we can obtain a small amortized update time; in particular, it can be shown that the amortized update time is $\tilde{O}(1)$ if $\alpha \geq 2 + \epsilon$. In analyzing an amortized time, it is usually tricky to come up with the right *potential function* that can keep track of the cost of moving nodes between levels, which is not frequent but expensive. In case of our algorithm, we have to define two potential functions for our amortized analysis, one on nodes and one on edges. (For intuition, we provide an analysis for the simpler case where we run this dynamic algorithm directly on the input graph in Section 7.)

Our goal is to run the dynamic algorithm on top of the sketch maintained by our streaming algorithm in order to maintain the (α, d, L)-decomposition. To do this, there are a few issues we have to deal with that makes the analysis rather complicated: Recall that in the sketch we maintain L sets of sampled edges, and for each of the L iterations we use different such sets to determine which nodes to remove. This causes the potential functions and its analysis to be even more complicated since whether a node should be moved from one level to another depends on its degree in one set, but the cost of moving such node depends on its degree in other sets as well. The analysis fortunately goes through (intuitively because all sets are sampled from the same graph and so their degree distributions are close enough). We explain our algorithm and how to analysis it in details in Section 4.

Notation. For any graph $G = (V, E)$, let $\mathcal{N}_v = \{u \in V : (u, v) \in E\}$ and $D_v = |\mathcal{N}_v|$ respectively denote the set of neighbors and the degree of a node $v \in V$. Let $G(S)$ denote the subgraph of G induced by the nodes in $S \subseteq V$. Given any two subsets $S \subseteq V, E' \subseteq E$, define $\mathcal{N}_u(S, E') = \{v \in \mathcal{N}_u \cap S : (u, v) \in E'\}$ and $D_u(S, E') = |\mathcal{N}_u(S, E')|$. To ease notation, we write $\mathcal{N}_u(S)$ and $D_u(S)$ instead of $\mathcal{N}_u(S, E)$ and $D_u(S, E)$. For a nonempty subset $S \subseteq V$, its density and average degree are defined as $\rho(S) = |E(S)|/|S|$ and $\delta(S) = \sum_{v \in S} D_v(S)/|S|$ respectively. Note that $\delta(S) = 2 \cdot \rho(S)$.

2 (α, d, L)-decomposition

Our (α, d, L)-decomposition is formally defined as follows.

Definition 2.1. *Fix any $\alpha \geq 1$, $d \geq 0$, and any positive integer L. Consider a family of subsets $Z_1 \supseteq \cdots \supseteq Z_L$. The tuple $(Z_1, \ldots, Z_L)$ is an (α, d, L)-decomposition of the input graph $G = (V, E)$ iff $Z_1 = V$ and, for every $i \in [L-1]$, we have $Z_{i+1} \supseteq \{v \in Z_i : D_v(Z_i) > \alpha d\}$ and $Z_{i+1} \cap \{v \in Z_i : D_v(Z_i) < d\} = \emptyset$.*

Given an (α, d, L)-decomposition $(Z_1, \ldots, Z_L)$, we define $V_i = Z_i \setminus Z_{i+1}$ for all $i \in [L-1]$, and $V_i = Z_i$ for $i = L$. We say that the nodes in V_i constitute the i^{th} level of this decomposition. We also denote the level of a node $v \in V$ by $\ell(v)$. Thus, we have $\ell(v) = i$ whenever $v \in V_i$. The following theorem and its immediate corollary will play the main role in the rest of the paper.

Roughly speaking, they state that we can use the (α, d, L)-decomposition to $2\alpha(1+\epsilon)^2$-approximate the densest subgraph by setting $L = O(\log n/\epsilon)$ and trying different values of d in powers of $(1+\epsilon)$.

Theorem 2.2. *Fix any $\alpha \geq 1$, $d \geq 0$, $\epsilon \in (0,1)$, $L \leftarrow 2+\lceil \log_{(1+\epsilon)} n \rceil$. Let $d^* \leftarrow \max_{S \subseteq V} \rho(S)$ be the maximum density of any subgraph in $G = (V, E)$, and let $(Z_1, \ldots, Z_L)$ be an (α, d, L)-decomposition of $G = (V, E)$. We have*

- *(1) If $d > 2(1+\epsilon)d^*$, then $Z_L = \emptyset$.*
- *(2) Else if $d < d^*/\alpha$, then $Z_L \neq \emptyset$ and there is an index $j \in \{1, \ldots, L-1\}$ such that $\rho(Z_j) \geq d/(2(1+\epsilon))$.*

Corollary 2.3. *Fix α, ϵ, L, d^* as in Theorem 2.2. Let $\pi, \sigma > 0$ be any two numbers satisfying $\alpha \cdot \pi < d^* < \sigma/(2(1+\epsilon))$. Discretize the range $[\pi, \sigma]$ into powers of $(1+\epsilon)$, by defining $d_k \leftarrow (1+\epsilon)^{k-1} \cdot \pi$ for every $k \in [K]$, where K is any integer strictly greater than $\lceil \log_{(1+\epsilon)}(\sigma/\pi) \rceil$. For every $k \in [K]$, construct an (α, d_k, L)-decomposition $(Z_1(k), \ldots, Z_L(k))$ of $G = (V, E)$. Let $k' \leftarrow \max\{k \in [K] : Z_L(k) \neq \emptyset\}$. Then we have the following guarantees:*

- *$d^*/(\alpha(1+\epsilon)) \leq d_{k'} \leq 2(1+\epsilon) \cdot d^*$.*
- *There exists an index $j' \in \{1, \ldots, L-1\}$ such that $\rho(Z_{j'}) \geq d_{k'}/(2(1+\epsilon))$.*

We will use the above corollary as follows. Since $K = O(\log_{1+\epsilon} n)$, it is not hard to maintain k' and the set of nodes $Z_{j'}(k')$. The corollary guarantees that the density of the set of nodes $Z_{j'}(k')$ is $(2\alpha(1+\epsilon)^2)$-approximation to d^*.

The rest of this section is devoted to proving Theorem 2.2.

The first lemma relates the density to the minimum degree. Its proof can be found in the full version.

Lemma 2.4. *Let $S^* \subseteq V$ be a subset of nodes with maximum density, i.e., $\rho(S^*) \geq \rho(S)$ for all $S \subseteq V$. Then $D_v(S^*) \geq \rho(S^*)$ for all $v \in S^*$. Thus, the degree of each node in $G(S^*)$ is at least the density of S^*.*

of Theorem 2.2. (1) Suppose that $d > 2(1+\epsilon)d^*$. Consider any level $i \in [L-1]$, and note that $\delta(Z_i) = 2 \cdot \rho(Z_i) \leq 2 \cdot \max_{S \subseteq V} \rho(S) = 2d^* < d/(1+\epsilon)$. It follows that the number of nodes v in $G(Z_i)$ with degree $D_v(Z_i) \geq d$ is less than $|Z_i|/(1+\epsilon)$, as otherwise $\delta(Z_i) \geq d/(1+\epsilon)$. Let us define the set $C_i = \{v \in Z_i : D_v(Z_i) < d\}$. We have $|Z_i \setminus C_i| \leq |Z_i|/(1+\epsilon)$. Now, from Definition 2.1 we have $Z_{i+1} \cap C_i = \emptyset$, which, in turn, implies that $|Z_{i+1}| \leq |Z_i \setminus C_i| \leq |Z_i|/(1+\epsilon)$. Thus, for all $i \in [L-1]$, we have $|Z_{i+1}| \leq |Z_i|/(1+\epsilon)$. Multiplying all these inequalities, for $i = 1$ to $L-1$, we conclude that $|Z_L| \leq |Z_1|/(1+\epsilon)^{L-1}$. Since $|Z_1| = |V| = n$ and $L = 2 + \lceil \log_{(1+\epsilon)} n \rceil$, we get $|Z_L| \leq n/(1+\epsilon)^{(1+\log_{(1+\epsilon)} n)} < 1$. This can happen only if $Z_L = \emptyset$.

(2) Suppose that $d < d^*/\alpha$, and let $S^* \subseteq V$ be a subset of nodes with highest density, i.e., $\rho(S^*) = d^*$. We will show that $S^* \subseteq Z_i$ for all $i \in \{1, \ldots, L\}$. This will imply that $Z_L \neq \emptyset$. Clearly, we have $S^* \subseteq V = Z_1$. By induction hypothesis, assume that $S^* \subseteq Z_i$ for some $i \in [L-1]$. We show that $S^* \subseteq Z_{i+1}$. By Lemma 2.4, for every node $v \in S^*$, we have $D_v(Z_i) \geq D_v(S^*) \geq \rho(S^*) = d^* > \alpha d$. Hence, from Definition 2.1, we get $v \in Z_{i+1}$ for all $v \in S^*$. This implies that $S^* \subseteq Z_{i+1}$.

Next, we will show that if $d < d^*/\alpha$, then there is an index $j \in \{1, \ldots, L-1\}$ such that $\rho(Z_j) \geq d/(2(1+\epsilon))$. For the sake of contradiction, suppose that this is not the case. Then we have $d < d^*/\alpha$ and $\delta(Z_i) = 2 \cdot \rho(Z_i) < d/(1+\epsilon)$ for every $i \in \{1, \ldots, L-1\}$. Then, applying an argument similar to case (1), we conclude that $|Z_{i+1}| \leq |Z_i|/(1+\epsilon)$ for every $i \in \{1, \ldots, L-1\}$, which implies that $Z_L = \emptyset$. Thus, we arrive at a contradiction. □

3 Warmup: A Single Pass Streaming Algorithm

In this section, we present a single-pass streaming algorithm for maintaining a $(2+\epsilon)$-approximate solution to the densest subgraph problem. The algorithm handles a dynamic (turnstile) stream of edge insertions/deletions in $\tilde{O}(n)$ space. In particular, we do not worry about the update time of our algorithm. Our main result in this section is summarized in Theorem 3.1.

Theorem 3.1. *We can process a dynamic stream of updates in the graph G in $\tilde{O}(n)$ space, and with high probability return a $(2+O(\epsilon))$-approximation of $d^* = \max_{S \subseteq V} \rho(S)$ at the end of the stream.*

Throughout this section, we fix a small constant $\epsilon \in (0, 1/2)$ and a sufficiently large constant $c > 1$. Moreover, we set $\alpha \leftarrow (1+\epsilon)/(1-\epsilon)$, $L \leftarrow 2 + \lceil \log_{(1+\epsilon)} n \rceil$. The main technical lemma is below and states that we can construct a (α, d, L)-decomposition by sampling $\tilde{O}(n)$ edges.

Lemma 3.2. *Fix an integer $d > 0$, and let S be a collection of $cm(L-1)\log n/d$ mutually independent random samples (each consisting of one edge) from the edge-set E of the input graph $G = (V, E)$. With high probability we can construct from S an (α, d, L)-decomposition $(Z_1, \ldots, Z_L)$ of G, using only $\tilde{O}((n + m/d))$ bits of space.*

Proof. We partition the samples in S evenly among $(L-1)$ groups $\{S_i\}, i \in [L-1]$. Thus, each S_i is a collection of $cm \log n/d$ mutually independent random samples from the edge-set E, and, furthermore, the collections $\{S_i\}, i \in [L-1]$, themselves are mutually independent. Our algorithm works as follows.

- Set $Z_1 \leftarrow V$.
- For $i = 1$ to $(L-1)$: Set $Z_{i+1} \leftarrow \{v \in Z_i : D_v(Z_i, S_i) \geq (1-\epsilon)\alpha c \log n\}$.

To analyze the correctness of the algorithm, define the (random) sets $A_i = \{v \in Z_i : D_v(Z_i, E) > \alpha d\}$ and $B_i = \{v \in Z_i : D_v(Z_i, E) < d\}$ for all $i \in [L-1]$. Note that for all $i \in [L-1]$, the random sets Z_i, A_i, B_i are completely determined by the outcomes of the samples in $\{S_j\}, j < i$. In particular, the samples in S_i are chosen independently of the sets Z_i, A_i, B_i. Let $\mathcal{E}_i$ be the event that (a) $Z_{i+1} \supseteq A_i$ and (b) $Z_{i+1} \cap B_i = \emptyset$. By Definition 2.1, the output $(Z_1, \ldots, Z_L)$ is a valid (α, d, L)-decomposition of G iff the event $\bigcap_{i=1}^{L-1} \mathcal{E}_i$ occurs. Consider any $i \in [L-1]$. Below, we show that the event $\mathcal{E}_i$ occurs with high probability. The lemma follows by taking a union bound over all $i \in [L-1]$.

Fix any instantiation of the random set Z_i. Condition on this event, and note that this event completely determines the sets A_i, B_i. Consider any node $v \in A_i$. Let $X_{v,i}(j) \in \{0, 1\}$ be an indicator random variable for the event that the j^{th} sample in S_i is of the form (u, v), with $u \in \mathcal{N}_v(Z_i)$. Note that the random variables $\{X_{v,i}(j)\}, j$, are mutually independent. Furthermore, we have $E[X_{v,i}(j)|Z_i] = D_v(Z_i)/m > \alpha d/m$ for all j. Since there are $cm \log n/d$ such samples in S_i, by linearity of expectation we get: $E[D_v(Z_i, S_i)|Z_i] = \sum_j E[X_{v,i}(j)|Z_i] > (cm \log n/d) \cdot (\alpha d/m) = \alpha c \log n$. The node v is included in Z_{i+1} iff $D_v(Z_i, S_i) \geq (1-\epsilon)\alpha c \log n$, and this event, in turn, occurs with high probability (by Chernoff bound). Taking a union bound over all nodes $v \in A_i$, we conclude that $\Pr[Z_{i+1} \supseteq A_i \,|\, Z_i] \geq 1 - 1/(\text{poly } n)$. Using a similar line of reasoning, we get that $\Pr[Z_{i+1} \cap B_i = \emptyset \,|\, Z_i] \geq 1 - 1/(\text{poly } n)$. Invoking a union bound over these two events, we get $\Pr[\mathcal{E}_i \,|\, Z_i] \geq 1 - 1/(\text{poly } n)$. Since this holds for all possible instantiations of Z_i, the event $\mathcal{E}_i$ itself occurs with high probability.

The space requirement of the algorithm, ignoring poly log factors, is proportional to the number of samples in S (which is $cm(L-1)\log n/d$) plus the number of nodes in V (which is n). Since

c is a constant and since $L = O(\text{poly}\log n)$, we derive that the total space requirement is $O((n + m/d)\ \text{poly}\log n)$. □

Now, to turn Lemma 3.2 into a streaming algorithm, we simply have to invoke Lemma 3.3 which follows from a well-known result about ℓ_0-sampling in the streaming model [21], and a simple observation (yet very important) in Lemma 3.4.

Lemma 3.3 (ℓ_0-sampler [21])**.** *We can process a dynamic stream of $O(\text{poly}\ n)$ updates in the graph $G = (V, E)$ in $O(\text{poly}\log n)$ space, and with high probability, at each step we can maintain a simple random sample from the set E. The algorithm takes $O(\text{poly}\log n)$ time to handle each update in the stream.*

Lemma 3.4. *Let $d^* = \max_{S \subseteq V} \rho(S)$ be the maximum density of any subgraph in G. Then $m/n \leq d^* < n$.*

of Theorem 3.1. Using binary search, we guess the number of edges m in the graph $G = (V, E)$ at the end of the stream. Define $\pi \leftarrow m/(2\alpha n)$ and $\sigma \leftarrow 2(1+\epsilon)n$. Since $\epsilon \in (0, 1/2)$, by Lemma 3.4 we have $\alpha \cdot \pi < d^* < \sigma/(2(1+\epsilon))$. Thus, we can discretize the range $[\pi, \sigma]$ in powers of $(1+\epsilon)$ by defining the values $\{d_k\}, k \in [K]$, as per Corollary 2.3. It follows that to return a $2\alpha(1+\epsilon)^2 = (2 + O(\epsilon))$-approximation of optimal density, all we need to do is to construct an (α, d_k, L)-decomposition of the graph $G = (V, E)$ at the end of the stream, for every $k \in [K]$. Since $K = O(\log_{(1+\epsilon)}(\sigma/\pi)) = O(\text{poly}\log n)$, Theorem 3.1 follows from Claim 3.5.

Claim 3.5. *Fix any $k \in [K]$. We can process a dynamic stream of updates in the graph G in $O(n\ \text{poly}\log n)$ space, and with high probability return an (α, d_k, L)-decomposition of G at the end of the stream.*

Now we prove Claim 3.5. Define $\lambda_k \leftarrow cm(L-1)\log n/d_k$. Since $d_k \geq \pi = m/(2\alpha n)$, we have $\lambda_k = O(n\ \text{poly}\log n)$. While going through the dynamic stream of updates in G, we simultaneously run λ_k mutually independent copies of the ℓ_0-sampler as specified in Lemma 3.3. Thus, with high probability, we get λ_k mutually independent simple random samples from the edge-set E at the end of the stream. Next, we use these random samples to construct an (α, d_k, L)-decomposition of G, with high probability, as per Lemma 3.2.

By Lemma 3.3, each ℓ_0-sampler requires $O(\text{poly}\log n)$ bits of space, and there are λ_k many of them. Furthermore, the algorithm in Lemma 3.2 requires $O((n + m/d_k)\ \text{poly}\log n)$ bits of space. Thus, the total space requirement of our algorithm is $O((\lambda_k + n + m/d_k)\ \text{poly}\log n) = O(n\ \text{poly}\log n)$ bits. □

4 A Single Pass Dynamic Streaming Algorithm

We devote this section to the proof of our main result (Theorem 1.1). Throughout this section, fix $\alpha = 2 + \Theta(\epsilon)$, $L \leftarrow 2 + \lceil \log_{(1+\epsilon)} n \rceil$, and let $c \gg \lambda$ be a sufficiently large constant. We call the input graph "sparse" whenever it has less than $4\alpha c^2 n \log^2 n$ edges, and "dense" otherwise. We simultaneously run two algorithms while processing the stream of updates – the first (resp. second) one outputs a correct value whenever the graph is sparse (resp. dense). It is the algorithm for dense graphs that captures the technical difficulty of the problem. To focus on this case (due to space constraints), we assume that the first $4\alpha c^2 n \log^2 n$ updates in the dynamic stream consist of only edge-insertions, so that the graph $G^{(t)}$ becomes dense at $t = 4\alpha c^2 n \log^2 n$. Next, we assume that the graph $G^{(t)}$ remains dense at each $t \geq 4\alpha c^2 n \log^2 n$. We focus on maintaining the value of $\textsc{Output}^{(t)}$ during the latter phase. For a full proof of Theorem 1.1 that does not require any of these simplifying assumptions, see Section 8.

Assumption 4.1. *Define* $T' \leftarrow \lceil 4\alpha c^2 n \log^2 n \rceil$. *We have* $m^{(t)} \geq 4\alpha c^2 n \log^2 n$ *for all* $t \in [T', T]$.

Consider any $t \in [T', T]$. Define $\pi^{(t)} = m^{(t)}/(2\alpha n)$ and $\sigma = 2(1+\epsilon)n$. It follows that $\alpha \cdot \pi^{(t)} < \textsc{Opt}^{(t)} < \sigma/(2(1+\epsilon))$. Discretize the range $[\pi^{(t)}, \sigma]$ in powers of $(1+\epsilon)$, by defining $d_k^{(t)} \leftarrow (1+\epsilon)^{k-1} \cdot \pi^{(t)}$ for all $k \in [K]$, where $K \leftarrow 1 + \lceil \log_{(1+\epsilon)}(\sigma \cdot (2\alpha n)) \rceil$. Note that for all $t \in [T', T]$ we have $K > \lceil \log_{(1+\epsilon)}(\sigma/\pi^{(t)}) \rceil$. Also note that $K = O(\text{poly} \log n)$. By Corollary 2.3, the algorithm only has to maintain an $(\alpha, d_k^{(t)}, L)$-decomposition for each $k \in [K]$. Specifically, Theorem 1.1 follows from Theorem 4.2.

Theorem 4.2. *Let us fix any* $k \in [K]$. *There is an algorithm that processes the first* T *updates in the dynamic stream using* $\tilde{O}(n)$ *space, and under Assumption 4.1, it gives the following guarantees with high probability: At each* $t \in [T', T]$, *the algorithm maintains an* $(\alpha, d_k^{(t)}, L)$*-decomposition* $(Z_1^{(t)}, \ldots, Z_L^{(t)})$ *of* $G^{(t)}$. *Further, the total amount of computation performed is* $O(T \ \text{poly} \log n)$.

As we mentioned earlier in Section 1, our algorithm can output an approximate densest subgraph by maintaining the density at each level of the (α, d, L) decomposition and simply keeping track of the level that gives us maximum density. We devote the rest of this Section to the proof of Theorem 4.2.

Proof of Theorem 4.2 **Notation.** Define $s_k^{(t)} = cm^{(t)} \log n / d_k^{(t)}$ for all $t \in [T', T]$. Plugging in the value of $d_k^{(t)}$, we get $s_k^{(t)} = 2\alpha cn \log n/(1+\epsilon)^{k-1}$. Since $s_k^{(t)}$ does not depend on t, we omit the superscript and refer to it as s_k instead.

Overview of our approach. As a first step, we want to show that for each $i \in [L-1]$, we can maintain a random set of s_k edges $S_i^{(t)} \subseteq E^{(t)}$ such that $\Pr[e \in S_i^{(t)}] = s_k/m^{(t)}$ for all $e \in E^{(t)}$. This has the following implication: Fix any subset of nodes $U \subseteq V$. If a node $u \in U$ has $D_u(U, E^{(t)}) > \alpha d_k^{(t)}$, then in expectation we have $D_u(U, S_i^{(t)}) > \alpha c \log n$. Since this expectation is large enough, a suitable Chernoff bound implies that $D_u(U, S_i^{(t)}) > (1-\epsilon)\alpha c \log n$ with high probability. Accordingly, we can use the random sets $\{S_i^{(t)}\}$, $i \in [L-1]$, to construct an $(\alpha, d_k^{(t)}, L)$-decomposition of $G^{(t)}$ as follows. We set $Z_1^{(t)} = V$, and for each $i \in [L-1]$, we iteratively construct the subset $Z_{i+1}^{(t)}$ by taking the nodes $u \in Z_i^{(t)}$ with $D_u(Z_i^{(t)}, S_i^{(t)}) > (1-\epsilon)\alpha c \log n$. Here, we crucially need the property that the random set $S_i^{(t)}$ is chosen independently of the contents of $Z_i^{(t)}$. Note that $Z_i^{(t)}$ is actually determined by the contents of the sets $\{S_j^{(t)}\}, j < i$. Since $s_k = \tilde{O}(n)$, each of these random sets $S_i^{(t)}$ consists of $\tilde{O}(n)$ many edges. While following up on this high level approach, we need to address two major issues, as described below.

Fix some $i \in [L-1]$. A naive way of maintaining the set $S_i^{(t)}$ would be to invoke a well known result on ℓ_0-sampling on dynamic streams (see Lemma 3.3). This allows us to maintain a uniformly random sample from $E^{(t)}$ in $\tilde{O}(1)$ update time. So we might be tempted to run s_k mutually independent copies of such an ℓ_0-Sampler on the edge-set $E^{(t)}$ to generate a random set of size s_k. The problem is that when an edge insertion/deletion occurs in the input graph, we have to probe each of these ℓ_0-Samplers, leading to an overall update time of $O(s_k \ \text{poly} \log n)$, which can be as large as $\tilde{\Theta}(n)$ when k is small (say for $k = 1$). In Lemma 4.3, we address this issue by showing how to maintain the set $S_i^{(t)}$ in $\tilde{O}(1)$ worst case update time and $\tilde{O}(n)$ space.

The remaining challenge is to maintain the decomposition $(Z_1^{(t)}, \ldots, Z_L^{(t)})$ dynamically as the random sets $\{S_i^{(t)}\}, i \in [L-1]$, change with t. Again, a naive implementation – building the decomposition from scratch at each t – would require $\Theta(n)$ update time. In Section 4.1, we give a procedure that builds a new decomposition at any given $t \in [T', T]$, based on the old decomposition

at $(t-1)$ and the new random sets $\{S_i^{(t)}\}, i \in [L-1]$. In Section 4.2, we present the data structures for implementing this procedure and analyze the space complexity. In Section 4.3, we bound the amortized update time using an extremely fine-tuned potential function. Theorem 4.2 follows from Lemmata 4.5, 4.6, 4.10 and Claim 4.9.

Lemma 4.3. *We can process the first T updates in a dynamic stream using $\tilde{O}(n)$ space and maintain a random subset of edges $S_i^{(t)} \subseteq E^{(t)}, |S_i^{(t)}| = s_k$, at each $t \in [T', T]$. Let $X_{e,i}^{(t)}$ denote an indicator variable for the event $e \in S_i^{(t)}$. The following guarantee holds w.h.p.:*

- *At each $t \in [T', T]$, we have that $\Pr[X_{e,i}^{(t)} = 1] \in \left[(1 \pm \epsilon) c \log n / d_k^{(t)}\right]$ for all $e \in E^{(t)}$. The variables $\left\{X_{e,i}^{(t)}\right\}, e \in E^{(t)}$, are negatively associated.*
- *Each update in the dynamic stream is handled in $\tilde{O}(1)$ time and leads to at most two changes in S_i.*

Proof. (Sketch) Let E^* denote the set of all possible ordered pairs of nodes in V. Thus, $E^* \supseteq E^{(t)}$ at each $t \in [1, T]$, and furthermore, we have $|E^*| = O(n^2)$. Using a well known result from the hashing literature [30], we construct a $(2cs_k \log n)$-wise independent uniform hash function $h : E^* \to [s_k]$ in $\tilde{O}(n)$ space. This hash function partitions the edge-set $E^{(t)}$ into s_k mutually disjoint buckets $\{Q_j^{(t)}\}, j \in [s_k]$, where the bucket $Q_j^{(t)}$ consists of those edges $e \in E^{(t)}$ with $h(e) = j$. For each $j \in [s_k]$, we run an independent copy of ℓ_0-Sampler, as per Lemma 3.3, that maintains a uniformly random sample from $Q_j^{(t)}$. The set $S_i^{(t)}$ consists of the collection of outputs of all these ℓ_0-Samplers. Note that (a) for each $e \in E^*$, the hash value $h(e)$ can be evaluated in constant time [30], (b) an edge insertion/deletion affects exactly one of the buckets, and (c) the ℓ_0-Sampler of the affected bucket can be updated in $\tilde{O}(1)$ time. Thus, we infer that this procedure handles an edge insertion/deletion in the input graph in $\tilde{O}(1)$ time, and furthermore, since $s_k = \tilde{O}(n)$, the procedure can be implemented in $\tilde{O}(n)$ space.

Fix any time-step $t \in [T', T]$ (see Assumption 4.1). Since $m^{(t)} \geq 2cs_k \log n$, we can partition (purely as a thought experiment) the edges in $E^{(t)}$ into at most polynomially many groups $\left\{H_{j'}^{(t)}\right\}$, in such a way that the size of each group lies between $cs_k \log n$ and $2cs_k \log n$. Thus, for any $j \in [s_k]$ and any j', we have $|H_{j'}^{(t)} \cap Q_j^{(t)}| \in [c \log n, 2c \log n]$ in expectation. Since the hash function h is $(2cs_k \log n)$-wise independent, by applying a Chernoff bound we infer that with high probability, the value $|H_{j'}^{(t)} \cap Q_j^{(t)}|$ is very close to its expectation. Applying the union bound over all j, j', we infer that with high probability, the sizes of all the sets $\left\{H_{j'}^{(t)} \cap Q_j^{(t)}\right\}$ are very close to their expected values – let us call this event $\mathcal{R}^{(t)}$. Since $E[|Q_j^{(t)}|] = m^{(t)}/s_k$ and $|Q_j^{(t)}| = \sum_{j'} |Q_j^{(t)} \cap H_{j'}^{(t)}|$, under the event $\mathcal{R}^{(t)}$, we have that $|Q_j^{(t)}|$ is very close to $m^{(t)}/s_k$ for all $j \in [s_k]$. Under the same event $\mathcal{R}^{(t)}$, due to the ℓ_0-Samplers, the probability that a given edge $e \in E^{(t)}$ becomes part of $S_i^{(t)}$ is very close to $1/|Q_j^{(t)}| \approx s_k/m^{(t)} = c \log n / d_k^{(t)}$.

Finally, the property of negative association follows from the observations that (a) if two edges are hashed to different buckets, then they are included in $S_i^{(t)}$ in a mutually independent manner, and (b) if they are hashed to the same bucket, then they are never simultaneously included in $S_i^{(t)}$. □

4.1 Maintaining an $(\alpha, d_k^{(t)}, L)$-decomposition using the random sets $S_i^{(t)}, i \in [L-1]$

While processing the stream of updates, we run an independent copy of the algorithm in Lemma 4.3 for each $i \in [L-1]$. Thus, we assume that we have access to the random sets $S_i^{(t)}, i \in [L-1]$, at each $t \in [T', T]$. In this section, we present an algorithm that maintains a decomposition $(Z_1^{(t)}, \ldots, Z_L^{(t)})$ at each time-step $t \in [T', T]$ as long as the graph is dense (see Assumption 4.1), using the random sets $S_i^{(t)}, i \in [L-1]$. Specifically, we handle the t^{th} update in the dynamic stream as per the procedure in Figure 1. The procedure outputs the new decomposition $(Z_1^{(t)}, \ldots, Z_L^{(t)})$ starting from the old decomposition $(Z_1^{(t-1)}, \ldots, Z_L^{(t-1)})$ and the new samples $\left\{S_i^{(t)}\right\}, i \in [L-1]$.

```
01.  Set Z_1^(t) ← V.
02.  FOR i = 1 to L
03.        Set Y_i ← Z_i^(t−1).
04.  FOR i = 1 to (L − 1)
05.        Let A_i^(t) be the set of nodes y ∈ Z_i^(t) having D_y(Z_i^(t), S_i^(t)) > (1 − ε)^2 αc log n.
06.        Let B_i^(t) be the set of nodes y ∈ Z_i^(t) having D_y(Z_i^(t), S_i^(t)) < (1 + ε)^2 c log n.
07.        Set Y_{i+1} ← Y_{i+1} ∪ A_i^(t).
08.        FOR ALL j = (i + 1) to (L − 1)
09.              Set Y_j ← Y_j \ B_i^(t).
10.        Set Z_{i+1}^(t) ← Y_{i+1}.
```

Figure 1: RECOVER-SAMPLE(t).

We have the following observation.

Lemma 4.4. *Fix a $t \in [T', T]$ and an $i \in [L-1]$. (1) The set $Z_i^{(t)}$ is completely determined by the contents of the sets $\left\{S_j^{(t)}\right\}, j < i$. (2) The sets $\left\{S_j^{(t)}\right\}, j \geq i$, are chosen independently of the contents of the set $Z_i^{(t)}$.*

Lemma 4.5. *With high probability, at each $t \in [T', T]$ the tuple $(Z_1^{(t)} \ldots Z_L^{(t)})$ is an $(\alpha, d_k^{(t)}, L)$-decomposition of $G^{(t)}$.*

Proof. (sketch) For $t \in [T', T], i \in [L-1]$, let $\mathcal{E}_i^{(t)}$ denote the event that (a) $Z_{i+1}^{(t)} \supseteq \{v \in Z_i^{(t)} : D_v(Z_i^{(t)}, E^{(t)}) > \alpha d_k^{(t)}\}$ and (b) $Z_{i+1}^{(t)} \cap \{v \in Z_i^{(t)} : D_v(Z_i^{(t)}, E^{(t)}) < d_k^{(t)}\} = \emptyset$. By Definition 2.1, the tuple $(Z_1^{(t)} \ldots Z_L^{(t)})$ is an $(\alpha, d_k^{(t)}, L)$-decomposition of $G^{(t)}$ iff the event $\mathcal{E}_i^{(t)}$ holds for all $i \in [L-1]$. Below, we show that $\Pr[\mathcal{E}_i^{(t)}] \geq 1 - 1/(\text{poly } n)$ for any given $i \in [L-1]$ and $t \in [T', T]$. The lemma follows by taking a union bound over all i, t.

Fix any instance of the random set $Z_i^{(t)}$ and condition on this event. Consider any node $v \in Z_i^{(t)}$ with $D_v(Z_i^{(t)}, E^{(t)}) > \alpha d_k^{(t)}$. By Lemma 4.3, each edge $e \in E^{(t)}$ appears in $S_i^{(t)}$ with probability $(1 \pm \epsilon) c \log n / d_k^{(t)}$ and these events are negatively associated. By linearity of expectation, we have $E[D_v(Z_i^{(t)}, S_i^{(t)})] \geq (1-\epsilon)\alpha c \log n$. Since the random set $S_i^{(t)}$ is chosen independently of the contents of $Z_i^{(t)}$ (see Lemma 4.4), we can apply a Chernoff bound on this expectation and derive that $\Pr[v \notin Z_{i+1}^{(t)} \mid Z_i^{(t)}] = \Pr[D_v(Z_i^{(t)}, S_i^{(t)}) \leq (1-\epsilon)^2 \alpha c \log n \mid Z_i^{(t)}] \leq 1/(\text{poly } n)$. Next, consider any node $u \in Z_i^{(t)}$ with $D_u(Z_i^{(t)}, E^{(t)}) < d_k^{(t)}$. Using a similar argument, we get $\Pr[u \in Z_{i+1}^{(t)} \mid Z_i^{(t)}] = \Pr[D_u(Z_i^{(t)}, E^{(t)}) \geq (1+\epsilon)^2 c \log n \mid Z_i^{(t)}] \leq 1/(\text{poly } n)$. Taking a union bound over all possible nodes, we infer that $\Pr[\mathcal{E}_i^{(t)} \mid Z_i^{(t)}] \geq 1 - 1/(\text{poly } n)$.

Since the guarantee $\Pr[\mathcal{E}_i^{(t)} \mid Z_i^{(t)}] \geq 1 - 1/(\text{poly } n)$ holds for every possible instance of $Z_i^{(t)}$, we get $\Pr[\mathcal{E}_i^{(t)}] \geq 1 - 1/(\text{poly } n)$. □

4.2 Data structures for the procedure in Figure 1

Recall the notations introduced immediately after Definition 2.1.

- Consider any node $v \in V$ and any $i \in \{1, \ldots, L-1\}$. We maintain the doubly linked lists $\{\textsc{Friends}_i[v, j]\}, 1 \leq j \leq L-1$ as defined below. Each of these lists is defined by the neighborhood of v induced by the sampled edges in S_i.
  - If $i \leq \ell(v)$, then we have:
    - $\textsc{Friends}_i[v, j]$ is empty for all $j > i$.
    - $\textsc{Friends}_i[v, j] = \mathcal{N}_v(Z_j, S_i)$ for $j = i$.
    - $\textsc{Friends}_i[v, j] = \mathcal{N}_v(V_j, S_i)$ for all $j < i$.
  - Else if $i > \ell(v)$, then we have:
    - $\textsc{Friends}_i[v, j]$ is empty for all $j > \ell(v)$.
    - $\textsc{Friends}_i[v, j] = \mathcal{N}_v(Z_j, S_i)$ for $j = \ell(v)$.
    - $\textsc{Friends}_i[v, j] = \mathcal{N}_v(V_j, S_i)$ for all $j < \ell(v)$.

For every node $v \in V$, we maintain a counter $\textsc{Degree}_i[v]$. If $\ell(v) \geq i$, then this counter equals the number of nodes in $\textsc{Friends}_i[v, i]$. Else if $\ell(v) < i$, then this counter equals zero. Further, we maintain a doubly linked list $\textsc{Dirty-Nodes}[i]$. This list consists of all the nodes $v \in V$ having either
$\left\{\textsc{Degree}_i[v] > (1-\epsilon)^2 \alpha c \log n \text{ and } \ell(v) = i\right\}$ or $\left\{\textsc{Degree}_i[v] < (1+\epsilon)^2 c \log n \text{ and } \ell(v) > i\right\}$.

Implementing the procedure in Figure 1. Fix any $t \in [T', T]$, and consider the i^{th} iteration of the main For loop (Steps 05-10) in Algorithm 1. The purpose of this iteration is to construct the set $Z_{i+1}^{(t)}$, based on the sets $Z_i^{(t)}$ and $S_i^{(t)}$. Below, we state an alternate way of visualizing this iteration.

We scan through the list of nodes u with $\ell(u) = i$ and $D_u(Z_i^{(t)}, S_i^{(t)}) > (1-\epsilon)^2 \alpha c \log n$. While considering each such node u, we increment its level from i to $(i+1)$. This takes care of the Steps (05) and (07). Next, we scan through the list of nodes v with $\ell(v) > i$ and $D_v(Z_i^{(t)}, S_i^{(t)}) < (1+\epsilon)^2 c \log n$. While considering any such node v at level $\ell(v) = j_v > i$ (say), we decrement its level from j_v to i. This takes care of the Steps (06), (08) and (09).

Note that the nodes undergoing a level-change in the preceding paragraph are precisely the ones that appear in the list $\textsc{Dirty-Nodes}[i]$ just before the i^{th} iteration of the main For loop. Thus, we can implement Steps (05-10) as follows: Scan through the nodes y in $\textsc{Dirty-Nodes}[i]$ one after another. While considering any such node y, change its level as per Algorithm 1, and then update the relevant data structures to reflect this change.

Lemma 4.6. *The procedure in Figure 1 can be implemented in $\tilde{O}(n)$ space.*

Proof. (sketch) The amount of space needed is dominated by the number of edges in $\left\{S_i^{(t)}\right\}, i \in [L-1]$. Since $|S_i^{(t)}| \leq s_k$ for each $i \in [L-1]$, the space complexity is $(L-1) \cdot s_k = \tilde{O}(n)$. □

Claim 4.7. *Fix a $t \in [T', T]$ and consider the i^{th} iteration of the main For loop in Figure 1. Consider any two nodes $u, v \in Z_i^{(t)}$ such that (a) the level of u is increased from i to $(i+1)$ in*

Step (07) and (b) the level of v is decreased to i in Steps (08-09). Updating the relevant data structures require $\sum_{i'>i} O(1 + D_y(Z_i^{(t)}, S_{i'}^{(t)}))$ time, where $y = u$ (resp. v) in the former (resp. latter) case.

Proof. (sketch) Follows from the fact that we only need to update the lists $\textsc{Friends}_{i'}[x, j]$ where $i' > i$, $x \in \{y\} \cup \mathcal{N}_y(Z_i^{(t)}, S_{i'}^{(t)})$ and $j \in \{1, \ldots, L-1\}$. □

4.3 Bounding the amortized update time

Potential function. To determine the amortized update time we use a potential function $\mathcal{B}$ as defined in equation 4. Note that the potential $\mathcal{B}$ is uniquely determined by the assignment of the nodes $v \in V$ to the levels $[L]$ and by the content of the random sets $S_1, \ldots, S_{(L-1)}$. For all nodes $v \in V$, we define:

$$\Gamma_i(v) = \max(0, (1-\epsilon)^2 \alpha c \log n - D_v(Z_i, S_i)) \tag{1}$$

$$\Phi(v) = (L/\epsilon) \cdot \sum_{i=1}^{\ell(v)-1} \Gamma_i(v) \tag{2}$$

For all $u, v \in V$, let $f(u,v) = 1$ if $\ell(u) = \ell(v)$ and 0 otherwise. Also, let $r_{uv} = \min(\ell(u), \ell(v))$. For all $i \in [L-1]$, $(u,v) \in S_i$, we define:

$$\Psi_i(u,v) = \begin{cases} 0 & \text{if } r_{uv} \geq i; \\ 2 \cdot (i - r_{uv}) + f(u,v) & \text{otherwise.} \end{cases}, \tag{3}$$

$$\mathcal{B} = \sum_{v \in V} \Phi(v) + \sum_{i=1}^{(L-1)} \sum_{e \in S_i} \Psi_i(e) \tag{4}$$

Below, we show that an event $\mathcal{F}$ holds with high probability (Definition 4.8, Claim 4.9). Next, conditioned on this event, we show that our algorithm has $O(\text{poly} \log n)$ amortized update time (Lemma 4.10).

Definition 4.8. *For all $i, i' \in [L-1]$, $i < i'$, let $\mathcal{F}_{i,i'}^{(t)}$ be the event that: $\left\{D_v(Z_i^{(t)}, S_{i'}^{(t)}) \geq \frac{(1-\epsilon)^4}{(1+\epsilon)^2} \cdot (\alpha c \log n)\right.$ for all $v \in A_i^{(t)}\}$, and $\left\{D_v(Z_i^{(t)}, S_{i'}^{(t)}) \leq \frac{(1+\epsilon)^4}{(1-\epsilon)^2} \cdot c \log n\right.$ for all $v \in B_i^{(t)}\}$. Define $\mathcal{F}^{(t)} = \bigcap_{i,i'} \mathcal{F}_{i,i'}^{(t)}$.*

Claim 4.9. *Define the event $\mathcal{F} = \bigcap_{t=T'}^{T} \mathcal{F}^{(t)}$. The event $\mathcal{F}$ holds with high probability.*

Proof. (sketch) Fix any $1 \leq i < i' \leq L-1$, any $t \in [T', T]$, and condition on any instance of the random set $Z_i^{(t)}$. By Lemma 4.4, the random sets $S_i^{(t)}, S_{i'}^{(t)}$ are chosen independently of $Z_i^{(t)}$. Further, for all $v \in Z_i^{(t)}$, we have $E[D_v(Z_i^{(t)}, S_i^{(t)})] = E[D_v(Z_i^{(t)}, S_{i'}^{(t)})] = (c \log n / d_k^{(t)}) \cdot D_v(Z_i^{(t)}, E^{(t)})$, and by Lemma 4.3 we can apply a Chernoff bound on this expectation. Thus, applying union bounds over $\{i, i'\}$, we infer that w.h.p. the following condition holds: If $D_v(Z_i^{(t)}, E^{(t)})$ is sufficiently smaller (resp. larger) than $d_k^{(t)}$, then both $D_v(Z_i^{(t)}, S_i^{(t)})$ and $D_v(Z_i^{(t)}, S_{i'}^{(t)})$ are sufficiently smaller (resp. larger) than $c \log n$. The proof follows by deriving a variant of this claim and then applying union bounds over all i, i' and t. □

Lemma 4.10. *Condition on event $\mathcal{F}$. We have (a) $0 \leq \mathcal{B} = \tilde{O}(n)$ at each $t \in [T', T]$, (b) insertion/deletion of an edge in G (ignoring the call to Algorithm 1) changes the potential $\mathcal{B}$ by $\tilde{O}(1)$, and (c) for every constant amount of computation performed while implementing Algorithm 1, the potential $\mathcal{B}$ drops by $\Omega(1)$.*

Theorem 4.2 follows from Lemmata 4.5, 4.6, 4.10 and Claim 4.9. We now focus on proving Lemma 4.10.

Proof of part (a). Follows from three facts. (1) We have $0 \leq \Phi(v) \leq (L/\epsilon) \cdot L \cdot (1-\epsilon)^2 \alpha c \log n = O(\text{poly} \log n)$ for all $v \in V$. (2) We have $0 \leq \Psi_i(u,v) \leq 3L = O(\text{poly} \log n)$ for all $i \in [L-1], (u,v) \in S_i^{(t)}$. (3) We have $|S_i^{(t)}| \leq s_k = O(n \text{ poly} \log n)$ for all $i \in [L-1]$.

Proof of part (b). By Lemma 4.3, insertion/deletion of an edge in G leads to at most two insertions/deletions in the random set S_i, for all $i \in [L-1]$. As $L = O(\text{poly} \log n)$, it suffices to show that for every edge insertion/deletion in any given $S_i^{(t)}$, the potential $\mathcal{B}$ changes by at most $O(\text{poly} \log n)$ (ignoring call to Figure 1).

Towards this end, fix any $i \in [L-1]$, and suppose that a single edge (u,v) is inserted into (resp. deleted from) $S_i^{(t)}$. For each node $v \in V$, this changes the potential $\Phi(v)$ by at most $O(L/\epsilon)$. Additionally, the potential $\Psi_i(u,v) \in [0, 3L]$ is created (resp. destroyed). Summing over all the nodes $v \in V$, we infer that the absolute value of the change in the overall potential $\mathcal{B}$ is at most $O(3L + nL/\epsilon) = O(n \text{ poly} \log n)$.

Proof of part (c). Focus on a single iteration of the FOR loop in Figure 1. Consider two possible operations.

CASE 1: A NODE $v \in Z_i^{(t)}$ IS PROMOTED FROM LEVEL i TO LEVEL $(i+1)$ IN STEP 07 OF FIGURE 1.

This can happen only if $v \in A_i^{(t)}$. Let C denote the amount of computation performed during this step.

$$C \quad = \quad \sum_{i'=(i+1)}^{(L-1)} O\left(1 + D_v(Z_i^{(t)}, S_{i'}^{(t)})\right) \tag{5}$$

Let Δ be the net decrease in the overall potential $\mathcal{B}$ due to this step. We make the following observations.

1. Consider any $i' > i$. For each edge $(u,v) \in S_{i'}^{(t)}$ with $u \in Z_i^{(t)}$, the potential $\Psi_{i'}(u,v)$ decreases by at least one. For every other edge $e \in S_{i'}^{(t)}$, the potential $\Psi_{i'}(e)$ remains unchanged.

2. For each $i' \in [i]$ and each edge $e \in S_{i'}^{(t)}$, the potential $\Psi_{i'}(e)$ remains unchanged.

3. Since the node v is being promoted to level $(i+1)$, we have $D_v(Z_i^{(t)}, S_i^{(t)}) \geq (1-\epsilon)^2 \alpha c \log n$. Thus, the potential $\Phi(v)$ remains unchanged. For each node $u \neq v$, the potential $\Phi(u)$ can only decrease.

Taking into account all these observations, we infer the following inequality.

$$\Delta \quad \geq \quad \sum_{i'=(i+1)}^{(L-1)} D_v(Z_i^{(t)}, S_{i'}^{(t)}) \tag{6}$$

Since $v \in A_i^{(t)}$, and since we have conditioned on the event $\mathcal{F}^{(t)}$ (see Definition 4.8), we get:

$$D_v(Z_i^{(t)}, S_{i'}^{(t)}) \quad > \quad 0 \quad \text{for all } i' \in [i+1, L-1]. \tag{7}$$

Equations (5), (6), (7) imply that the decrease in $\mathcal{B}$ is sufficient to pay for the computation performed.

Case 2: A node $v \in Z_i^{(t)}$ is demoted from level $j > i$ to level i in Steps (08-09) of Figure 1.

This can happen only if $v \in B_i^{(t)}$. Let C denote the amount of computation performed during this step. By Claim 4.7, we have

$$C \quad = \quad \sum_{i'=(i+1)}^{(L-1)} O(1 + D_v(Z_i^{(t)}, S_{i'}^{(t)})) \tag{8}$$

Let $\gamma = (1+\epsilon)^4/(1-\epsilon)^2$. Equation (9) holds since $v \in B_i^{(t)}$ and since we conditioned on the event $\mathcal{F}$. Equation (10) follows from equations (8), (9) and the facts that γ, c are constants,

$$D_v(Z_i^{(t)}, S_{i'}^{(t)}) \quad \leq \quad \gamma c \log n \text{ for all } i' \in [i, L-1] \tag{9}$$

$$C \quad = \quad O(L \log n) \tag{10}$$

Let Δ be the net decrease in the overall potential $\mathcal{B}$ due to this step. We make the following observations.

1. By eq. (9), the potential $\Phi(v)$ decreases by at least $(j-i) \cdot (L/\epsilon) \cdot ((1-\epsilon)^2\alpha - \gamma) \cdot (c \log n)$.
2. For $u \in V \setminus \{v\}$ and $i' \in [1, i] \cup [j+1, L-1]$, the potential $\Gamma_{i'}(u)$ remains unchanged. This observation, along with equation (9), implies that the sum $\sum_{u \neq v} \Phi(u)$ increases by at most $(L/\epsilon) \cdot \sum_{i'=(i+1)}^{j} D_v(Z_i^{(t)}, S_{i'}^{(t)}) \leq (j-i) \cdot (L/\epsilon) \cdot (\gamma c \log n)$.
3. For every $i' \in [1, i]$, and $e \in S_{i'}^{(t)}$ the potential $\Psi_{i'}(e)$ remains unchanged. Next, consider any $i' \in [i+1, L-1]$. For each edge $(u, v) \in S_{i'}^{(t)}$ with $u \in Z_i^{(t)}$, the potential $\Psi_{i'}(u, v)$ increases by at most $3(j-i)$. For every other edge $e \in S_{i'}^{(t)}$, the potential $\Psi_{i'}(e)$ remains unchanged. These observations, along with equation (9), imply that the sum $\sum_{i'} \sum_{e \in S_{i'}} \Psi_{i'}(e)$ increases by at most $\sum_{i'=(i+1)}^{(L-1)} 3(j-i) \cdot D_v(Z_i^{(t)}, S_{i'}^{(t)}) \leq (j-i) \cdot (3L) \cdot (\gamma c \log n)$.

Taking into account all these observations, we get:

$$\begin{aligned} \Delta \quad &\geq \quad (j-i)(L/\epsilon)((1-\epsilon)^2\alpha - \gamma)(c \log n) \\ &\quad\quad -(j-i)(L/\epsilon)(\gamma c \log n) - (j-i)(3L)(\gamma c \log n) \\ &= \quad (j-i) \cdot (L/\epsilon) \cdot ((1-\epsilon)^2\alpha - 2\gamma - 3\epsilon\gamma) \cdot (c \log n) \\ &\geq \quad Lc \log n \end{aligned} \tag{11}$$

The last inequality holds since $(j-i) \geq 1$ and $\alpha \geq (\epsilon + (2+3\epsilon)\gamma)/(1-\epsilon)^2 = 2 + \Theta(\epsilon)$, for some sufficiently small constant $\epsilon \in (0, 1)$. From eq. (10) and (11), we conclude that the net decrease in the overall potential $\mathcal{B}$ is sufficient to pay for the cost of the computation performed.

5 Open problems

An obvious question is whether the $(4+\epsilon)$ approximation ratio provided by our algorithm is tight. In particular, it will be interesting if one can improve the approximation ratio to $(2+\epsilon)$ to match the case where an update time is not a concern. Getting this approximation ratio even with larger space complexity is still interesting. (Epasto et al. [**?**] almost achieved this except that they have to assume that the deletions happen uniformly at random.) It is equally interesting to show a hardness result. Currently, there is only a hardness result for maintaining the optimal solution [**?**]. It will be interesting to show a hardness result for approximation algorithms. Another interesting question is whether a similar result to ours can be achieved with polylogarithmic *worst-case* update time. Finally, a more general question is whether one can obtain space- and time-efficient fully-dynamic algorithm like ours for other fundamental graph problems, e.g. maximum matching and single-source shortest paths.

References

[1] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 459–467, 2012. 2

[2] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 5–14, 2012. 2, 4

[3] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Spectral sparsification in dynamic graph streams. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 16th International Workshop, APPROX 2013, and 17th International Workshop, RANDOM 2013, Berkeley, CA, USA, August 21-23, 2013. Proceedings*, pages 1–10, 2013. 2

[4] Albert Angel, Nick Koudas, Nikos Sarkas, and Divesh Srivastava. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. *PVLDB*, 5(6):574–585, 2012. 2

[5] Bahman Bahmani, Ashish Goel, and Kamesh Munagala. Efficient primal-dual algorithms for MapReduce. In *Algorithms and Models for the Web Graph - 11th International Workshop, WAW 2014, Proceedings*, 2014. 4

[6] Bahman Bahmani, Ravi Kumar, and Sergei Vassilvitskii. Densest subgraph in streaming and mapreduce. *PVLDB*, 5(5):454–465, 2012. 2, 3, 4, 41

[7] Gerth Stølting Brodal and Rolf Fagerberg. Dynamic representation of sparse graphs. In *Algorithms and Data Structures, 6th International Workshop, WADS '99, Vancouver, British Columbia, Canada, August 11-14, 1999, Proceedings*, pages 342–351, 1999. 4

[8] Amit Chakrabarti, Subhash Khot, and Xiaodong Sun. Near-optimal lower bounds on the multi-party communication complexity of set disjointness. In *18th Annual IEEE Conference on Computational Complexity (Complexity 2003), 7-10 July 2003, Aarhus, Denmark*, pages 107–117, 2003. 41

[9] Moses Charikar. Greedy approximation algorithms for finding dense components in a graph. In Klaus Jansen and Samir Khuller, editors, *APPROX*, volume 1913 of *Lecture Notes in Computer Science*, pages 84–95. Springer, 2000. 2

[10] Bernard Chazelle, Ronitt Rubinfeld, and Luca Trevisan. Approximating the minimum spanning tree weight in sublinear time. *SIAM J. Comput.*, 34(6):1370–1379, 2005. 3, 41

[11] Graham Cormode, S. Muthukrishnan, Ke Yi, and Qin Zhang. Optimal sampling from distributed streams. In *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2010, June 6-11, 2010, Indianapolis, Indiana, USA*, pages 77–86, 2010. 3, 41, 42

[12] Artur Czumaj and Christian Sohler. Sublinear-time algorithms. In *Property Testing*, pages 41–64, 2010. 41

[13] Yon Dourisboure, Filippo Geraci, and Marco Pellegrini. Extraction and classification of dense communities in the web. In Carey L. Williamson, Mary Ellen Zurko, Peter F. Patel-Schneider, and Prashant J. Shenoy, editors, *WWW*, pages 461–470. ACM, 2007. 2

[14] Paul Erdös, Arthur H Stone, et al. On the structure of linear graphs. *Bull. Amer. Math. Soc*, 52:1087–1091, 1946. 4

[15] Shimon Even and Yossi Shiloach. An On-Line Edge-Deletion Problem. *Journal of the ACM*, 28(1):1–4, 1981. 2

[16] David Gibson, Ravi Kumar, and Andrew Tomkins. Discovering large dense subgraphs in massive graphs. In *VLDB*, pages 721–732, 2005. 2

[17] Ashish Goel, Michael Kapralov, and Sanjeev Khanna. Perfect matchings in o(nlog n) time in regular bipartite graphs. *SIAM J. Comput.*, 42(3):1392–1404, 2013. Announced at STOC 2010. 3, 41

[18] A. V. Goldberg. Finding a maximum density subgraph. Technical report, Berkeley, CA, USA, 1984. 2

[19] Oded Goldreich. A brief introduction to property testing. In *Studies in Complexity and Cryptography*, pages 465–469. 2011. 41

[20] Oded Goldreich. Introduction to testing graph properties. In *Studies in Complexity and Cryptography*, pages 470–506. 2011. 41

[21] Hossein Jowhari, Mert Saglam, and Gábor Tardos. Tight bounds for lp samplers, finding duplicates in streams, and related problems. In *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2011, June 12-16, 2011, Athens, Greece*, pages 49–58, 2011. 8, 28, 31

[22] Michael Kapralov, Yin Tat Lee, Cameron Musco, Christopher Musco, and Aaron Sidford. Single pass spectral sparsification in dynamic streams. In *55th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2014, 28-22 October, 2014, Philadelphia, PA, USA*, pages 561–570, 2014. 2

[23] Michael Kapralov and David P. Woodruff. Spanners and sparsifiers in dynamic streams. In *ACM Symposium on Principles of Distributed Computing, PODC '14, Paris, France, July 15-18, 2014*, pages 272–281, 2014. 2

[24] Bruce M. Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1131–1142, 2013. 2

[25] Eugene Lawler. *Combinatorial Optimization: Networks and Matroids*. Saunders College Publishing, Fort Worth, 1976. 2

[26] Victor E. Lee, Ning Ruan, Ruoming Jin, and Charu C. Aggarwal. A survey of algorithms for dense subgraph discovery. In *Managing and Mining Graph Data*, pages 303–336. 2010. 2

[27] David W Matula. A min-max theorem for graphs with application to graph coloring. In *SIAM REVIEW*, volume 10, page 481. SIAM PUBLICATIONS 3600 UNIV CITY SCIENCE CENTER, PHILADELPHIA, PA 19104-2688, 1968. 4

[28] J. Ian Munro and Mike Paterson. Selection and sorting with limited storage. *Theor. Comput. Sci.*, 12:315–323, 1980. Announced at FOCS'78. 2

[29] Krzysztof Onak. Sublinear graph approximation algorithms. In *Property Testing*, pages 158–166, 2010. 41

[30] Anna Pagh and Rasmus Pagh. Uniform hashing in constant time and optimal space. *SIAM J. Comput.*, 38(1):85–96, 2008. 10, 29, 31

[31] Dana Ron. Algorithmic and analysis techniques in property testing. *Foundations and Trends in Theoretical Computer Science*, 5(2):73–205, 2009. 41

[32] Ronitt Rubinfeld and Asaf Shapira. Sublinear time algorithms. *SIAM J. Discrete Math.*, 25(4):1562–1588, 2011. 41

[33] Ahmet Erdem Sariyüce, Bugra Gedik, Gabriela Jacques-Silva, Kun-Lung Wu, and Ümit V. Çatalyürek. Streaming algorithms for k-core decomposition. *PVLDB*, 6(6):433–444, 2013. 4

[34] George Szekeres and Herbert S Wilf. An inequality for the chromatic number of a graph. *Journal of Combinatorial Theory*, 4(1):1–3, 1968. 4

[35] Lei Tang and Huan Liu. Graph mining applications to social network analysis. In *Managing and Mining Graph Data*, pages 487–513. 2010. 2

[36] Charalampos E. Tsourakakis. A novel approach to finding near-cliques: The triangle-densest subgraph problem. *CoRR*, abs/1405.1477, 2014. 2

Part II
FULL DETAILS

6 Notations and Preliminaries

The input graph $G = (V, E)$ has $|V| = n$ nodes and $|E| = m$ edges. Let $\mathcal{N}_v = \{u \in V : (u, v) \in E\}$ and $D_v = |\mathcal{N}_v|$ respectively denote the set of neighbors and the degree of a node $v \in V$. Let $G(S)$ denote the subgraph of G induced by the nodes in $S \subseteq V$. Given any two subsets $S \subseteq V, E' \subseteq E$, define $\mathcal{N}_u(S, E') = \{v \in \mathcal{N}_u \cap S : (u, v) \in E'\}$ and $D_u(S, E') = |\mathcal{N}_u(S, E')|$. To ease notation, we write $\mathcal{N}_u(S)$ and $D_u(S)$ instead of $\mathcal{N}_u(S, E)$ and $D_u(S, E)$. For a nonempty subset $S \subseteq V$, its density and average degree are defined as $\rho(S) = |E(S)|/|S|$ and $\delta(S) = \sum_{v \in S} D_v(S)/|S|$ respectively. The next lemma, which shows that the average degree of a set is twice its density, follows from a standard counting argument.

Lemma 6.1. *For all $S \subseteq V$, we have $\delta(S) = 2 \cdot \rho(S)$.*

In the "Densest Subgraph Problem", the goal is to find a subset $S \subseteq V$ that maximizes $\rho(S)$. Further, a subset $S \subseteq V$ is called a γ-approximate densest subgraph, for $\gamma \geq 1$, iff $\gamma \cdot \rho(S) \geq \max_{S' \subseteq V} \rho(S')$.

Lemma 6.2. *Let $S^* \subseteq V$ be a subset of nodes with maximum density, i.e., $\rho(S^*) \geq \rho(S)$ for all $S \subseteq V$. Then $D_v(S^*) \geq \rho(S^*)$ for all $v \in S^*$. Thus, the degree of each node in $G(S^*)$ is at least the density of S^*.*

Proof. Suppose that there is a node $v \in S^*$ with $D_{S^*}(v) < \rho(S^*)$. Define the set $S' \leftarrow S^* \setminus \{v\}$. We derive the following bound on the average degree in S'.

$$\begin{aligned}
\delta(S') &= \frac{\sum_{u \in S'} D_{S'}(u)}{|S'|} \\
&= \frac{\sum_{u \in S^*} D_{S^*}(u) - 2 \cdot D_{S^*}(v)}{|S^*| - 1} \\
&= \frac{\delta(S^*) \cdot |S^*| - 2 \cdot D_{S^*}(v)}{|S^*| - 1} \\
&> \frac{\delta(S^*) \cdot |S^*| - \delta(S^*)}{|S^*| - 1} && \text{(since } D_{S^*}(v) < \rho(S^*) = \delta(S^*)/2\text{)} \\
&= \delta(S^*)
\end{aligned}$$

Since $\delta(S') > \delta(S^*)$, we infer that $\rho(S') > \rho(S^*)$. But this contradicts the assumption that the subset of nodes S^* has maximum density. Thus, we conclude that $D_{S^*}(v) \geq \rho(S^*)$ for every node $v \in S^*$. □

Definition 6.3 ((α, d, L)-decomposition)**.** *Fix any $\alpha \geq 1$, $d \geq 0$, and any positive integer L. Consider a family of subsets $Z_1 \supseteq \cdots \supseteq Z_L$. The tuple $(Z_1, \ldots, Z_L)$ is an (α, d, L)-decomposition of the input graph $G = (V, E)$ iff $Z_1 = V$ and the following condition holds for every $i \in [L - 1]$:*

- *We have $Z_{i+1} \supseteq \{v \in Z_i : D_v(Z_i) > \alpha d\}$ and $Z_{i+1} \cap \{v \in Z_i : D_v(Z_i) < d\} = \emptyset$.*

Given an (α, d, L)-decomposition $(Z_1, \ldots, Z_L)$, we define $V_i = Z_i \setminus Z_{i+1}$ for all $i \in [L - 1]$, and $V_i = Z_i$ for $i = L$. We say that the nodes in V_i constitute the i^{th} level of this decomposition. We also denote the level of a node $v \in V$ by $\ell(v)$. Thus, we have $\ell(v) = i$ whenever $v \in V_i$.

Theorem 6.4. *Fix any $\alpha \geq 1$, $d \geq 0$, $\epsilon \in (0,1)$, $L \leftarrow 2+\lceil \log_{(1+\epsilon)} n \rceil$. Let $d^* \leftarrow \max_{S \subseteq V} \rho(S)$ be the maximum density of any subgraph in $G=(V,E)$, and let $(Z_1, \ldots, Z_L)$ be an (α, d, L)-decomposition of $G=(V,E)$. We have: (1) If $d > 2(1+\epsilon)d^*$, then $Z_L = \emptyset$, and (2) if $d < d^*/\alpha$, then $Z_L \neq \emptyset$.*

Proof. (1) Suppose that $d > 2(1+\epsilon)d^*$. Consider any level $i \in [L-1]$, and note that $\delta(Z_i) = 2 \cdot \rho(Z_i) \leq 2 \cdot \max_{S \subseteq V} \rho(S) = 2d^* < d/(1+\epsilon)$. It follows that the number of nodes v in $G(Z_i)$ with degree $D_v(Z_i) \geq d$ is less than $|Z_i|/(1+\epsilon)$, as otherwise $\delta(Z_i) \geq d/(1+\epsilon)$. Let us define the set $C_i = \{v \in Z_i : D_v(Z_i) < d\}$. We have $|Z_i \setminus C_i| \leq |Z_i|/(1+\epsilon)$. Now, from Definition 6.3 we have $Z_{i+1} \cap C_i = \emptyset$, which, in turn, implies that $|Z_{i+1}| \leq |Z_i \setminus C_i| \leq |Z_i|/(1+\epsilon)$. Thus, for all $i \in [L-1]$, we have $|Z_{i+1}| \leq |Z_i|/(1+\epsilon)$. Multiplying all these inequalities, for $i=1$ to $L-1$, we conclude that $|Z_L| \leq |Z_1|/(1+\epsilon)^{L-1}$. Since $|Z_1| = |V| = n$ and $L = 2+\lceil \log_{(1+\epsilon)} n \rceil$, we get $|Z_L| \leq n/(1+\epsilon)^{(1+\log_{(1+\epsilon)} n)} < 1$. This can happen only if $Z_L = \emptyset$.

(2) Suppose that $d < d^*/\alpha$, and let $S^* \subseteq V$ be a subset of nodes with highest density, i.e., $\rho(S^*) = d^*$. We will show that $S^* \subseteq Z_i$ for all $i \in \{1, \ldots, L\}$. This will imply that $Z_L \neq \emptyset$. Clearly, we have $S^* \subseteq V = Z_1$. By induction hypothesis, assume that $S^* \subseteq Z_i$ for some $i \in [L-1]$. We show that $S^* \subseteq Z_{i+1}$. By Lemma 6.2, for every node $v \in S^*$, we have $D_v(Z_i) \geq D_v(S^*) \geq \rho(S^*) = d^* > \alpha d$. Hence, from Definition 6.3, we get $v \in Z_{i+1}$ for all $v \in S^*$. This implies that $S^* \subseteq Z_{i+1}$. □

Corollary 6.5. *Fix α, ϵ, L, d^* as in Theorem 6.4. Let $\pi, \sigma > 0$ be any two numbers satisfying $\alpha \cdot \pi < d^* < \sigma/(2(1+\epsilon))$. Discretize the range $[\pi, \sigma]$ into powers of $(1+\epsilon)$, by defining $d_k \leftarrow (1+\epsilon)^{k-1} \cdot \pi$ for every $k \in [K]$, where K is any integer strictly greater than $\lceil \log_{(1+\epsilon)} (\sigma/\pi) \rceil$. For every $k \in [K]$, construct an (α, d_k, L)-decomposition $(Z_1(k), \ldots, Z_L(k))$ of $G=(V,E)$. Let $k' \leftarrow \min\{k \in [K] : Z_L(k) = \emptyset\}$. Then we have the following guarantee: $d^*/\alpha \leq d_{k'} \leq 2(1+\epsilon)^2 \cdot d^*$.*

6.1 Concentration bounds

We will use the following concentration bounds in our proofs.

Theorem 6.6. *(Chernoff bound-I) Consider a set of mutually independent random variables $\{X_1, \ldots, X_t\}$ such that $X_i \in [0,1]$ for all $i \in \{1, \ldots, t\}$. Let $X = \sum_{i=1}^{t} X_i$ be the sum of these random variables. Then we have $\Pr[X > (1+\epsilon)\mu] \leq e^{-\epsilon^2 \mu/3}$ whenever $E[X] \leq \mu$.*

Theorem 6.7. *(Chernoff bound-II) Consider a set of mutually independent random variables $\{X_1, \ldots, X_t\}$ such that $X_i \in [0,1]$ for all $i \in \{1, \ldots, t\}$. Let $X = \sum_{i=1}^{t} X_i$ be the sum of these random variables. Then we have $\Pr[X < (1-\epsilon)\mu] \leq e^{-\epsilon^2 \mu/2}$ whenever $E[X] \geq \mu$.*

Definition 6.8. *(Negative association) A set of random variables $\{X_1, \ldots, X_t\}$ are negatively associated iff for all disjoint subsets $I, J \subseteq \{1, \ldots, t\}$ and all non-decreasing functions f and g, we have $E[f(X_i, i \in I) \cdot g(X_j, j \in J)] \leq E[f(X_i, i \in I)] \cdot E[g(X_j, j \in J)]$.*

Theorem 6.9. *(Chernoff bound with negative dependence) The Chernoff bounds, as stated in Theorems 6.6 and 6.7, hold even if the random variables $\{X_1, \ldots, X_t\}$ are negatively associated.*

7 A dynamic algorithm in $O(n+m)$ space and $\tilde{O}(1)$ update time

In this section, we show how to maintain a $(4+O(\epsilon))$-approximation to the density of the densest subgraph, deterministically and using $O(n+m)$ space, in amortized time $O(\log^2 n/\epsilon^3)$ per update.

Set $\pi = 1/(4n)$, $\sigma = 4n$, $L = 2+\lceil \log_{(1+\epsilon)} n \rceil$, and $\alpha = 2+10\epsilon$, where $\epsilon \in (0,1)$ is some small constant. Note that $\alpha \cdot \pi < d^* < \sigma/(2(1+\epsilon))$, where d^* is the optimal density in the input graph. As in Corollary 6.5, discretize the range $[\pi, \sigma]$ in powers of $(1+\epsilon)$ by defining the values $\{d_k\}, k \in [K]$.

We show in Section 7.1 how to maintain an (α, d_k, L)-decomposition of G for each $k \in [K]$ in time $O(L/\epsilon) = O(\log n/\epsilon^2)$ per edge update. Since it is possible to ensure that $K = O(\log n/\epsilon)$, the total time for all K decompositions is $O(\log^2 n/\epsilon^3)$ per update operation. By Corollary 6.5, this gives a $2\alpha(1+\epsilon)^2 = 4 + O(\epsilon)$-approximation to the optimal density in G, for sufficiently small ϵ.

7.1 Dynamically maintaining an (α, d, L)-decomposition

We present a deterministic data structure that is initially given α, d, L, a graph $G = (V, E)$ with $|V| = n$, $E = \emptyset$, and supports the following operations:

- *Insert*(u, v)*:* Insert the edge (u, v) into the graph.
- *Delete* (u, v)*:* Delete the edge (u, v) from the graph.

We want to maintain an (α, d, L)-decomposition of the graph $G = (V, E)$ at each time-step.

Data Structures. We use the following data structures.

1. Every node $v \in V$ maintains L lists $\textsc{Friends}_i[v]$, for $i \in \{1, \ldots, L\}$. For $i < \ell(v)$, the list $\textsc{Friends}_i[v]$ consists of the neighbors of v that are at level i (given by the set $\mathcal{N}_v(V_i)$). For $i = \ell(v)$, the set $\textsc{Friends}_i[v]$ consists of the neighbors of v that are at level i or above (given by the set $\mathcal{N}_v(Z_i)$). For $i > \ell(v)$, the list $\textsc{Friends}_i[v]$ is empty. Each list is stored in a doubly linked list together with its size, $\textsc{Count}_i[v]$. Using appropriate pointers, we can insert or delete a given node to or from a concerned list in constant time.
2. The counter $\textsc{Level}[v]$ keeps track of the level of the node v.

Algorithm. If a node violates one of the conditions of an (α, d, L)-decomposition (see Definition 6.3), then we call the node "dirty", else the node is called "clean". Specifically a node y at level $\ell(y) = i$ is dirty iff either $D_y(Z_i) > \alpha d$ or $D_y(Z_{i-1}) < d$. Initially, the input graph $G = (V, E)$ is empty, every node $v \in V$ is at level 1, and every node is clean.

When an edge (u, v) is inserted/deleted, we first update the $\textsc{Friends}$ lists of u and v by adding or removing in constant time. Next we check whether u or v are dirty. If so, we run the RECOVER() procedure described in Figure 2. Note that a single iteration of the $\textsc{While}$ loop (Steps 01-05) may change the status of some more nodes from clean to dirty (or vice versa). If and when the procedure terminates, however, every node is clean by definition.

01.	$\textsc{While}$ there exists a dirty node y
02.	$\quad\textsc{If}$ $D_y(Z_{\ell(y)}) > \alpha d$ and $\ell(y) < L$, $\textsc{Then}$
03.	$\quad\quad$ Increment the level of y by setting $\ell(y) \leftarrow \ell(y) + 1$.
04.	$\quad\textsc{Else if}$ $D_y(Z_{\ell(y)-1}) < d$ and $\ell(y) > 1$, $\textsc{Then}$
05.	$\quad\quad$ Decrement the level of y by setting $\ell(y) \leftarrow \ell(y) - 1$.

Figure 2: RECOVER().

Analyzing the space complexity. Since each edge in G appears in two linked lists (corresponding to each of its endpoints), the space complexity of the data structure is $O(n + m)$, where $m = |E|$.

Analysis of the Update Time. Each update operation takes constant time plus the time for the RECOVER() procedure. We show below that the total time spent in procedure RECOVER() during t update operations is $O(tL/\epsilon)$.

The following theorem summarizes the main result.

Theorem 7.1. *For every polynomially bounded $\alpha \geq 2+3\epsilon$, we can maintain an (α, d, L)-decomposition of $G = (V, E)$ during a sequence of t update operations starting from an empty graph in time $O(tL/\epsilon)$. The space complexity of the data structure at a given time-step is $O(n + m)$, where $m = |E|$ denotes the number of edges in the input graph at that time-step.*

Potential Function. To determine the amortized update time we use a potential function $\mathcal{B}$. Let $f(u, v) = 1$ if $l(u) = l(v)$ and let it be 0 otherwise. We define $\mathcal{B}$ and the node and edge potentials $\Phi(v)$ and $\Psi(u, v)$ as follows.

$$\mathcal{B} = \sum_{v \in V} \Phi(v) + \sum_{e \in E} \Psi(e) \tag{12}$$

$$\Phi(v) = \frac{1}{\epsilon} \sum_{i=1}^{\ell(v)-1} \max(0, \alpha d - D_v(Z_i)) \quad \text{for all nodes } v \in V \tag{13}$$

$$\Psi(u, v) = 2(L - \min(\ell(u), \ell(v))) + f(u, v) \quad \text{for all edges } (u, v) \in E \tag{14}$$

It is easy to check that all these potentials are nonnegative, and that they are uniquely defined by the partition $V_1, \ldots, V_L$ of the set of nodes V.

The Analysis. Initially, the input graph G is empty and the total potential $\mathcal{B}$ is zero. We show that (a) insertion/deletion of an edge (excluding subroutine RECOVER()) increases the total potential by at most $3L/\epsilon$, and (b) for each unit of computation performed by procedure RECOVER() in Figure 2, the total potential decreases by at least $\Omega(1)$. Since the total potential remains always nonnegative, these two conditions together imply an amortized update time of $O(L/\epsilon)$.

Insertion. The insertion of edge (u, v) creates a new potential $\Psi(u, v)$ with value at most $3L$. Further, the potentials $\Phi(u)$ and $\Phi(v)$ do not increase, and the potentials associated with all other nodes and edges remain unchanged. Thus, the net increase in the potential $\mathcal{B}$ is at most $3L$.

Deletion. The deletion of edge (u, v) destroys the (nonnegative) potential $\Psi(u, v)$. Further, each of the potentials $\Phi(u)$ and $\Phi(v)$ increases by at most L/ϵ, and the potentials of all other nodes and edges remain unchanged. Thus, the net increase in the potential $\mathcal{B}$ is at most $2L/\epsilon$.

Analyzing the subroutine RECOVER(). It remains to relate the change in the potential $\mathcal{B}$ with the amount of computation performed. Towards this end, we will analyze any single iteration of the WHILE loop in Figure 2. During this iteration, a dirty node y either increments its level by one unit, or decrements its level by one unit. Accordingly, we consider two possible events.

EVENT 1: A DIRTY NODE y CHANGES ITS LEVEL FROM i TO $(i+1)$.

First, we upper bound the amount of computation performed during this event. Our algorithm scans through the list $\text{FRIENDS}_i[y]$ and identifies the neighbors of y that are at level $(i+1)$ or above. For every such node $x \in \mathcal{N}_y \cap Z_{i+1}$, we need to (a) remove y from the list $\text{FRIENDS}_i[x]$ and add y to the list $\text{FRIENDS}_{i+1}[x]$, (b) increment the counter $\text{COUNT}_{i+1}[x]$ by one unit, (c) add x to the list $\text{FRIENDS}_{i+1}[y]$ and remove x from the list $\text{FRIENDS}_i[y]$, (d) decrement the counter $\text{COUNT}_i[y]$

by one unit and increment the counter $\textsc{Count}_{i+1}[y]$ by one unit. Finally, we set $\textsc{Level}[y] \leftarrow i+1$. Overall, $O(1 + D_y(Z_i))$ units of computation are performed during this event.

Next, we lower bound the net decrease in the $\mathcal{B}$ due to this event. We first discuss the node potentials.

- (a) Since the node y gets promoted to level $i+1$, we must have $D_y(Z_i) > \alpha d$, which implies that $\max(0, \alpha d - D_y(Z_i)) = 0$, so that the potential $\Phi(y)$ does not change.
- (b) The potential of a node $x \in \mathcal{N}_y$ can only decrease.
- (b) The potential of a node $x \notin \{y \cup \mathcal{N}_y\}$ does not change.

Accordingly, the sum $\sum_{v \in V} \Phi(v)$ does not increase.

Next we consider the edge potentials. Consider a node $x \in \mathcal{N}_y$.

- If $\ell(x) < i$, the potential of the edge (x, y) does not change.
- If $\ell(x) = i$, the potential of (x, y) is $2(L-i)+1$ before the level change and $2(L-i)$ afterwards, i.e., the potential drops by one.
- If $\ell(x) = i+1$, the potential of (x, y) is $2(L-i)$ before the level change and $2(L-(i+1))+1 = 2(L-i)-1$ afterwards, i.e., it drops by one.
- If $\ell(x) > i+1$, the potential of (x, y) is $2(L-i)$ before the level change and $2(L-(i+1))$ afterwards, i.e., it drops by two.
- The potentials associated with all edges remain unchanged.

Thus, the sum $\sum_{e \in E} \Psi(e)$ drops by at least $D_y(Z_i)$.

We infer that the net decrease in the overall potential $\mathcal{B}$ is at least $D_y(Z_i)$. Note that $D_y(Z_i) > 0$ (for otherwise the node y would not have been promoted to level $i+1$). It follows that the net decrease in $\mathcal{B}$ is sufficient to pay for the cost of the computation performed, which, as shown above, is $O(1 + D_y(Z_i))$.

Event 2: A dirty node y changes its level from level i to $(i-1)$.

First, we upper bound the amount of computation performed during this event. Our algorithm scans through the nodes in the list $\textsc{Friends}_i[y]$. For each such node $x \in \mathcal{N}_y \cap Z_i$, we need to (a) remove y from the list $\textsc{Friends}_i[x]$ and add y to the list $\textsc{Friends}_{i-1}[x]$ and (b) decrement the counter $\textsc{Count}_i[x]$. Finally, we need to add all the nodes in $\textsc{Friends}_i[y]$ to the list $\textsc{Friends}_{i-1}[y]$, make $\textsc{Friends}_i[y]$ into an empty list, and set $\textsc{Counter}_i[y]$ to zero. Finally, we set $\textsc{Level}[y] \leftarrow i-1$. Overall, $O(1 + D_y(Z_i))$ units of computation are performed during this event.

Next, we lower bound the net decrease in the overall potential $\mathcal{B}$ due to this event. We first consider the changes in the node potentials.

- (a) Since the node y was demoted to level $i-1$, we must have $D_v(Z_{i-1}) < d$. Accordingly, the potential $\Phi(y)$ drops by at least $(\alpha - 1) \cdot (d/\epsilon)$ units due to the decrease in $\ell(y)$.

- (b) For every neighbor x of y, $D_x(Z_i)$ decreases by one while $D_x(Z_j)$ for $j \neq i$ is unchanged. The potential function of a node x considers only the $D_x(Z_j)$ values if $j < \ell(x)$. Thus, only for neighbors x with $\ell(x) > i$ does the potential function change, specifically it increases by at most $1/\epsilon$. Thus the sum $\sum_{x \in \mathcal{N}_y} \Phi(x)$ increases by at most $D_y(Z_{i+1})/\epsilon$. Further, note that $D_y(Z_{i+1})/\epsilon \leq D_y(Z_{i-1})/\epsilon < d/\epsilon$. The last inequality holds since the node y was demoted from level i to level $(i-1)$.

- The potential $\Phi(x)$ remains unchanged for every node $x \notin \{y\} \cup \mathcal{N}_y$.

Thus, the sum $\sum_{v \in V} \Phi(v)$ drops by at least $(\alpha - 1) \cdot (d/\epsilon) - (d/\epsilon) = (\alpha - 2) \cdot (d/\epsilon)$.

We next consider edge potentials. Consider any node $x \in \mathcal{N}_y$.

- If $\ell(x) < i - 1$, the potential of the edge (x, y) does not change.

- If $\ell(x) = i-1$, the potential of (x, y) is $2(L-(i-1))$ before the level change and $2(L-(i-1))+1$ afterwards, i.e., the potential increases by one.

- If $\ell(x) = i$, the potential of (x, y) is $2(L - i) + 1$ before the level change and $2(L - (i - 1)) = 2(L - i) + 2$ afterwards, i.e., it increases by one.

- If $\ell(x) \geq i + 1$, the potential of (x, y) is $2(L - i)$ before the level change and $2(L - (i - 1))$ afterwards, i.e., it increases by two.

- The potentials associated with all other edges remain unchanged.

Thus, the sum $\sum_{e \in E} \Psi(e)$ increases by at most $2D_y(Z_{i-1}) < 2d$.

We infer that the overall potential $\mathcal{B}$ drops by at least $(\alpha - 2) \cdot (d/\epsilon) - 2d = (\alpha - 2 - 2\epsilon) \cdot (d/\epsilon)$. Accordingly, for $\alpha \geq 2 + 3\epsilon$ this potential drop is at least $d \geq D_y(Z_i) + 1$. We conclude that the net drop in the overall potential $\mathcal{B}$ is again sufficient to pay for the cost of the computation performed.

7.2 A high level overview of the potential function based analysis

In hindsight, the intuition behind this potential function is as follows. We maintain a data structure so that the change of the level of a node y from i to $i+1$ or from i to $i-1$ takes time $O(1+D_y(Z_i))$. Ignoring the constant factor (as we can multiply the potential function by this constant), we assume in the following that the cost is simply $1 + D_y(Z_i)$. The basic idea is that the insertion or deletion of an edge should increase the potential function in order to pay for all future level changes. To implement this idea (1) each node gets a potential that increases when an adjacent edge is deleted and that will pay for future level *decreases* of the node, and (2) each edge in G gets a potential that pays for future level *increases* of its end points. We explain how we implement this in more detail next: We know that when a node moves up to level $i + 1$ it has degree at least αd to nodes in Z_i, while when it moves back down it has degree at most d to nodes in Z_i. Assuming that the drop in the nodes degree was caused by the deletion of adjacent edges, the difference of the two, i.e. $(\alpha - 1)d$ has to be used to pay for the cost of a level decreases of a node, which is $1 + D_y(Z_i) \leq d$. This is possible if we set $\alpha \geq 2$. The value of α can even be reduced by multiplying the potential of each node by $1/\epsilon$. Then the drop in potential is $(\alpha - 1)d/\epsilon$ while the cost is only of d. There is, however, an additional complication in this scheme, which forces us to set $\alpha = 2 + \Theta(\epsilon)$: A node on level i might not only decrease its level because of edge deletions (of edges to nodes on level i or higher), but also if a node on level i moves down to level $i - 1$. Said differently, the drop of $(\alpha - 1)d/\epsilon$ of the degree of a node y on level i might not only be caused by edge deletions, but also

by the level drop of incident nodes. Thus, when the level of a node y decreases, the potential of all its neighbors on a larger level has to increase by $1/\epsilon$ to pay for their future level decrease. Thus the drop of the potential of y by $(\alpha-1)d/\epsilon$ has to "pay" for the increase of the potential of its neighbors, which is in total at most d/ϵ, *and* the cost of the operation, which is d. This is possible if we set $\alpha = 2+\epsilon$.

8 A single-pass dynamic streaming algorithm in $\tilde{O}(n)$-space and $\tilde{O}(1)$ update time

Let $\mathbf{N}^+$ denote the set of all positive integers, and let $G^{(t)} = (V, E^{(t)})$ be the state of the input graph $G = (V, E)$ at time-step $t \in \mathbf{N}^+$. Define $n = |V|$ to be the number of nodes (which do not change across different time-steps) and $m^{(t)} = |E^{(t)}|$ to be the number of edges in this graph at time-step $t \in \mathbf{N}^+$. For simplicity of exposition, we assume that (a) the graph has initially only one edge, i.e., $E^{(0)} = \emptyset$, and (b) the edge-set is always nonempty afterwards, i.e., $E^{(t)} \neq \emptyset$ for all $t \in \mathbf{N}^+$. We further assume that at most one edge insertion/deletion occurs at each time-step, implying that $|E^{(t+1)} \oplus E^{(t)}| \leq 1$ for all $t \in \mathbf{N}^+$. Let $\textsc{Opt}^{(t)}$ denote the density of the densest subgraph in $G^{(t)}$. We will design a dynamic algorithm that maintains a good approximation to the value of $\textsc{Opt}$ for polynomially many time-steps, in $\tilde{O}(n)$ space and $\tilde{O}(1)$ amortized update time. Our main result is summarized in the theorem below.

Theorem 8.1. *Fix some small constant $\epsilon \in (0,1)$, a constant $\lambda > 1$, and let $T = \lceil n^\lambda \rceil$. There is a dynamic algorithm that uses $\tilde{O}(n)$ bits of space and maintains a value $\textsc{Output}^{(t)}$ at each time-step $t \in [T]$. The algorithm is randomized and gives the following two guarantees with high probability:*

- *(a) We have $\textsc{Opt}^{(t)}/(4+O(\epsilon)) \leq \textsc{Output}^{(t)} \leq \textsc{Opt}^{(t)}$ for all $t \in [T]$.*
- *(b) The amortized update time of the algorithm is $\tilde{O}(1)$. In other words, the total amount of computation performed by the algorithm across the time-steps $1, \ldots, T$ is $O(T \cdot \mathrm{poly}\log n)$.*

8.1 Defining some parameter values

For the rest of Section 8, we fix the constants $\epsilon \in (0,1)$ and $\lambda > 1$, and define the following parameters.

$$
\begin{aligned}
T &\leftarrow \lceil n^\lambda \rceil && (15)\\
L &\leftarrow 2 + \lceil \log_{(1+\epsilon)} n \rceil && (16)\\
\alpha &\leftarrow (2 + \Theta(\epsilon)) && (17)\\
c &\leftarrow 1000\lambda && (18)\\
\pi^{(t)} &\leftarrow m^{(t)}/(2\alpha n) && (19)\\
\sigma &\leftarrow 2(1+\epsilon)n && (20)\\
K &\leftarrow 1 + \lceil \log_{(1+\epsilon)}(\sigma \cdot (2\alpha n)) \rceil && (21)\\
d_k^{(t)} &\leftarrow (1+\epsilon)^{k-1} \cdot \pi^{(t)} \quad \text{for all } k \in [K], t \in [T] && (22)\\
s_k^{(t)} &\leftarrow m^{(t)} c \log n / d_k^{(t)} \quad \text{for all } k \in [K] && (23)
\end{aligned}
$$

Claim 8.2. *We have $s_k^{(t)} = s_k = 2\alpha c n \log n/(1+\epsilon)^{k-1}$ for all $t \in [1,T]$ and $k \in [K]$.*

Proof. Follows from equations 19, 22 and 23. □

Claim 8.3. *We have* $K > \lceil \log_{(1+\epsilon)}(\sigma/\pi^{(t)}) \rceil$ *for all* $t \in [T]$.

Proof. Follows from equations 19, 20 21, and the fact that $m^{(t)} \geq 1$ for all $t \in [T]$. □

Claim 8.4. *For all* $t \in [T]$, *we have* $\alpha \cdot \pi^{(t)} < \textsc{Opt}^{(t)} < \sigma/(2(1+\epsilon))$.

Proof. In the graph $G^{(t)}$, the density of the entire set of nodes V is $m^{(t)}/n$. So we have $m^{(t)}/n \leq \textsc{Opt}^{(t)}$. On the other hand, the density of any subset of nodes $V' \subseteq V$ is maximum when the subgraph induced by V' is a complete graph, and in that case the density of V' is $(|V'|-1)/2 < n$. Thus, we get $\textsc{Opt}^{(t)} < n$. So we have: $m^{(t)}/n \leq \textsc{Opt}^{(t)} < n$. Since $\pi^{(t)} = m^{(t)}/(2\alpha n)$ and $\sigma = 2(1+\epsilon)n$, we infer that:

$$\alpha \cdot \pi^{(t)} < \textsc{Opt}^{(t)} < \sigma/(2(1+\epsilon)).$$

□

8.2 The main algorithm: An overview of the proof of Theorem 8.1

We classify each time-step $t \in [T]$ as either *dense* or *sparse*. We do this "on the fly" as per Figure 3. The only thing the procedure in Figure 3 needs is a counter to keep track of the number of edges $m^{(t)}$ at each time-step t. Since $m^{(t)} \leq n^2$, the counter requires $\tilde{O}(1)$ space.

This partitions the range $[1, T]$ into dense and sparse intervals, where a dense (resp. sparse) interval is defined to be a maximal and contiguous block of dense (resp. sparse) time-steps (see Definitions 8.5, 8.6). Our classification scheme ensures that the following properties are satisfied (see Lemma 8.7).

- We have $m^{(t)} \leq 8\alpha c^2 n \log^2 n$ for every sparse time-step $t \in [T]$.
- We have $m^{(t)} \geq 4\alpha c^2 n \log^2 n$ for every dense time-step $t \in [T]$.
- If a dense interval begins at a time-step $t \in [T]$, then we have $m^{(t)} = 1 + 8\alpha c^2 n \log^2 n$.
- Every interval – dense or sparse – spans at least $4\alpha c^2 n \log^2 n$ time-steps, unless it is the interval ending at T.

```
01.  The time-step 1 is classified as sparse.
02.  FOR  t = 2 to T
03.        IF time-step (t − 1) was sparse, THEN
04.              IF m^(t) ≤ 8αc^2 n log^2 n, THEN
05.                    Classify time-step t as sparse.
06.              ELSE IF m^(t) > 8αc^2 n log^2 n, THEN
07.                    Classify time-step t as dense.
08.        ELSE IF time-step (t − 1) was dense, THEN
09.              IF m^(t) ≥ 4αc^2 n log^2 n, THEN
10.                    Classify time-step t as dense.
11.              ELSE IF m^(t) < 4αc^2 n log^2 n, THEN
12.                    Classify time-step t as sparse.
```

Figure 3: CLASSIFY-TIME-STEPS(.).

Definition 8.5. *For $t_0, t_1 \in [T], t_0 \leq t_1$, the interval $[t_0, t_1]$ is called a* sparse time interval *iff (a) time-step t is sparse for all $t \in [t_0, t_1]$, (b) either $t_0 = 1$ or time-step $(t_0 - 1)$ is dense, and (c) either $t_1 = T$ or time-step $(t_1 + 1)$ is dense.*

Definition 8.6. *For $t_0, t_1 \in [T], t_0 \leq t_1$, the interval $[t_0, t_1]$ is called a* dense time interval *iff (a) time-step t is dense for all $t \in [t_0, t_1]$, (b) either $t_0 = 1$ or time-step $(t_0 - 1)$ is sparse, and (c) either $t_1 = T$ or time-step $(t_1 + 1)$ is sparse.*

Lemma 8.7. *The procedure in Figure 3 gives the following guarantees.*

- *Consider any sparse time-interval $[t_0, t_1]$. We have (a) $m^{(t)} \leq 8\alpha c^2 n \log^2 n$ for all $t \in [t_0, t_1]$, and (b) either $t_1 = T$ or $(t_1 - t_0) > 4\alpha c^2 n \log^2 n$.*
- *Consider any dense time-interval $[t_0, t_1]$. We have (a) $m^{(t_0)} = 1 + 8\alpha c^2 n \log^2 n$, (b) $m^{(t)} \geq 4\alpha c^2 n \log^2 n$ for all $t \in [t_0, t_1]$, and (c) either $t_1 = T$ or $(t_1 - t_0) > 4\alpha c^2 n \log^2 n$.*

Proof. (sketch) The lemma holds since the input graph $G = (V, E)$ has only one edge at time $t = 1$, and since one edge insertion/deletion occurs in the graph per time-step. □

We run two algorithms simultaneously: The first one maintains a correct output at every sparse time-step $t \in [T]$, while the latter one maintains a correct output at every dense time-step $t \in [T]$. As both of them require $\tilde{O}(n)$ space and $\tilde{O}(1)$ amortized update time, Theorem 8.1 follows from Theorems 8.8 and 8.9.

Theorem 8.8. *There is a dynamic algorithm that uses $\tilde{O}(n)$ bits of space and maintains a value $\textsc{Output}^{(t)}$ at every time-step $t \in [T]$. The algorithm is randomized and gives the following two guarantees.*

- *With high probability, $\textsc{Opt}^{(t)}/(4 + O(\epsilon)) \leq \textsc{Output}^{(t)} \leq \textsc{Opt}^{(t)}$ at every* sparse time-step *$t \in [T]$.*
- *With high probability, the algorithm performs $O(T \operatorname{poly}\log n)$ units of computation over the entire duration of the time-period $[1, T]$.*

Theorem 8.9. *There is a dynamic algorithm that uses $\tilde{O}(n)$ bits of space and maintains a value $\textsc{Output}^{(t)}$ at every time-step $t \in [T]$. The algorithm is randomized and gives the following two guarantees:*

- *With high probability, $\textsc{Opt}^{(t)}/(4 + O(\epsilon)) \leq \textsc{Output}^{(t)} \leq \textsc{Opt}^{(t)}$ at every* dense time-step *$t \in [T]$.*
- *With high probability, the algorithm performs $O(T \operatorname{poly}\log n)$ units of computation over the entire duration of the time-period $[1, T]$.*

The proof of Theorem 8.8 is given in Section 8.3, while the proof of Theorem 8.9 is given in Section 8.4.

8.3 Algorithm for sparse time-steps: Proof of Theorem 8.8

Our algorithm consists of two major ingredients. First, we run a subroutine as per Lemma 8.10 throughout the entire time-period $[1, T]$. This ensures that with high probability, we can maintain a subset $F^{(t)} \subseteq E^{(t)}$ such that $F^{(t)} = E^{(t)}$ at every sparse time-step $t \in [T]$. The worst case update time and space complexity of this subroutine is $\tilde{O}(1)$ and $\tilde{O}(n)$ respectively.

Second, we run our dynamic algorithm – which we refer to as DYNAMIC-ALGO – from Section 7 on the graph $(V, F^{(t)})$ during every sparse time-interval. Since $F^{(t)} = E^{(t)}$ throughout the duration of such an interval, this allows us to maintain an $\text{OUTPUT}^{(t)} \in \left[\text{OPT}^{(t)}/(4+O(\epsilon)), \text{OPT}^{(t)}\right]$ at every sparse time-step $t \in [T]$. As the graph has $\tilde{O}(n)$ edges at every sparse time-step, the space complexity of DYNAMIC-ALGO is $\tilde{O}(n)$. It remains to analyze the amortized update time of DYNAMIC-ALGO.

Consider any sparse time-interval $[t_0, t_1]$ with $t_1 < T$. Let $C(t_0, t_1)$ denote the amount of computation performed during this interval by the subroutine DYNAMIC-ALGO. Our analysis from Section 7 implies that $C(t_0, t_1) = O(n \text{ poly}\log n + (t_1 - t_0) \text{ poly}\log n)$. Since $t_1 < T$, by Lemma 8.7 we have $(t_1 - t_0) = \Omega(n \text{ poly}\log n)$. This gives us the guarantee that $C(t_0, t_1) = O((t_1 - t_0) \text{ poly}\log n)$.

In contrast, if the sparse time-interval under consideration were the one that ends at T, and if it were the case that $(t_1 - t_0) = o(n \text{ poly}\log n)$, then we would have had $C(t_0, t_1) = O(n \text{ poly}\log n)$.

Let $\left[t_0^i, t_1^i\right]$ denote the i^{th} sparse time-interval, and let C denote the amount of computation performed by DYNAMIC-ALGO during the entire time-period $[1, T]$. Since the sparse time-intervals are mutually disjoint, and since there can be at most one sparse time-interval that ends at T, we get the following guarantee.

$$\begin{aligned} C &= \left(\sum_i O((t_1^i - t_0^i) \text{ poly}\log n)\right) + O(n \text{ poly}\log n) \\ &\leq O(T \text{ poly}\log n) + O(n \text{ poly}\log n) \\ &= O(T \text{ poly}\log n) \end{aligned}$$

The last equality holds as $T = \Theta(n^\lambda)$ and $\lambda \geq 1$ (equation 15). This concludes the proof of Theorem 8.8.

Lemma 8.10. *We can dynamically maintain a set of edges $F^{(t)} \subseteq E^{(t)}$ at each time-step $t \in [T]$ such that:*

1. *With high probability, we have $F^{(t)} = E^{(t)}$ at each sparse time-step $t \in [T]$.*
2. *The algorithm requires $\tilde{O}(n)$ bits of space and $\tilde{O}(1)$ worst case update time.*

8.3.1 Proof of Lemma 8.10

We give a stronger guarantee, by presenting an algorithm for maintaining a set of edges $F \subseteq E$ such that:

- With high probability, we have $F^{(t)} = E^{(t)}$ at each time-step $t \in [T]$ with $m^{(t)} \leq 8\alpha c^2 n \log^2 n$.
- The algorithm requires $\tilde{O}(n)$ bits of space and $\tilde{O}(1)$ worst case update time.

We use two well known results from the literature on streaming and hashing, as described below.

ℓ_0 sampling in the streaming model [21?]. There is an algorithm – which we call STREAMING-SAMPLER – that satisfies the following properties. The input to the STREAMING-SAMPLER is a set of elements $I \subseteq E$. At each time-step, at most one element is inserted into (or deleted from) the set I. Let $I^{(t)}$ denote the state of I at time-step $t \in \mathbf{N}^+$. With high probability, for polynomially many time-steps $t \in [T]$, the STREAMING-SAMPLER maintains an edge $e^{(t)}$ chosen uniformly at random from the set $I^{(t)}$. It requires $\tilde{O}(1)$ bits of space and $\tilde{O}(1)$ worst case update time.

Uniform hashing in constant time and optimal space [30]. Let E^* be the set of all possible unordered pairs of nodes in V. We have $|E^*| = O(n^2)$ and $E^{(t)} \subseteq E^*$ at each $t \in [T]$. Consider any two integers $w, q \geq 1$. We can construct a w-wise independent uniform hash function $h : E^* \to [q]$ using $O(w \text{ poly}(\log w, \log q, \log n))$ space. Given any $e \in E^*$, the hash value $h(e)$ can be evaluated in $O(1)$ time.

Our algorithm. We set $w \leftarrow 8\alpha c^2 n \log^2 n$, $q \leftarrow 8\alpha c^2 n \log^2 n$, and build a w-wise independent hash function $h : E^* \to [q]$ as described above. This requires $\tilde{O}(n)$ space, and the hash value $h(e)$ for any given $e \in E^*$ can be evaluated in $O(1)$ time. For all $t \in [T]$ and $i \in [q]$, let $B_i^{(t)}$ denote the set of edges $e \in E^{(t)}$ with $h(e) = i$. Thus, the edge-set $E^{(t)}$ is partitioned into q random subsets $B_1^{(t)}, \ldots, B_q^{(t)}$. For each $i \in [q]$, we run $r = c^2 \log^2 n$ copies of Streaming-Sampler. For $i \in [q], j \in [r]$, Streaming-Sampler(i, j) maintains a uniformly random sample from the set $B_i^{(t)}$. Furthermore, the algorithms $\{$Streaming-Samplers$(i, j)\}$, $i \in [q], j \in [r]$, use mutually independent random bits. Let $Y^{(t)}$ denote the collection of the random samples maintained by all these Streaming-Samplers. Since we have multiple Streaming-Samplers running on the same set $B_i^{(t)}, i \in [q]$, a given edge can occur multiple times in $Y^{(t)}$. We define $F^{(t)}$ to be the set of all edges in $E^{(t)}$ that appear at least once in $Y^{(t)}$.

Update time of our algorithm. Suppose that an edge e is inserted into (resp. deleted from) the graph $G = (V, E)$ at time-step $t \in [T]$. To handle this edge insertion (resp. deletion), we first compute the value $i = h(e)$ in constant time. Then we insert (resp. delete) the edge to the set $B_i^{(t)}$, and call the subroutines Streaming-Sampler(i, j) for all $j \in [r]$ so that they can accordingly update the random samples maintained by them. Each Streaming-Sampler takes $O(\text{poly} \log n)$ time in the worst case to handle an update. Since $r = O(\log^2 n)$, the total time taken by our algorithm to handle an edge insertion/deletion is $O(r \text{ poly} \log n) = \tilde{O}(1)$.

Space complexity of our algorithm. We need $O(\text{poly} \log n)$ space to implement each Streaming-Sampler(i, j), $i \in [q], j \in [r]$. Since $q = O(n \log^2 n)$ and $r = O(\log^2 n)$, the total space required by all the streaming samplers is $O(qr \text{ poly} \log n) = \tilde{O}(n)$. Next, note that we can construct the hash function h using $\tilde{O}(n)$ space. These observations imply that the total space requirement of our scheme is $\tilde{O}(n)$.

Correctness of our algorithm. It remains to show that with high probability, at each time-step $t \in [T]$ with $m^{(t)} \leq 8\alpha c^2 n \log^2 n$, we have $F^{(t)} = E^{(t)}$.

Fix any time-step $t \in [T]$ with $m^{(t)} \leq 8\alpha c^2 n \log^2 n$. Consider any $i \in [q]$. The probability that any given edge $e \in E^{(t)}$ has $h(e) = i$ is equal to $1/q$. Since $q = 8\alpha c^2 n \log^2 n$, linearity of expectation implies that $E\left[|B_i^{(t)}|\right] = m^{(t)}/q \leq 1$. Since the hash function h is w-wise independent, and since $m^{(t)} \leq w$, we can apply Chernoff bound and infer that $|B_i^{(t)}| \leq c \log n$ with high probability. Now, a union bound over all $i \in [q]$ shows that with high probability, we have $|B_i^{(t)}| \leq c \log n$ for all $i \in [q]$. Let us call this event $\mathcal{E}^{(t)}$.

Condition on the event $\mathcal{E}^{(t)}$. Fix any edge $e \in E^{(t)}$. Let $h(e) = i$, for some $i \in [q]$. We know that $e \in B_i^{(t)}$, that there are at most $c \log n$ edges in $B_i^{(t)}$, and that our algorithm runs $r = c^2 \log^2 n$ many Streaming-Samplers on $B_i^{(t)}$. Each such Streaming-Sampler maintains (independently of others) a uniformly random sample from $B_i^{(t)}$. Consider the event where the edge e is not picked in any of these random samples. This event occurs with probability at most $(1 - 1/(c \log n))^{c^2 \log^2 n} \leq 1/n^c$.

In other words, conditioned on the event $\mathcal{E}^{(t)}$, an edge $e \in E^{(t)}$ appears in $F^{(t)}$ with high probability. Taking a union bound over all $e \in E^{(t)}$, we infer that $F^{(t)} = E^{(t)}$ with high probability,

conditioned on the event $\mathcal{E}^{(t)}$. Next, we recall that the event $\mathcal{E}^{(t)}$ itself occurs with high probability. Thus, we get that the event $F^{(t)} = E^{(t)}$ also occurs with high probability. To conclude the proof, we take a union bound over all time-steps $t \in [T]$ with $m^{(t)} \leq 8\alpha c^2 n \log^2 n$.

8.4 Algorithm for dense time-steps: Proof of Theorem 8.9

From Section 8.1, recall the definitions of $\pi^{(t)}, \sigma, K$ and $d_k^{(t)}$. Claim 8.3 shows that the numbers $d_1^{(t)} \dots d_K^{(t)}$ shatter the interval $[\pi^{(t)}, \sigma]$ in powers of $(1+\epsilon)$. Claim 8.4, Corollary 6.5 imply the following observation.

- Suppose that we want to maintain a $2\alpha(1+\epsilon)^2 = 4 + O(\epsilon)$ approximation to the value of $\textsc{Opt}^{(t)}$ at every dense time-step $t \in [T]$. For this purpose it is sufficient to maintain, at each dense time-step $t \in [T]$, an $(\alpha, d_k^{(t)}, L)$-decomposition of the input graph $G^{(t)}$ for all $k \in [K]$. Specifically, Theorem 8.9 follows from Lemma 8.11.

Lemma 8.11. *Fix any integer $k \in [K]$. There is a dynamic algorithm that uses $\tilde{O}(n)$ bits of space and maintains L subsets of nodes $V = Z_1^{(t)} \supseteq \dots \supseteq Z_L^{(t)}$ at every dense time-step $t \in [T]$. The algorithm is randomized and gives the following two guarantees with high probability.*

- *The tuple $(Z_1^{(t)} \dots Z_L^{(t)})$ is an $(\alpha, d_k^{(t)}, L)$-decomposition of $G^{(t)}$ at every dense time-step $t \in [T]$.*
- *The algorithm performs $O(T \cdot \text{poly}\log n)$ units of computation during the time-period $[1, T]$.*

We devote the rest of Section 8.4 to the proof of Lemma 8.11.

8.4.1 Overview of our algorithm for Lemma 8.11.

We first prove Lemma 8.12. Intuitively, the lemma shows the existence of a procedure that can maintain a random set of edges $S \subseteq E, |S| \leq s_k$, (see Claim 8.2) throughout the duration of the entire time-horizon $[1, T]$. The lemma gives the further guarantee that whenever a time-step $t \in [T]$ is dense, the set $S^{(t)}$ behaves like a subset of $E^{(t)}$ that is chosen uniformly at random out of all subsets of $E^{(t)}$ that are of size s_k. We run $(L-1)$ mutually independent copies of the procedure guaranteed by Lemma 8.12 throughout the duration of the time-horizon $[1, T]$. We let $S_i^{(t)} \subseteq E^{(t)}$ denote the random set of edges maintained by the i^{th} procedure, for $i \in [L-1]$, at time-step $t \in [T]$.

Next, we present another subroutine Dynamic-stream (see Section 8.4.3) that is invoked only during the dense time-intervals. This subroutine can access the edges in the random subsets $\left\{S_i^{(t)}\right\}, i \in [L-1]$, and it maintains L subsets of nodes $V = Z_1^{(t)} \supseteq \dots \supseteq Z_L^{(t)}$ at each dense time-step $t \in [T]$.

Roadmap. The rest of this section is organized as follows.

- In Section 8.4.2, we prove Lemma 8.12.
- In Section 8.4.3, we give an initial high level description of the subroutine Dynamic-stream.
- In Section 8.4.4, we prove two crucial properties of the subroutine Dynamic-stream. Specifically, Claim 8.16 implies that with high probability $(Z_1^{(t)}, \dots, Z_L^{(t)})$ is an $(\alpha, d_k^{(t)}, L)$-decomposition of $G^{(t)}$ at each dense time-step $t \in [T]$. Further, Claim 8.18 will be used to analyze the amortized update time of this subroutine later in Section 8.4.6.
- In Section 8.4.5, we present the relevant data structures along with a detailed implementation of the subroutine Dynamic-stream. Claim 8.21 gives the space complexity of the subroutine.

- In Section 8.4.6, we analyze the total update time of the DYNAMIC-STREAM subroutine during any given dense time-interval.
- In Section 8.4.7, we summarize our findings and conclude the proof of Lemma 8.11.

Lemma 8.12. *There is a dynamic algorithm for maintaining a random subset of edges $S \subseteq E$, $|S| \leq s_k$, that satisfies the following properties.*

1. *For each time-step $t \in [T]$ with $m^{(t)} \geq 4\alpha c^2 n \log^2 n$ and each edge $e \in E^{(t)}$, we have $\Pr[e \in S^{(t)}] \in \left[(1 \pm \epsilon) c \log n / d_k^{(t)}\right]$.*
2. *Fix any time-step $t \in [T]$. For each edge $e \in E^{(t)}$, consider an indicator random variable $X_e^{(t)} \in \{0, 1\}$ that is set to one iff $e \in S^{(t)}$. Then the variables $\{X_e^{(t)}\}, e \in E^{(t)}$, are negatively associated.*
3. *For all $t \in [T]$, we have $|S^{(t+1)} \oplus S^{(t)}| \leq 2$. In other words, every edge insertion/deletion in $G = (V, E)$ leads to at most two edge insertion/deletions in S.*
4. *The algorithm requires $\tilde{O}(n)$ bits of space and $\tilde{O}(1)$ worst case update time.*

8.4.2 Proof of Lemma 8.12

The next theorem follows from a well-known result about ℓ_0-sampling in the streaming model [21**?**].

Theorem 8.13 (ℓ_0-SAMPLER [21**?**])**.** *We can process a dynamic stream of $O(poly\, n)$ updates in any graph $G = (V, E)$ in $O(poly \log n)$ space, and with high probability, at each step we can maintain a uniformly random sample from the set E. The algorithm takes $O(poly \log n)$ time to handle each update in the stream.*

Recall that $s_k = m^{(t)} c \log n / d_k^{(t)} = 2\alpha c n \log n / (1 + \epsilon)^{k-1}$. Let E^* denote the set of all possible ordered pairs of nodes in V. Thus, $E^* \supseteq E^{(t)}$ at each $t \in [1, T]$, and furthermore, we have $|E^*| = O(n^2)$. Using a well know result from the hashing literature [30], we construct a $(2cs_k \log n)$-wise independent uniform hash function $h : E^* \to [s_k]$ in $O(n \text{ poly} \log n)$ space. Using this hash function, we maintain a partition of the edge-set $E^{(t)}$ into s_k mutually disjoint buckets $\{Q_j^{(t)}\}, j \in [s_k]$, where the bucket $Q_j^{(t)}$ consists of those edges $e \in E^{(t)}$ with $h(e) = j$. For each $j \in [s_k]$, we run an independent copy of ℓ_0-SAMPLER, as per Theorem 8.13, that maintains a uniformly random sample from $Q_j^{(t)}$. The set $S^{(t)}$ consists of the collection of outputs of all these ℓ_0-SAMPLERS. Note that (a) for each $e \in E^*$, the hash value $h(e)$ can be evaluated in constant time [30], (b) an edge insertion/deletion affects exactly one of the buckets, and (c) the ℓ_0-SAMPLER of the affected bucket can be updated in $O(\text{poly} \log n)$ time. Thus, we infer that this procedure handles an edge insertion/deletion in the input graph in $O(\text{poly} \log n)$ time, and furthermore, since $s_k = O(n \text{ poly} \log n)$, the procedure can be implemented in $O(n \text{ poly} \log n)$ space.

Fix any dense time-step $t \in [T]$. Since $m^{(t)} \geq 2cs_k \log n$, we can partition (purely as a thought experiment) the edges in $E^{(t)}$ into at most polynomially many groups $\left\{H_{j'}^{(t)}\right\}$, in such a way that the size of each group lies between $cs_k \log n$ and $2cs_k \log n$. Thus, for any $j \in [s_k]$ and any j', we have $|H_{j'}^{(t)} \cap Q_j^{(t)}| \in [c \log n, 2c \log n]$ in expectation. Since the hash function h is $(2cs_k \log n)$-wise independent, applying Chernoff bound we infer that with high probability, the value $|H_{j'}^{(t)} \cap Q_j^{(t)}|$ is very close to its expectation. Applying union bound over all j, j', we infer that with high

probability, the sizes of all the sets $\left\{H_{j'}^{(t)} \cap Q_j^{(t)}\right\}$ are very close to their expected values – let us call this event $\mathcal{R}^{(t)}$. Since $E[|Q_j^{(t)}|] = m^{(t)}/s_k$ and $|Q_j^{(t)}| = \sum_{j'} |Q_j^{(t)} \cap H_{j'}^{(t)}|$, under the event $\mathcal{R}^{(t)}$, we have that $|Q_j^{(t)}|$ is very close to $m^{(t)}/s_k$ for all $j \in [s_k]$. Under the same event $\mathcal{R}^{(t)}$, due to the ℓ_0-SAMPLERs, the probability that a given edge $e \in E^{(t)}$ becomes part of $S^{(t)}$ is very close to $1/|Q_j^{(t)}| = s_k/m^{(t)} = c \log n/d_k^{(t)}$.

Finally, the property of negative association follows from the observations that (a) if two edges are hashed to different buckets, then they are included in $S^{(t)}$ in a mutually independent manner, and (b) if they are hashed to the same bucket, then they are never simultaneously included in $S^{(t)}$.

8.4.3 Description of the subroutine DYNAMIC-STREAM

Notations. Consider any dense time-interval $t \in [t_0, t_1]$. The goal is to maintain L subsets of nodes $V = Z_1^{(t)} \supseteq \cdots \supseteq Z_L^{(t)}$, in such a way that they form a valid $(\alpha, d_k^{(t)}, L)$-decomposition of $G^{(t)} = (V, E^{(t)})$ at each time-step $t \in [t_0, t_1]$. Recall that we run $(L-1)$ mutually independent copies of the procedure stated in Lemma 8.12 throughout the entire duration of the time-period $[1, T]$. Hence, for each $i \in [L-1]$, we can access a random subset of edges $S_i^{(t)} \subseteq E^{(t)}$ as per Lemma 8.12 during the interval $[t_0, t_1]$. Furthermore, at each time-step $t \in [t_0, t_1]$, the random subsets $S_1^{(t)}, \ldots, S_{(L-1)}^{(t)}$ are mutually independent.

Initialization. Just before time-step t_0, we perform the initialization step outlined in Figure 4. It ensures that $Z_1^{(t_0-1)} = V$ and $Z_i^{(t_0-1)} = \emptyset$ for all $i \in \{2, \ldots, L\}$.

01.	Set $Z_1^{(t_0-1)} \leftarrow V$.
02.	FOR $i = 2$ to L
03.	Set $Z_i^{(t_0-1)} \leftarrow \emptyset$.

Figure 4: INITIALIZE(.).

Handling an edge insertion/deletion in the input graph. At time-step $t \in [t_0, t_1]$, an edge insertion/deletion occurs in the input graph. Thus, the edge-set $E^{(t)}$ is different from the edge-set $E^{(t-1)}$. Accordingly, for all $i \in [L-1]$, the subset $S_i^{(t)}$ may differ from the subset $S_i^{(t-1)}$ by at most two edges (see Lemma 8.12). Therefore, at this time we call the subroutine RECOVER-SAMPLE(t) in Figure 5. Its input is the old decomposition $(Z_1^{(t-1)}, \ldots, Z_L^{(t-1)})$. Based on this old decomposition, and the new samples $\left\{S_i^{(t)}\right\}, i \in [L-1]$, the subroutine constructs a new decomposition $(Z_1^{(t)}, \ldots, Z_L^{(t)})$.

Claim 8.14. *Fix any dense time-step $t \in [t_0, t_1]$ and any $i \in [L-1]$. The set $Z_i^{(t)}$ is completely determined by the outcomes of the samples in $\left\{S_j^{(t)}\right\}, j < i$. In other words, the random sets $S_i^{(t)}, \ldots, S_{L-1}^{(t)}$ are chosen independently of the contents of $Z_i^{(t)}$.*

Proof. (sketch) Follows from the description of the procedure in Figure 5. □

8.4.4 Some crucial properties of the subroutine DYNAMIC-STREAM

Definition 8.15. *At every dense time-step $t \in [T]$, let $\mathcal{E}_i^{(t)}$, $i \in [L-1]$, denote the event that:*

01. Set $Z_1^{(t)} \leftarrow V$.
02. FOR $i = 1$ to L
03. Set $Y_i \leftarrow Z_i^{(t-1)}$.
04. FOR $i = 1$ to $(L-1)$
05. Let $A_i^{(t)}$ be the set of nodes $y \in Z_i^{(t)}$ having $D_y(Z_i^{(t)}, S_i^{(t)}) \geq (1-\epsilon)^2 \alpha c \log n$.
06. Let $B_i^{(t)}$ be the set of nodes $y \in Z_i^{(t)}$ having $D_y(Z_i^{(t)}, S_i^{(t)}) \leq (1+\epsilon)^2 c \log n$.
07. Set $Y_{i+1} \leftarrow Y_{i+1} \cup A_i^{(t)}$.
08. FOR ALL $j = (i+1)$ to $(L-1)$
09. Set $Y_j \leftarrow Y_j \setminus B_i^{(t)}$.
10. Set $Z_{i+1}^{(t)} \leftarrow Y_{i+1}$.

Figure 5: RECOVER-SAMPLE(t). The detailed implementation of this subroutine appears in Section 8.4.5.

- *(1)* $\left\{v \in Z_i^{(t)} : D_v(Z_i^{(t)}, E^{(t)}) > \alpha d_k^{(t)}\right\} \subseteq Z_{i+1}^{(t)}$, and
- *(2)* $Z_{i+1}^{(t)} \bigcap \left\{v \in Z_i^{(t)} : D_v(Z_i^{(t)}, E^{(t)}) < d_k^{(t)}\right\} = \emptyset$.

Define $\mathcal{E}^{(t)} = \bigcap_{i \in [L-1]} \mathcal{E}_i^{(t)}$. *Note that under the event* $\mathcal{E}^{(t)}$, *the tuple* $(V_1^{(t)}, \ldots, V_L^{(t)})$ *is a valid* $(\alpha, d_k^{(t)}, L)$*-decomposition of the input graph* $G^{(t)}$.

Claim 8.16. *At every dense time-step* $t \in [T]$, *we have* $\Pr[\mathcal{E}^{(t)}] \geq 1 - 1/(\text{poly } n)$. *In other words, with high probability, the tuple* $(Z_1^{(t)}, \ldots, Z_L^{(t)})$ *is an* $(\alpha, d_k^{(t)}, L)$*-decomposition of the input graph* $G^{(t)}$.

Proof. Consider any dense time-interval $[t_0, t_1]$. Recall the description of the subroutine DYNAMIC-STREAM from Section 8.4.3. For the rest of the proof, fix any time step $t \in [t_0, t_1]$, any $i \in [L-1]$, and condition on the outcomes of the samples $\left\{S_j^{(t)}\right\}, j < i$. By Claim 8.14, the outcomes of these samples determines the contents of the set $Z_i^{(t)}$. This, in turn, determines the sets $\left\{\mathcal{N}_u(Z_i^{(t)}, E^{(t)})\right\}, u \in V$. Also by Claim 8.14, the random set $S_i^{(t)}$ is chosen independently of $Z_i^{(t)}$. Thus, we get:

For every node $u \in V$, the random set $S_i^{(t)} \subseteq E^{(t)}$ is chosen independently of $\mathcal{N}_u(Z_i^{(t)}, E^{(t)})$. (24)

Suppose we are able to show that $\Pr\left[\mathcal{E}_i^{(t)} \mid Z_i^{(t)}\right] \geq 1 - 1/(\text{poly } n)$, and this holds for every possible instantiation of $Z_i^{(t)}$. This will imply that $\Pr\left[\mathcal{E}_i^{(t)}\right] \geq 1 - 1/(\text{poly } n)$. Taking a union bound over all $i \in [L-1]$, we will get $\Pr\left[\mathcal{E}^{(t)}\right] \geq 1 - 1/(\text{poly } n)$, concluding the proof of the claim. Motivated by this reasoning, for the rest of the proof we focus on showing that $\Pr\left[\mathcal{E}_i^{(t)} \mid Z_i^{(t)}\right] \geq 1 - 1/(\text{poly } n)$.

For all $e \in E^{(t)}$, let $X_e^{(t)} \in \{0, 1\}$ be an indicator random variable for the event $e \in S_i^{(t)}$. Thus, we get:

$$D_u(Z_i^{(t)}, S_i^{(t)}) = \sum_{e=(u,v) \,:\, v \in \mathcal{N}_u(Z_i^{(t)}, E^{(t)})} X_e^{(t)} \quad \text{for each node } u \in V. \tag{25}$$

Applying Lemma 8.12, we get the following guarantee.

$$\text{The variables } \{X_e^{(t)}\}, e \in E^{(t)}, \text{ are negatively associated, and } E[X_e^{(t)}] \in \left[(1 \pm \epsilon)c \log n / d_k^{(t)}\right]. \tag{26}$$

From equations (24), (25), (26), and linearity of expectation, we get:

$$E\left[D_u(Z_i^{(t)}, S_i^{(t)}) \mid Z_i^{(t)}\right] \in \left[(1 \pm \epsilon)c \log n / d_k^{(t)} \cdot D_u(Z_i^{(t)}, E^{(t)})\right] \quad \text{for each node } u \in V. \tag{27}$$

The procedure in Figure 5 ensures that for all node $u \in Z_i^{(t)}$, we have:

$$\Pr\left[u \notin Z_{i+1}^{(t)} \mid Z_i^{(t)}\right] \leq \Pr\left[D_u(Z_i^{(t)}, S_i^{(t)}) \leq (1-\epsilon)^2 \alpha c \log n \mid Z_i^{(t)}\right].$$

Accordingly, from equations (24) - (27), and Theorem 6.9 (Chernoff bound), we get:

$$\Pr\left[u \notin Z_{i+1}^{(t)} \mid Z_i^{(t)}\right] \leq 1/(\text{poly } n) \quad \text{for all } u \in Z_i^{(t)} \text{ with } D_u(Z_i^{(t)}, E^{(t)}) > \alpha d_k^{(t)}. \tag{28}$$

The procedure in Figure 5 ensures that for all node $u \in Z_i^{(t)}$, we have:

$$\Pr\left[u \in Z_{i+1}^{(t)} \mid Z_i^{(t)}\right] \leq \Pr\left[D_u(Z_i^{(t)}, S_i^{(t)}) \geq (1+\epsilon)^2 c \log n \mid Z_i^{(t)}\right].$$

Accordingly, from equations (24) - (27), and Theorem 6.9 (Chernoff bound), we get:

$$\Pr\left[u \in Z_{i+1}^{(t)} \mid Z_i^{(t)}\right] \leq 1/(\text{poly } n) \quad \text{for all } u \in Z_i^{(t)} \text{ with } D_u(Z_i^{(t)}, E^{(t)}) < d_k^{(t)}. \tag{29}$$

We now apply union bound over the events described in equations (28), (29), for all $u \in Z_i^{(t)}$, and get:

$$\Pr\left[\mathcal{E}_i^{(t)} \mid Z_i^{(t)}\right] \geq 1 - 1/(\text{poly } n).$$

□

Definition 8.17. *For all $i, i' \in [L-1]$, $i < i'$, let $\mathcal{F}_{i,i'}^{(t)}$ be the event that: $\left\{D_v(Z_i^{(t)}, S_{i'}^{(t)}) \geq \frac{(1-\epsilon)^4}{(1+\epsilon)^2} \cdot (\alpha c \log n)\right.$ for all $v \in A_i^{(t)}\}$, and $\left\{D_v(Z_i^{(t)}, S_{i'}^{(t)}) \leq \frac{(1+\epsilon)^4}{(1-\epsilon)^2} \cdot c \log n\right.$ for all $v \in B_i^{(t)}\}$. Define $\mathcal{F}^{(t)} = \bigcap_{i,i'} \mathcal{F}_{i,i'}^{(t)}$.*

Claim 8.18. *With high probability, the event $\mathcal{F}^{(t)}$ holds at each dense time-step $t \in [T]$.*

Proof. (sketch) We use an argument similar to the proof of Claim 8.16.

Fix any $1 \leq i < i' \leq L-1$, any dense time-step $t \in [T]$, and condition on any instantiation of the random set $Z_i^{(t)}$. By Claim 8.14, the random sets $S_i^{(t)}, S_{i'}^{(t)}$ are chosen independently of $Z_i^{(t)}$. Further, for all $v \in Z_i^{(t)}$, we have $E[D_v(Z_i^{(t)}, S_i^{(t)})] = E[D_v(Z_i^{(t)}, S_{i'}^{(t)})] \in \left[(1 \pm \epsilon) \cdot (c \log n / d_k^{(t)}) \cdot D_v(Z_i^{(t)}, E^{(t)})\right]$, and by Lemma 8.12 we can apply Chernoff bound on this expectation. Thus, applying union bounds over $\{i, i'\}$, we infer that w.h.p. the following condition holds: If $D_v(Z_i^{(t)}, E^{(t)})$ is sufficiently smaller (resp. larger) than $d_k^{(t)}$, then both $D_v(Z_i^{(t)}, S_i^{(t)})$ and $D_v(Z_i^{(t)}, S_{i'}^{(t)})$ are sufficiently smaller (resp. larger) than $c \log n$. The proof follows by deriving a variant of this claim and then applying union bounds over all i, i' and t. □

8.4.5 Implementing the subroutine DYNAMIC-STREAM

Consider any dense time-interval $[t_0, t_1]$. We will show how to implement the procedure DYNAMIC-STREAM described in Section 8.4.3. For all $j \in [L-1]$, define the subset $V_j \leftarrow Z_j \setminus Z_{j+1}$, and let $V_L \leftarrow Z_L$. The set of nodes in V_j form the j^{th} level of this decomposition. We let $\ell(v) \in [L]$ denote the level of a node $v \in V$, so that we have $\ell(v) = j$ for all $v \in V_j$.

Data structures. We use the following data structures.

- For every node $v \in V$, we maintain a counter $\textsc{Level}[v]$ that keeps track of $\ell(v)$.
- For every $i \in [L-1]$, we maintain a set of doubly linked lists $\textsc{Friends}_i[.\,,.]$ for the edges in $S_i \subseteq E$.
  - For $i' \in [L-1]$ and every $v \in V_{i'}$, maintain the following lists $\{\textsc{Friends}_i[v,j]\}, j \in [L-1]$.
    - If $i \leq i'$, then $\textsc{Friends}_i[v,i] = \mathcal{N}_v(Z_i, S_i)$ and for all $j < i$, $\textsc{Friends}_i[v,j] = \mathcal{N}_v(V_j, S_i)$.
    - If $i > i'$, then $\textsc{Friends}_i[v,i'] = \mathcal{N}_v(Z_{i'}, S_i)$ and for all $j < i'$, $\textsc{Friends}_i[v,j] = \mathcal{N}_v(V_j, S_i)$.
    - For all $j > \min(i, i')$, the list $\textsc{Friends}_i[v,j]$ is empty.
- For every node $v \in V$, we maintain a counter $\textsc{Degree}_i[v]$. If $\ell(v) \geq i$, then this counter is set to $D_v(Z_i, S_i)$. Else if $\ell(v) < i$, then this counter is set to zero.
- We maintain a doubly linked list $\textsc{Dirty-Nodes}[i]$. This consists of all the nodes $v \in Z_i$ having either $\left\{\textsc{Degree}_i[v] \geq (1-\epsilon)^2 \alpha c \log n \text{ and } \ell(v) = i\right\}$ or $\left\{\textsc{Degree}_i[v] \leq (1+\epsilon)^2 c \log n \text{ and } \ell(v) > i\right\}$.
- Using appropriate pointers, we can decide in $O(1)$ time whether a given node is part of a given linked list, and if yes (resp. no), then we can insert it into (resp. delete it from) the given list in $O(1)$ time.

Initialization. Just before the dense time-interval begins, we have to implement a call to the subroutine in Figure 4. This assigns all the nodes to level 1. Since there are $\tilde{O}(n)$ edges in $\bigcup_{i \in [L-1]} S_i$, initializing the relevant data structures can be done in $\tilde{O}(n)$ time.

Handling the insertion/deletion of an edge in the input graph. Suppose that an edge insertion/deletion occurs in $G = (V, E)$ at time-step $t \in [t_0, t_1]$. We first update each of the subsets $S_1, \ldots, S_{(L-1)}$ as per Lemma 8.12. Overall, this takes $O(L \operatorname{poly}\log n) = \tilde{O}(1)$ worst case time. Furthermore, this leads to at most 2 edge insertion/deletion in each S_i, $i \in [L-1]$. We need to accordingly update our data structures to make them consistent with the new samples $\left\{S_i^{(t)}\right\}, i \in [L-1]$. Since we are dealing with only $2L = O(\operatorname{poly}\log n)$ many changes, this takes worst case $\tilde{O}(1)$ time.

Finally, we implement RECOVER-SAMPLE(t) (see Figure 5) by calling the subroutine stated in Figure 6. The following two claims will be crucial while analyzing the amortized update time of this scheme.

Claim 8.19. *Suppose that a node y is promoted to level $(i+1)$ from level i during the execution of the procedure in Figure 6 (see Step 05). Then updating the relevant data structures to reflect this change requires $\sum_{i'=i+1}^{(L-1)} O(1 + D_y(Z_i, S_{i'}))$ units of computation.*

Proof. (sketch) We do not need to update the $\textsc{Friends}_{i'}[.\,,.]$ lists for $i' \leq i$. In addition, while updating a list $\textsc{Friends}_{i'}[.\,,.]$ with $i' > i$, we only need to look at the nodes in $\{v\} \cup \mathcal{N}_v(Z_i, S_{i'})$. □

Claim 8.20. *Suppose that a node y is demoted to level i during the execution of the procedure in Figure 6 (see Step 07). Then updating the relevant data structures to reflect this change requires $\sum_{i'=i+1}^{(L-1)} O(1 + D_y(Z_i, S_{i'}))$ units of computation.*

Proof. (sketch) Similar to the proof of Claim 8.19. □

Claim 8.21. *The subroutine* Dynamic-stream *can be implemented in $\tilde{O}(n)$ space.*

Proof. (sketch) The amount of space needed is dominated by the number of random samples in $\left\{S_i^{(t)}\right\}, i \in [L-1]$. Since $|S_i^{(t)}| \leq s_k$ for each $i \in [L-1]$, the space complexity is $(L-1) \cdot s_k = O(n \text{ poly} \log n)$. □

```
01.  For i = 1 to (L − 1)
02.      While the list Dirty-Nodes[i] is nonempty:
03.          Let y be any node in the list Dirty-Nodes[i].
04.          If Level[y] = i and Degree_i[y] ≥ (1 − ε)^2 αc log n, Then
05.              Set Level[y] ← Level[y] + 1 and update the relevant data structures.
06.          Else if Level[y] > i and Degree_i[y] ≤ (1 + ε)^2 c log n, Then
07.              Set Level[y] ← i and update the relevant data structures.
```

Figure 6: Implementing the subroutine RECOVER-SAMPLE(t).

8.4.6 Analyzing the amortized update time of the subroutine Dynamic-stream

Claim 8.22. *Consider our implementation of the* Dynamic-stream *subroutine in Section 8.4.5 applied to any dense time-interval $[t_0, t_1]$. We have the following guarantee.*

- *With high probability, the total amount of computation performed by the subroutine* Dynamic-stream *over all time-steps $t \in [t_0, t_1]$ is at most $O(n \; poly \log n + (t_1 - t_0) \; poly \log n)$.*

Fix any dense time-interval $[t_0, t_1]$. We devote the rest of this section to the proof of Claim 8.22.

Potential function. To determine the amortized update time we use a potential function $\mathcal{B}$ as defined in equation 33. Note that the potential $\mathcal{B}$ is uniquely determined by the assignment of the nodes $v \in V$ to the levels $[L]$ and by the content of the random sets $S_1, \ldots, S_{(L-1)}$. For all nodes $v \in V$, we define:

$$\Gamma_i(v) \;=\; \max(0, (1-\epsilon)^2 \alpha c \log n - D_v(Z_i, S_i)) \tag{30}$$

$$\Phi(v) \;=\; (L/\epsilon) \cdot \sum_{i=1}^{\ell(v)-1} \Gamma_i(v) \tag{31}$$

For all $u, v \in V$, let $f(u,v) = 1$ if $\ell(u) = \ell(v)$ and 0 otherwise. For all $i \in [L-1]$, $(u,v) \in S_i$, we define:

$$\Psi_i(u,v) = \begin{cases} 0 & \text{if } \min(\ell(u), \ell(v)) \geq i; \\ 2 \cdot (i - \min(\ell(u), \ell(v))) + f(u,v) & \text{otherwise.} \end{cases} \tag{32}$$

$$\mathcal{B} = \sum_{v \in V} \Phi(v) + \sum_{i=1}^{(L-1)} \sum_{e \in S_i} \Psi_i(e) \tag{33}$$

Claim 8.23. *Conditioned on the event $\bigcap_{t=t_0}^{t_1} \mathcal{F}^{(t)}$ (see Definition 8.17), we have:*

- *(a) $0 \leq \mathcal{B} = \tilde{O}(n)$ at each $t \in [t_0, t_1]$.*
- *(b) Insertion/deletion of an edge in G (ignoring the call to Figure 5) changes the potential $\mathcal{B}$ by $\tilde{O}(1)$.*
- *(c) For every constant amount of computation performed while implementing Figure 5, the potential $\mathcal{B}$ drops by $\Omega(1)$.*

Claim 8.18 guarantees that the event $\bigcap_{t=t_0}^{t_1} \mathcal{F}^{(t)}$ holds with high probability. Thus, Claim 8.23 implies Claim 8.22. For the rest of this section, we focus on proving Claim 8.23.

Proof of part (a). Follows from three facts.

1. We have $0 \leq \Phi(v) \leq (L/\epsilon) \cdot L \cdot (1-\epsilon)^2 \alpha c \log n = O(\text{poly} \log n)$ for all $v \in V$.
2. We have $0 \leq \Psi_i(u,v) \leq 3L = O(\text{poly} \log n)$ for all $i \in [L-1], (u,v) \in S_i$.
3. We have $|S_i| \leq s_k = O(n \text{ poly} \log n)$ for all $i \in [L-1]$.

Proof of part (b). By Lemma 8.12, insertion/deletion of an edge in G leads to at most two insertions/deletions in the random set S_i, for all $i \in [L-1]$. As $L = O(\text{poly} \log n)$, it suffices to show that for every edge insertion/deletion in any given S_i, the potential $\mathcal{B}$ changes by at most $O(\text{poly} \log n)$ (ignoring call to Figure 5).

Towards this end, fix any $i \in [L-1]$, and suppose that a single edge (u,v) is inserted into (resp. deleted from) S_i. For each node $v \in V$, this changes the potential $\Phi(v)$ by at most $O(L/\epsilon)$. And the potential $\Psi_i(u,v) \in [0, 3L]$ is created (resp. destroyed). Summing over all the nodes $v \in V$, we infer that the absolute value of the change in the overall potential $\mathcal{B}$ is at most $O(3L + nL/\epsilon) = O(n \text{ poly} \log n)$.

Proof of part (c). Focus on a single iteration of the For loop in Figure 5, and consider two possible cases.

Case 1: A node $v \in Z_i^{(t)}$ is promoted from level i to level $(i+1)$ in Step 07 of Figure 5.

This can happen only if $v \in A_i^{(t)}$. Let C denote the amount of computation performed during this step.

$$\text{By Claim 8.19, we have: } C = \sum_{i'=(i+1)}^{(L-1)} O\left(1 + D_v(Z_i^{(t)}, S_{i'}^{(t)})\right) \tag{34}$$

Let Δ be the net decrease in the overall potential $\mathcal{B}$ due to this step. We make the following observations.

1. Consider any $i' > i$. For each edge $(u,v) \in S_{i'}^{(t)}$ with $u \in Z_i^{(t)}$, the potential $\Psi_{i'}(u,v)$ decreases by at least one. For every other edge $e \in S_{i'}^{(t)}$, the potential $\Psi_{i'}(e)$ remains unchanged.
2. For each $i' \in [i]$ and each edge $e \in S_{i'}^{(t)}$, the potential $\Psi_{i'}(e)$ remains unchanged.

3. Since the node v is being promoted to level $(i+1)$, we must have $D_v(Z_i^{(t)}, S_i^{(t)}) \geq (1-\epsilon)^2 \alpha c \log n$. Thus, the potential $\Phi(v)$ remains unchanged.

4. Finally, for each node $u \in V \setminus \{v\}$, the potential $\Phi(u)$ can only decrease.

Taking into account all these observations, we infer the following inequality.

$$\Delta \geq \sum_{i'=(i+1)}^{(L-1)} D_v(Z_i^{(t)}, S_{i'}^{(t)}) \tag{35}$$

Since $v \in A_i^{(t)}$, and since we have conditioned on the event $\mathcal{F}^{(t)}$ (see Definition 8.17), we get:

$$D_v(Z_i^{(t)}, S_{i'}^{(t)}) > 0 \quad \text{for all } i' \in [i+1, L-1]. \tag{36}$$

Eq. (34), (35), (36) imply that the decrease in $\mathcal{B}$ is sufficient to pay for the computation performed.

Case 2: A node $v \in Z_i^{(t)}$ is demoted from level $j > i$ to level i in Steps (08-09) of Figure 5.

This can happen only if $v \in B_i^{(t)}$. Let C denote the amount of computation performed during this step.

$$\text{By Claim 8.19, we have: } C = \sum_{i'=(i+1)}^{(L-1)} O(1 + D_v(Z_i^{(t)}, S_{i'}^{(t)})) \tag{37}$$

Let $\gamma = (1+\epsilon)^4/(1-\epsilon)^2$. Equation 38 holds since $v \in B_i^{(t)}$ and since we conditioned on the event $\mathcal{F}^{(t)}$. Equation 39 follows from equations 37, 38, and the facts that γ, c are constants,

$$D_v(Z_i^{(t)}, S_{i'}^{(t)}) \leq \gamma c \log n \text{ for all } i' \in [i, L-1] \tag{38}$$

$$C = O(L \log n) \tag{39}$$

Let Δ be the net decrease in the overall potential $\mathcal{B}$ due to this step. We make the following observations.

1. By eq. (38), the potential $\Phi(v)$ decreases by at least $(j-i) \cdot (L/\epsilon) \cdot ((1-\epsilon)^2\alpha - \gamma) \cdot (c \log n)$.

2. If either $u \in V \setminus \{v\}$ or $i' \in [j+1, L-1] \cup [1, i]$, then the potential $\Gamma_{i'}(u)$ remains unchanged. This observation, along with equation (38), implies that the sum $\sum_{u \neq v} \Phi(u)$ increases by at most $(L/\epsilon) \cdot \sum_{i'=(i+1)}^{j} D_v(Z_i^{(t)}, S_{i'}^{(t)}) \leq (j-i) \cdot (L/\epsilon) \cdot (\gamma c \log n)$.

3. For every $i' \in [1, i]$, $e \in S_{i'}^{(t)}$ the potential $\Psi_{i'}(e)$ remains unchanged. Next, consider any $i' \in [i+1, L-1]$. For each edge $(u, v) \in S_{i'}^{(t)}$ with $u \in Z_i^{(t)}$, the potential $\Psi_{i'}(u, v)$ increases by at most $3(j-i)$. For every other edge $e \in S_{i'}^{(t)}$, the potential $\Psi_{i'}(e)$ remains unchanged. These observations, along with equation (38), imply that the sum $\sum_{i'} \sum_{e \in S_{i'}} \Psi_{i'}(e)$ increases by at most $\sum_{i'=(i+1)}^{(L-1)} 3(j-i) \cdot D_v(Z_i^{(t)}, S_{i'}^{(t)}) \leq (j-i) \cdot (3L) \cdot (\gamma c \log n)$.

Taking into account all these observations, we get:

$$\begin{aligned}\Delta &\geq (j-i)(L/\epsilon)((1-\epsilon)^2\alpha-\gamma)(c\log n)-(j-i)(L/\epsilon)(\gamma c\log n)-(j-i)(3L)(\gamma c\log n)\\ &= (j-i)\cdot(L/\epsilon)\cdot((1-\epsilon)^2\alpha-2\gamma-3\epsilon\gamma)\cdot(c\log n)\\ &\geq Lc\log n \qquad (40)\end{aligned}$$

The last inequality holds since $(j-i)\geq 1$ and $\alpha\geq(\epsilon+(2+3\epsilon)\gamma)/(1-\epsilon)^2$. From eq. (39) and (40), we conclude that the net decrease in the overall potential $\mathcal{B}$ is sufficient to pay for the computation performed.

8.4.7 Concluding the proof of Lemma 8.11

Lemma 8.12 ensures that we can maintain the random sets $\{S_i\}, i\in[L-1]$ in $\tilde{O}(n)$ space and $\tilde{O}(1)$ worst case update time. It remains to bound the update time and space complexity of the Dynamic-Stream subroutine described in Section 8.4.3. Claim 8.21 states that this subroutine requires $\tilde{O}(n)$ bits of space. Claim 8.22, combined with the same argument that was used in the first parts of Section 8.3, implies that with high probability the Dynamic-stream subroutine performes $O(T\ \mathrm{poly}\log n)$ amount of computation during the entire time-horizon $[1,T]$. In other words, the amortized update time of this subroutine is $\tilde{O}(1)$ with high probability. Finally, Claim 8.16 states that with high probability the subroutine maintains a valid $(\alpha, d_k^{(t)}, L)$-decomposition of the input graph $G^{(t)}$ at every dense time-step $t\in[T]$. This concludes the proof of Lemma 8.11.

Part III
APPENDIX

A Sublinear-Time Algorithm

Model. Whether we can achieve a sublinear-time algorithm for a problem depends highly on the representation of the input graph. There are many representations (see, e.g. [20, 31]). In this paper we focus on a standard representation called *incidence list*. (This is the representation assumed in, e.g., [10, 17].) In this representation, we allow two types of input access (called *query*): (1) the *degree query* which asks for the degree of some node v, and (2) the *neighbor query* which asks for the i^{th} neighbor of v (i.e. the i^{th} element in the incidence list corresponding to the neighbors of v). See, e.g., [12, 19, 20, 29, 31, 32] for further surveys.

Upper Bound. In the paper, we show that we can compute a $(2+\epsilon)$-approximate solution to the densest subgraph problem from a sketch constructed by sampling $\tilde{O}(n)$ edges uniformly at random. In the incidence list model, sampling an edge can be done easily (using one query). Thus, we needs only $\tilde{O}(n)$ queries. Moreover, we can process this sketch using $\tilde{O}(n)$ time and space, simply by computing the $(1+\epsilon, d, \tilde{O}(1))$-decompositions for different $\tilde{O}(1)$ values of d, to get the desired $(2+\epsilon)$-approximate solution.

Lower Bound. We adapt the proof of [6, Lemma 7] to show that for any $\alpha \geq 2$, an α-approximation algorithm needs to make $\Omega(n/\alpha^2 \operatorname{poly}\log(n))$ queries. Consider the following communication complexity problem: There are $k \geq 2$ players, denoted by $p_1, \ldots, p_k$ and an n-node input graph G consisting of ℓ disjoint subgraphs, denoted by $G_1, \ldots, G_\ell$. Each G_i has k nodes, denoted by $\{u_{i,1}, \ldots, u_{i,k}\}$ (thus $n = k\ell$). Further each subgraph is either a star or a clique. For any node $u_{i,j}$ in G_i, if its degree is more than one then player p_j knows about all edges incident to $u_{i,j}$. In other words, p_j knows about edges incident to nodes with degree more than one among $u_{1,j}, u_{2,j}, \ldots, u_{\ell,j}$. The players want to distinguish between the case where there is a clique (thus the densest subgraph has density at least $(k-1)/2$) and when there is no clique (thus the densest subgraph has density at most 1). Their communication protocol is in the blackboard model, where in each round a player can write a message on the backboard, which will be seen by all other players. By a reduction from the multi-party set disjointness problem [6, 8] showed that this problem require $\tilde{\Omega}(n/k)$ communication bits.

Lemma A.1. *If there is a sublinear-time algorithm with q queries for the above problem, then this problem can be solved using $\tilde{O}(q)$ communication bits.*

Proof. Let $\mathcal{A}$ be such algorithm. Player p_1 simulates $\mathcal{A}$ by answering each query of $\mathcal{A}$ using $\tilde{O}(1)$ communication bits, as follows. If $\mathcal{A}$ makes a degree query on node u_{ij}, player p_1 will ask for an answer from player p_j: either p_j knows all edges incident to u_{ij} (in which case the degree of u_{ij} is k) or the degree of u_{ij} is one. If $\mathcal{A}$ asks for the t^{th} neighbor of node u_{ij}, player p_1 asks for this from player p_j. If player p_j does not know the answer, then we know that the degree of u_{ij} is one and G_i is a star. In this case, player p_1 writes on a blackboard asking for the unique node $u_{ij'}$ in G_i whose degree is more than one. Then, the only edge incident to u_{ij} is $u_{ij}u_{ij'}$. This edge can be used to answer the query. □

The above lemma implies that any $((k-1)/2-\epsilon)$-approximation algorithm requires $\tilde{\Omega}(n/k^2)$ queries. In other words, any α-approximate algorithm requires $\tilde{\Omega}(n/\alpha^2)$ queries.

B Distributed Streams

In the distributed streaming setting (see, e.g., [11]), there might be k *sites* receiving different sequences of edge insertions (without any deletion), and these sites must coordinate with the

coordinator. The objective is to minimize the communication between the sites and the coordinator in order to maintain the densest subgraph. We can sample $\tilde{O}(n)$ edges (without replacement) as a sketch by using the sampling algorithm of Cormode et al. [11] which can sample $\tilde{O}(n)$ edges using $\tilde{O}(k+n)$ communication while the coordinator needs $\tilde{O}(n)$ space and each site needs $O(1)$ space. The coordinator can then use this sketch to compute a $(2+\epsilon)$-approximate solution.