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 $G = (V, E)$, the "density" of a subgraph induced by a subset of nodes $S \subseteq V$ is defined as $|E(S)|/|S|$, where $E(S)$ denotes the set of edges in E with both endpoints in S. In the densest subgraph problem, the goal is to find a subset of nodes that maximizes the density of the corresponding induced subgraph.

For any $\epsilon > 0$, we present a dynamic algorithm that, with high probability, maintains a $(4+\epsilon)$-approximate solution for the densest subgraph problem under a sequence of edge insertions and deletions in an input graph with n nodes. The algorithm uses $\tilde{O}(n)$ space, and has an amortized update time of $\tilde{O}(1)$ and a query time of $\tilde{O}(1)$. Here, $\tilde{O}$ hides a $O(\text{poly} \log_{1+\epsilon} n)$ term. The approximation ratio can be improved to $(2 + \epsilon)$ at the cost of increasing the query time to $\tilde{O}(n)$. 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 an 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.

i

Contents

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. [17, 35]) 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 queries in 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, 26, 27]). 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. [28][1].

1.1 Problem definition

In this paper, we study the *densest subgraph* problem in dynamic and streaming setting. Fix any unweighted undirected input graph $G = (V, E)$. The density of a subgraph induced by the set of nodes $H \subseteq V$ is defined as $\rho(H) = |E(H)|/|H|$, where $E(H) = \{(u, v) \in E : u, v \in H\}$ is the set of edges in the induced subgraph. The densest subgraph of G is the subgraph induced by a node set $H \subseteq V$ that maximizes $\rho(H)$, and we denote the density of such a subgraph by $\rho^*(G) = \max_{H \subseteq V} \rho(H)$. For any $\gamma \geq 1$ and η, we say that η is an γ-*approximate value* of $\rho^*(G)$ if $\rho^*(G)/\gamma \leq \eta \leq \rho^*(G)$. The (static) densest subgraph problem is to compute or approximate $\rho^*(G)$ and the corresponding subgraph. Throughout the paper, we use $n = |V|$ and $m = |E|$ 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. [18]) and finding stories and events (e.g. [4]); for many more applications of this problem see, e.g., [6, 31, 42, 43]. Goldberg [20] was one of the first to study this problem although the notion of graph density has been around much earlier (e.g. [30, 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, and the earlier algorithms cannot handle edge insertions/deletions in the input graph. Consider, for example, an application of detecting a dense community in social networks.

---

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

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 a dynamic version of this problem as defined below.

Our Model. We start with an empty graph $G = (V, E)$ where $E = \emptyset$. Subsequently, at each time-step, an adversary either inserts an edge into the graph, or deletes an already existing edge from the graph. The set of nodes, on the other hand, remain unchanged. The goal is to maintain a good approximation to the value of the densest subgraph while processing this sequence of edge insertions/deletions. More formally, we want to design a data structure for the input graph $G = (V, E)$ that supports the following operations.

- INITIALIZE(V): Initialize the data structure with an empty graph $G = (V, E)$ where $E = \emptyset$.

- INSERT(u, v): Insert the edge (u, v), where $u, v \in V$, into the graph G.

- DELETE(u, v): Delete an existing edge $(u, v) \in E$ from the graph G.

- QUERYVALUE: Return an estimate of the value of the maximum density $\rho^*(G) = \max_{S \subseteq V} \rho(S)$. If this estimate is always within a γ-factor of $\rho^*(G)$, for some $\gamma \geq 1$, then we say that the algorithm maintains a γ-approximation to the value of the densest subgraph. We want this approximation factor to be a small constant.

The performance of a data structure is measured in term of four different metrics, as defined below.

- *Space-complexity:* This is given by the total space (in terms of bits) used by the data structure.

- *Update-time:* This is the time taken to handle an INSERT or DELETE operation.

- *Query-time:* This is the time taken to handle a QUERYVALUE operation.

- *Preprocessing-time:* This is the time taken to handle the INITIALIZE operation. Unless explicitly mentioned otherwise, in this paper the preprocessing time will always be $\tilde{O}(n)$.

Comparison with the semi-streaming model. In the streaming algorithms literature, the *"semi-streaming model"* for graph problems is defined as follows. We start with an empty graph of n nodes. Subsequently, we have to process a *"stream"* of updates in the graph. For *"insert-only"* streams, each update consist of inserting a new edge into the graph. For *"dynamic"* (or, "turnstile") streams, each update consists of either inserting a new edge into the graph or deleting an already existing edge from the graph.

A *"semi-streaming algorithm"* can use only $\tilde{O}(n)$ bits of space while processing a stream of updates. In particular, the algorithm cannot store all the edges in the graph (which might require $\Omega(n^2)$ space). At the end of the stream, the algorithm has to output an (approximate) solution to the problem concerned, which, in our case, happens to be the value of the densest subgraph. The algorithm is allowed to make "multiple passes" over this stream. Typically, in the streaming algorithms literature the focus is on the space complexity, and optimizing the update-time and the query-time (which can be as large as $\Omega(n)$) are of secondary importance.

Our goal. We want to design algorithms that maintain constant factor approximations to the value of the densest subgraph in a dynamic setting, have very fast (polylogarithmic in n) update and query times, and use very little (near-linear in n) space. In other words, we want single-pass semi-streaming algorithms over dynamic streams with polylogarithmic update and query times.

Remark on the query operation. The QUERYVALUE operation described above asks only for an estimate of the value $\rho^*(G)$. This raises a natural question: Can we answer a more general query that asks for a subset of nodes which constitute an approximate densest subgraph (in time proportional to the number of nodes returned in response to the query)? The answer is yes. We can easily extend all the algorithms presented in this paper so as to enable them with this new feature (see the discussion after the proof of Corollary 2.7).

1.2 Our Results

Our main result is an efficient $(4+\epsilon)$-approximation algorithm for this problem (see Theorem 1.1). To be more specific, we present a randomized algorithm that can process a stream of polynomially many edge insertions/deletions starting from an empty graph using only $\tilde{O}(n)$ space, and with high probability, the algorithm maintains a $(4+\epsilon)$-approximation to the value of the densest subgraph. The algorithm has $\tilde{O}(1)$ amortized update-time and $\tilde{O}(1)$ query-time.

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 (edge insertions/deletions) in the dynamic stream, and define $m^{(t)} \leftarrow |E^{(t)}|$. Thus, we have $m^{(0)} = 0$ and $m^{(t)} \geq 0$ for all $t \geq 1$. We let $\text{OPT}^{(t)} = \rho^*(G^{(t)})$ denote the density of the densest subgraph in $G^{(t)}$.

Notation. Throughout this paper, the notations $\tilde{O}(.)$ and $\tilde{\Theta}(.)$ will hide $\text{poly}(\log n, 1/\epsilon)$ factors in the running times and space complexities of our algorithms, where $\epsilon \in (0, 1)$ is a small constant.

Theorem 1.1. *Fix a small constant $\epsilon \in (0, 1)$, a constant $\lambda > 1$, and let $T = \lceil n^\lambda \rceil$. We can process the first T updates (edge insertions/deletions) in a dynamic stream using $\tilde{O}(n)$ space, and maintain 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 stream is $\tilde{O}(T)$.*

Oblivious Adversary. We remark that Theorem 1.1 holds only when the sequence of edge insertions/deletions in the input graph does not depend on the random bits used by our algorithm. In other words, the "adversary", who decides upon the sequence of edge insertions/deletions, is "oblivious" to the random bits used in the algorithm. This is a standard assumption in the graph streaming literature. For example, the paper by Ahn, Guha and McGregor [1] also requires this assumption on the adversary. We prove Theorem 1.1 in Section 5. In addition, we obtain the following 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.

● **A $(4+\epsilon)$-approximation deterministic dynamic algorithm with $\tilde{O}(1)$ update time.** In Section 4, we present a deterministic algorithm that maintains a $(4+\epsilon)$-approximation to the value of the densest subgraph. This requires $\tilde{O}(m + n)$ space, and $\tilde{O}(1)$ update and query times.

● **Extensions to directed graphs.** In Section 6, we extend our result from Section 4 to directed graphs. Specifically, we present a deterministic dynamic algorithm that maintains a $(8 + \epsilon)$-approximation to the value of the densest subgraph of a directed graph. This requires $\tilde{O}(m + n)$ space, and $\tilde{O}(1)$ update and query times.

- **Sublinear-time algorithm:** We show that Charikar's linear-time linear-space algorithm [9] can be improved further. In particular, if the graph is represented by an incident list (this is a standard representation [10, 19]), 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 Section 7.

- **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 Section 8.

1.3   Previous work

To the best of our knowledge, our main algorithm is the first dynamic graph algorithm that requires $\tilde{O}(n)$ space and at the same time can quickly process each update and answer each query for densest subgraph. Previously, there was no space-efficient algorithm known for this problem, even when time efficiency is not a concern, and even for insert-only streams. In this insert-only model, Bahmani, Kumar, and Vassilvitskii [6] provided a deterministic $(2 + \epsilon)$-approximation $O(n)$-space 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].) In Section 3, we improve this result of Bahmani et al. in two respects: (a) We can process a dynamic stream of updates, and (b) we need only a single pass. Further, the space usage of our algorithm from Section 3 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 set of edges of G that belong to the subgraph induced by U. Observe that $\rho^*(G) \le \alpha(G) \le 2\rho^*(G)$. Thus, a γ-approximation algorithm for the arboricity problem will be a (2γ)-approximation algorithm for densest subgraph. 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 a work that appeared at about the same time as the preliminary version of this paper, Epasto et al. [14] presented a $(2 + \epsilon)$-approximation algorithm for densest subgraph which can handle arbitrary edge insertions and random edge deletions. After the preliminary version of our paper appeared, Esfandiari et al. [16] and McGregor et al. [33] presented semi-streaming algorithms for densest subgraph that give $(1 + \epsilon)$-approximation and require $\tilde{O}(n)$ space. The same result was obtained independently by Mitzenmacher et al. [34]. These improve the approximation ratio of our $(2 + \epsilon)$-approximation semi-streaming algorithm. Like our $(2 + \epsilon)$-approximation algorithm, their algorithms have an update-time of $\tilde{O}(1)$, but the query-time can be as large as $\tilde{\Omega}(n)$.

1.4 Overview of our techniques

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., [15, 32, 41]) and has played an important role in the studies of the densest subgraph problem (e.g., [6, 40]). 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 2.2.

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 is approximately $\rho^*(G)$, 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 4.)

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: 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, however, goes through (intuitively because all sets are sampled from the same graph and so their degree distributions are close enough). See Section 5 for further details.

1.5 Roadmap

The rest of the paper is organized as follows.

- We define the preliminary concepts and notations in Section 2.

- In Section 3, we present an algorithm that returns a $(2+\epsilon)$-approximation to the value of the densest subgraph. The algorithm processes a stream of edge insertions/deletions using only $\tilde{O}(n)$ bits of space, and at the end of the stream returns an estimate of $\rho^*(G)$ in $\tilde{O}(n)$ time. The output of the algorithm is correct with high probability.

- In Section 4, we present a deterministic algorithm that maintains a $(4+\epsilon)$-approximation to the value of the densest subgraph in $\tilde{O}(m+n)$ space. It has $\tilde{O}(1)$ update and query times.

- We present our main result in Section 5. Specifically, combining the techniques from Sections 3 and 4, we design an algorithm that maintains a $(4+\epsilon)$-approximation to the value of the densest subgraph with high probability, and requires only $\tilde{O}(n)$ space and $\tilde{O}(1)$ update time.

- In Section 6, we extend the result from Section 4 to directed graphs. Specifically, in a directed graph, we present a deterministic algorithm that maintains an $(8+\epsilon)$-approximation to the value of the densest subgraph using $\tilde{O}(m+n)$ space. It has $\tilde{O}(1)$ update and query times.

- In Sections 7 and 8 we present simple extensions of our result from Section 3, giving sublinear time and distributed-streaming algorithms for densest subgraph.

2 Notations and Preliminaries

We start by defining some notations that will be used throughout the rest of the paper. We denote the input graph by $G = (V, E)$. It has $n = |V|$ nodes and $m = |E|$ 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 $v \in V$. Consider any subset of nodes $S \subseteq V$. Let $E(S) = \{(u, v) \in E : u, v \in S\}$ denote the set of edges with both endpoints in S, and let $G(S) = (V, E(S))$ denote the subgraph of G induced by the nodes in S. Further, given any subset of edges $E' \subseteq E$ and any node $u \in V$, define $\mathcal{N}_u(S, E') = \{v \in \mathcal{N}_u \cap S : (u, v) \in E'\}$ and $D_u(S, E') = |\mathcal{N}_u(S, E')|$. In other words, $\mathcal{N}_u(S, E')$ is the subset of nodes in S that are neighbors of u in the subgraph induced by the edges in E', whereas $D_u(S, E')$ denotes the degree of u among the nodes in S in the same subgraph. For simplicity, we write $\mathcal{N}_u(S)$ and $D_u(S)$ instead of $\mathcal{N}_u(S, E)$ and $D_u(S, E)$. If the set of nodes $S \subseteq V$ is nonempty, then 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. Throughout the paper, the symbol $\tilde{O}(.)$ will be used to hide $\text{poly}(\log n, 1/\epsilon)$ factors in the running times of our algorithms, where $\epsilon > 0$ is some arbitrary small constant (the approximation guarantee will depend on ϵ). Finally, for any positive integer k, we will use the symbol $[k]$ to denote the set $\{1, \ldots, k\}$.

This paper deals with the "Densest Subgraph Problem", which is about finding a subset of nodes of maximum density. Specifically, we want to find a subset of nodes $S \subseteq V$ in the input graph $G = (V, E)$ that maximizes $\rho(S)$. Further, a subset $S \subseteq V$ is called a *γ-approximate densest*

subgraph, for $\gamma \geq 1$, iff $\gamma \cdot \rho(S) \geq \max_{S' \subseteq V} \rho(S')$. We will consider the densest subgraph problem in a dynamic setting. For a detailed description of our model, see Section 1.1. The main result of this paper is summarized below.

Theorem 2.1. *There is a dynamic data structure for the densest subgraph problem that requires $\tilde{O}(n)$ bits of space, has an amortized update time of $\tilde{O}(1)$, a query time of $O(1)$, and with high probability maintains a $(4 + \epsilon)$-approximation to the value of the densest subgraph.*

It follows that Theorem 2.1 gives a single-pass semi-streaming algorithm over dynamic streams for the approximate densest subgraph problem. And, unlike most other semi-streaming algorithms, Theorem 2.1 gives very fast update and query times.

2.1 Three basic properties

We now state three basic lemmas that will be used throughout the rest of the paper. The first lemma shows that the average degree of a set of nodes is twice its density.

Lemma 2.2. *For all $S \subseteq V$, we have $\delta(S) = 2 \cdot \rho(S)$.*

Proof. We have $\delta(S) = \sum_{v \in S} D_v(S)/|S| = 2 \cdot |E(S)|/|S| = 2 \cdot \rho(S)$. The second equality holds since every edge is incident upon two nodes. □

The second lemma gives simple upper and lower bounds on the maximum density of a subgraph.

Lemma 2.3. *Let $d^* = \max_{S \subseteq V} \rho(S)$ be the maximum density of any subgraph in G. Then $m/n \leq d^* < n$.*

Proof. Clearly, we have $d^* \geq \rho(V) = |E|/|V| = m/n$. On the other hand, consider any subset of nodes $S' \subseteq V$. We have $\rho(S') = |E(S')|/|S'| < n|S'|/|S'| = n$. The inequality holds since the maximum degree of a node is $(n-1)$, and hence the subgraph induced by the nodes in S' can have at most $n|S'|$ edges. Thus, we get: $d^* = \max_{S' \subseteq V} \rho(S') < n$. □

The final lemma will also be very helpful in analyzing our algorithm in later sections.

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^*.*

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'.

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

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 $v \in S^*$. □

2.2 (α, d, L)-decomposition

Our algorithms will use the concept of an "(α, d, L)-decomposition", as defined below. To give some intuitions behind Definition 2.5, suppose that we start by setting $Z_1 \leftarrow V$. Next, suppose that we have already constructed the subsets $Z_1 \supseteq \cdots \supseteq Z_i$ for some positive integer $i < L$. While constructing the next subset Z_{i+1}, we ensure that the following two conditions are satisfied.

- All the nodes $v \in Z_i$ with $D_v(Z_i) > \alpha d$ must be included in Z_{i+1}.

- All the nodes $v \in Z_i$ with $D_v(Z_i) < d$ must be excluded from Z_{i+1}.

Using this iterative procedure, we can build an (α, d, L)-decomposition.

Definition 2.5. *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 ith 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 be of crucial importance. Roughly speaking, they state that we can use the (α, d, L)-decomposition to $2\alpha(1+\epsilon)^2$-approximate the densest subgraph by trying different values of d in powers of $(1 + \epsilon)$.

Theorem 2.6. *Fix any $\alpha \geq 1$, $d \geq 0$, $\epsilon \in (0,1)$, $L = 2 + \lceil \log_{(1+\epsilon)} n \rceil$. Let $d^* = \max_{S \subseteq V} \rho(S)$ 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 $G = (V, E)$. Then 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 $\rho(Z_j) \geq d/(2(1+\epsilon))$.*

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 2.5 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 2.5, 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.

$\square$

Corollary 2.7. *Fix α, ϵ, L, d^* as in Theorem 2.6. Let $\pi, \sigma > 0$ be any two numbers satisfying $\alpha \cdot \pi < d^* < \sigma/(2(1+\epsilon))$. Fix any integer $K \geq 2 + \lceil \log_{(1+\epsilon)}(\sigma/\pi) \rceil$. Discretize the range $[\pi, \sigma]$ into powers of $(1+\epsilon)$, by defining $d_k = (1+\epsilon)^{k-1} \cdot \pi$ for every $k \in [K]$. Next, for every $k \in [K]$, construct an (α, d_k, L)-decomposition $(Z_1(k), \ldots, Z_L(k))$ of $G = (V, E)$. Let $k' = \max\{k \in [K] : Z_L(k) \neq \emptyset\}$. Then we have the following guarantees:*

1. $d^*/(\alpha(1+\epsilon)) \leq d_{k'} \leq 2(1+\epsilon)^2 \cdot d^*$.

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

Proof.

1. Note that $\pi < d^*/\alpha$ and $\sigma > 2d^*(1+\epsilon)$. Furthermore, we have $d_1 < \pi$ and $d_K \geq (1+\epsilon)\sigma$. This implies that $d_1 < d^*/\alpha$ and $d_K > 2d^*(1+\epsilon)^2$. Next, note that successive d_k values differ from each other by a factor of $(1+\epsilon)$. Accordingly, there exists some index $k \in [K]$ for which $d^*/(\alpha(1+\epsilon)) \leq d_k \leq 2(1+\epsilon)^2 \cdot d^*$. In other words, the set $Q = \{k \in [K] : d^*/(\alpha(1+\epsilon)) \leq d_k \leq 2(1+\epsilon)^2 \cdot d^*\}$ is nonempty. Let $k_1 = \min_{k \in Q}\{k\}$ and $k_2 = \max_{k \in Q}\{k\}$ respectively denote the minimum and maximum indices in the set Q. Observe that the set Q is "contiguous", i.e., $Q = \{k_1, k_1 + 1, \ldots, k_2\}$. Since the d_k values are discretized in powers of $(1+\epsilon)$, we have $k_1 < d^*/\alpha$ and $k_2 > 2d^*(1+\epsilon)$. Hence, by Theorem 2.6, we have $Z_L(k_1) \neq \emptyset$, and $Z_L(k) = \emptyset$ for all $k \geq k_2$. It follows that the index k' must satisfy the inequality $k_1 \leq k' \leq k_2$, which means that $k' \in Q$. Thus, we have $d^*/(\alpha(1+\epsilon) \leq k' \leq 2(1+\epsilon)^2 d^*$.

2. Suppose that the claim is false. Then we have $Z_L(k') \neq \emptyset$ and $\delta(Z_i(k')) = 2 \cdot \rho(Z_i(k')) < d_{k'}/(1+\epsilon)$ for every $i \in \{1, \ldots, L-1\}$. Then, applying an argument similar to the proof of case (1) in Theorem 2.6, we conclude that $|Z_{i+1}(k')| \leq |Z_i(k')|/(1+\epsilon)$ for every $i \in \{1, \ldots, L-1\}$, which implies that $Z_L(k') = \emptyset$. Thus, we arrive at a contradiction.

$\square$

We will use the above corollary as follows. Lemma 2.3 states that $m/n \leq d^* < n$. Thus, in Corollary 2.7, we can choose the values of π, σ and K in such a way which ensures that $K = \tilde{\Theta}(1)$. Hence, to maintain a $2\alpha(1+\epsilon)^3 = (2\alpha + \Theta(\epsilon))$-approximation of the maximum density, it suffices to maintain $K = \tilde{\Theta}(1)$ many (α, d, L)-decompositions and to keep track of the maximum d for which the topmost level (i.e., the node set V_L) of the decomposition is nonempty. This gives a query time of $O(1)$. In addition, if we want to answer a more general query which asks us to output a subgraph of approximate maximum density, then we simply keep track of the densities of all the node-induced subgraphs of the form $Z_i(k)$, where $i \in [L-1]$, $k \in [K]$, and output the one among them with maximum density. Since there are only $K = \tilde{\Theta}(1)$ many decompositions to consider, and since each such decomposition has $L = \tilde{\Theta}(1)$ levels, this can be done by incurring an additional cost of no more than $\Theta(KL) = \tilde{\Theta}(1)$ in the update time. It turns our that this simple extension applies to all the dynamic and streaming algorithms presented in the paper. Accordingly, for simplicity of exposition, from now on we only focus on the simpler query which asks for an estimate of the *value* of the densest subgraph (and not the subgraph itself).

9

2.3 Two results on ℓ_0-sampling and uniform hashing

We now state a well known result on ℓ_0-sampling in the streaming setting. All the streaming algorithms in this paper will use this result.

Lemma 2.8 (ℓ_0-sampler [24]). *We can process a dynamic stream of $O(poly\ n)$ updates in the graph $G = (V, E)$ in $\tilde{O}(1)$ space, and with high probability, at each step we can maintain a simple random sample from the set E. The algorithm takes $\tilde{O}(1)$ time to handle each update in the stream.*

The next lemma deals with uniform hashing in constant time and optimal space. We will use this lemma in Section 5.2.

Lemma 2.9. *[37]. Let $E^* = \binom{[n]}{2}$ be the set of all possible unordered pairs of nodes in V. Consider any two integers $w, q \geq 1$. We can construct a w-wise independent uniform hash function $h : E^* \to [q]$ using $O(w\ poly(\log w, \log q, \log n))$ bits of space. Given any $e \in E^*$, the hash value $h(e)$ can be evaluated in $O(1)$ time.*

2.4 Concentration bounds

We will use the following concentration bounds throughout the rest of this paper.

Theorem 2.10. *(Chernoff bound-I) Consider a collection 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 2.11. *(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 2.12. *(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 2.13. *(Chernoff bound with negative dependence) The Chernoff bounds, as stated in Theorems 2.10 and 2.11, hold even if the random variables $\{X_1, \ldots, X_t\}$ are negatively associated.*

3 A Semi-Streaming Algorithm

In this section, we present a single-pass semi-streaming algorithm for the densest subgraph problem. The algorithm requires only $\tilde{O}(n)$ bits of space, and at the end of the stream outputs a $(2 + \epsilon)$-approximation to the value of the densest subgraph with high probability. On the negative side, its update time can be as large as $\Omega(n)$. Our result in this section is stated in the theorem below.

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

We devote the rest of this section to the proof of Theorem 3.1. Throughout this section, we fix a small constant $\epsilon \in (0, 1/2)$ and a sufficiently large constant $c > 1$. Moreover, we set $\alpha \leftarrow (1 + \epsilon)/(1 - \epsilon)$, $L \leftarrow 2 + \lceil \log_{(1+\epsilon)} n \rceil$.

First, we show 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.

Consider any index $i \in \{1, \ldots, L-1\}$. Note that an edge $(u, v) \in E$ can appear multiple times in the collection of samples S_i. We will slightly abuse the notation introduced in the beginning of Section 2, and let $D_v(V', S_i)$ denote the degree of a node $v \in V' \subseteq V$ in the *multigraph* induced by the node set V' and the samples in S_i. With this notation in hand, 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 2.5, 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 jth 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 $L = \tilde{O}(1)$, we derive that the total space requirement is $O((n + m/d) \text{ poly} \log n)$. $\square$

Now, to turn Lemma 3.2 into a streaming algorithm, we simply have to invoke Lemma 2.8 which follows from a well-known result about ℓ_0-sampling in the streaming model [24], and a simple (but important) observation in Lemma 2.3.

Proof of Theorem 3.1. Define $\lambda^* = 2\alpha \cdot (cn(L-1)\log n)$ and $K^* = 2 + \lceil \log_{(1+\epsilon)}(8\alpha n^2) \rceil$. While processing the stream of edge insertions/deletions, we simultaneously run $\lambda^* K^*$ mutually independent copies of the ℓ_0-sampler as per Lemma 2.8. Furthermore, we maintain a counter to keep track of the number of edges in the graph. Initially, the counter is set to zero. After each edge insertion

(resp. deletion), the counter is incremented (resp. decremented) by one. Thus, at the end of the stream, we get $\lambda^* K^*$ mutually independent uniform random samples from the edge set E, and a correct estimate of the number of edges in the graph (which is given by $m = |E|$). All these steps can be implemented in $\tilde{\Theta}(\lambda^* K^*) = \tilde{\Theta}(n)$ space. Note that if $m = 0$, then clearly the maximum density of a subgraph is also zero. Thus, for the rest of the proof, we assume that $1 \leq m \leq \binom{n}{2}$.

Next, at the end of the stream, we define $\pi = m/(2\alpha n)$ and $\sigma = 2(1 + \epsilon)n$. By Lemma 2.3, we have $\alpha \cdot \pi < d^* < \sigma/(2(1 + \epsilon))$. Thus, the values of π and σ satisfies the condition required by Corollary 2.7. Hence, following Corollary 2.7, we set $K = 2 + \lceil \log_{(1+\epsilon)}(\sigma/\pi) \rceil$, and discretize the range $[\pi, \sigma]$ in powers of $(1 + \epsilon)$ by defining $d_k = (1 + \epsilon)^{k-1} \cdot \pi$ for every integer $k \in [K]$. Furthermore, we define $\lambda_k = cm(L - 1) \log n/d_k$ for all $k \in [K]$. Our goal is to construct an (α, d_k, L)-decomposition of G for every $k \in [K]$. By Lemma 3.2, for this we need $\sum_{k=1}^{K} \lambda_k$ many mutually independent random samples from E. But note that:

$$\lambda_k \geq \lambda_1 = cm(L - 1) \log n/d_1 = cm(L - 1) \log n/\pi = 2\alpha \cdot (c(L - 1)n \log n) = \lambda^* \quad \text{for all } k \in [K].$$

Next, since $\pi = m/(2\alpha n)$, $\sigma = 2(1 + \epsilon)n$ and $m \geq 1$, we also have $K^* = 2 + \lceil \log_{(1+\epsilon)}(8\alpha n^2) \rceil \geq 2 + \lceil \log_{(1+\epsilon)}(\sigma/\pi) \rceil = K$. To summarize, we infer the following guarantee.

$$\sum_{k=1}^{K} \lambda_K \leq \lambda^* K^*.$$

In other words, while processing the stream of edge insertions/deletions, we have collected sufficiently many mutually independent random samples from E so as to construct an (α, d_k, L)-decomposition for every integer $k \in [K]$. We can now get a $(2\alpha + \Theta(\epsilon)) = (2 + \Theta(\epsilon))$-approximation to the value of the densest subgraph by invoking Corollary 2.7. $\qquad \square$

Remark on the update and query times. The semi-streaming algorithm presented in this section runs $\tilde{\Theta}(n)$ mutually independent ℓ_0-samplers as per Lemma 2.8. Thus, when an edge is inserted into (resp. deleted from) the graph, the algorithm has to update each of these ℓ_0-samplers. This requires an update time of $\tilde{\Theta}(n)$. Finally, to answer a query about the maximum density, we have to first construct $\tilde{\Theta}(1)$ many (α, d, L) decompositions (for different values of d) from the sampled edges maintained by the ℓ_0 samplers, and then invoke Corollary 2.7. Thus, the query time is also $\tilde{\Theta}(n)$.

4 A Dynamic Algorithm with Fast Update and Query Times

The algorithm in Section 3 maintains a $(2 + \epsilon)$-approximation to the value of the densest subgraph and requires only $\tilde{O}(n)$ space, but its update and query times can be as large as $\Omega(n)$. In this section, we present an algorithm with $\tilde{O}(1)$ update and query times. This algorithm, however, has to store all the edges in the graph and hence has a space requirement of $\tilde{\Theta}(m + n)$. Furthermore, this algorithm maintains an approximation guarantee of $(4 + \epsilon)$.

Throughout this section, we set $\pi = 1/(4n)$, $\sigma = 4n$, $L = 2 + \lceil \log_{(1+\epsilon)} n \rceil$, and $\alpha = 2 + 3\epsilon$, 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 2.7, we discretize the range $[\pi, \sigma]$ in powers of $(1 + \epsilon)$ by defining the values $\{d_k\}, k \in [K]$, by setting $K = 2 + \lceil \log_{(1+\epsilon)}(\sigma/\pi) \rceil$.

We show in Section 4.1 how to maintain an (α, d_k, L)-decomposition of G for each $k \in [K]$ in $O(L/\epsilon) = O(\log n/\epsilon^2)$ amortized update time and $\Theta(m + n)$ space (see Theorem 4.2). Since $K = O(\log n/\epsilon)$, the total update time for all the K decompositions is $O(K \log n/\epsilon^2) = \tilde{O}(1)$ and

the total space requirement is also $O(K \cdot (m + n)) = \tilde{O}(m + n)$. By Corollary 2.7, this gives a $2\alpha(1+\epsilon)^2 = 4+O(\epsilon)$-approximation to the optimal density in G, for sufficiently small ϵ. To answer a query about the value of the densest subgraph, the algorithm needs to keep track of the index k' as defined in Corollary 2.7. Since there are $O(K) = \tilde{O}(1)$ decompositions to deal with, the time taken for this operation can be subsumed within the $\tilde{O}(1)$ update time. This gives us a query time of $O(1)$. We thus get the main result of this section, which is summarized below.

Theorem 4.1. *There is a deterministic dynamic algorithm that maintains a $(4+O(\epsilon))$-approximation to the value of the densest subgraph. The algorithm requires $\tilde{O}(m+n)$ space, has an amortized update time of $\tilde{O}(1)$ and a query time of $O(1)$.*

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

We present a deterministic data structure that is initially given α, d, L, and a graph $G = (V, E)$ with $|V| = n$, $E = \emptyset$. The data structure maintains an (α, d, L)-decomposition of the graph $G = (V, E)$ at each time-step, 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.

Theorem 4.2 summarizes our result. We devote the rest of Section 4.1 to its proof.

Theorem 4.2. *For every polynomially bounded $\alpha \geq 2 + 3\epsilon$, we can deterministically maintain an (α, d, L)-decomposition of $G = (V, E)$. Starting from an empty graph, our data structure handles a sequence of t update operations (edge insertions/deletions) in total time $O(tL/\epsilon)$. Thus, we get an amortized update time of $O(L/\epsilon)$. 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.*

Data Structures. We use the following data structures.

1. Every node $v \in V$ maintains L lists FRIENDS$_i[v]$, for $i \in \{1, \ldots, L\}$. For $i < \ell(v)$, the list 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 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 FRIENDS$_i[v]$ is empty. Each list is stored in a doubly linked list together with its size, 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 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 2.5), 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 FRIENDS lists of u and v by adding or removing neighbors in constant time. Next we check whether u or v are dirty. If so, we run the RECOVER() procedure described in Figure 1. Note that a single iteration of the 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.

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 $m = |E|$.

13

```
01.     WHILE there exists a dirty node $y$
02.          IF $D_y(Z_{\ell(y)}) > \alpha d$ and $\ell(y) < L$, THEN
03.               Increment the level of $y$ by setting $\ell(y) \leftarrow \ell(y) + 1$.
04.          ELSE IF $D_y(Z_{\ell(y)-1}) < d$ and $\ell(y) > 1$, THEN
05.               Decrement the level of $y$ by setting $\ell(y) \leftarrow \ell(y) - 1$.
```

Figure 1: RECOVER().

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 $O(tL/\epsilon)$.

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.

$$\mathcal{B} = \sum_{v \in V} \Phi(v) + \sum_{e \in E} \Psi(e) \tag{1}$$

$$\Phi(v) = \frac{1}{\epsilon} \sum_{i=1}^{\ell(v)-1} \max(0, \alpha d - D_v(Z_i)) \quad \text{for all nodes } v \in V \tag{2}$$

$$\Psi(u, v) = 2(L - \min(\ell(u), \ell(v))) + f(u, v) \quad \text{for all edges } (u, v) \in E \tag{3}$$

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. 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 1, 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 $2L/\epsilon$.

It remains to relate the change in the potential $\mathcal{B}$ with the amount of computation performed. See Section 4.1.2. For ease of exposition, we first describe a high level overview of the analysis.

4.1.1   A high level overview of the potential function based analysis

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 $\alpha = 2 + \epsilon$.

4.1.2 Analyzing the subroutine RECOVER().

We will analyze any single iteration of the WHILE loop in Figure 1. 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 $\text{COUNT}_{i+1}[y]$ by one unit. Finally, we set $\text{LEVEL}[y] \leftarrow i + 1$. 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. Towards this end, we first consider the edges incident upon y. Specifically, consider a node $x \in \mathcal{N}_y$.

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

15

- 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 other 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. Towards this end, we first consider the edges incident upon y. Specifically, 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. This concludes the proof of Theorem 4.2.

5 A Semi-Streaming Algorithm with Fast Update and Query times

In Section 3, we presented a semi-streaming algorithm that maintains a $(2 + \epsilon)$-approximation of $\rho^*(G)$. Specifically, the algorithm can process a dynamic stream of updates (edge insertions/deletions) using only $\tilde{O}(n)$ bits of space. Unfortunately, however, it has a large update time of $\tilde{\Theta}(n)$, and it answers a query only at the end of the stream (also in time $\tilde{\Theta}(n)$).

On the other hand, in Section 4 we presented an algorithm that maintains a $(4+\epsilon)$-approximation of $\rho^*(G)$. This algorithm has the advantage of having very fast (i.e., $\tilde{O}(1)$) update and query times. Furthermore, it can answer a query at any given time-instant (even in the middle of the stream). But, unlike the algorithm from Section 3 whose space complexity is $\tilde{O}(n)$, it has to store all the edges and requires $\tilde{\Theta}(m + n)$ bits of space.

In this section, we combine the techniques from Sections 3 and 4 to get a result that captures the best of both worlds. Specifically, we present a new algorithm that maintains a $(4+\epsilon)$-approximation of $\rho^*(G)$ while processing a stream of updates (edge insertions/deletions). We highlight that:

- The algorithm has very fast (i.e., $\tilde{O}(1)$) update and query times.

- It requires very little (i.e., $\tilde{O}(n)$) space.

- It can answer a query at any time-instant (i.e., even in the middle of the stream).

5.1 An Overview of Our Result

We denote the input graph by $G = (V, E)$. It has $|V| = n$ nodes, and in the beginning of our algorithm the graph is empty (i.e., $E = \emptyset$). Subsequently, our algorithm processes a "stream of updates" in the graph. Each update consists of an edge insertion/deletion. Specifically, at each "time-step", either an edge is inserted into the graph or an already existing edge is deleted from the graph. The node set of the graph, however, remains unchanged over time. For any integer $t \geq 0$, we let $G^{(t)} = (V, E^{(t)})$ denote the status of the input graph at time-step t (i.e., after the tth edge insertion/deletion). Thus, $G^{(0)} = (V, E^{(0)})$ denotes the status of G in the beginning, which implies that $E^{(0)} = \emptyset$. Further, we let $m^{(t)} = |E^{(t)}|$ denote the number of edges in the graph $G^{(t)}$. Finally, $\text{OPT}^{(t)} = \rho(G^{(t)})$ gives the value of the densest subgraph in $G^{(t)}$. We also use the notations and concepts introduced in Section 2. Our algorithm will maintain a value $\text{OUTPUT}^{(t)}$ at each time-step t. We want $\text{OUTPUT}^{(t)}$ to be a $(4 + \epsilon)$-approximation to $\text{OPT}^{(t)}$.

Throughout Section 5, we fix the symbols $\epsilon, \alpha, L, c, \lambda$ and T as defined below.

$$\epsilon \in (0,1) \text{ has a sufficiently small positive value.} \quad (4)$$

$$\alpha = 2 + \Theta(\epsilon) = 2 + c^* \cdot \epsilon, \text{ where } c^* \text{ is some constant independent of } \epsilon \text{ (to be decided later).} \quad (5)$$

$$L = 2 + \lceil \log_{(1+\epsilon)} n \rceil \quad (6)$$

$$\lambda \geq 1 \text{ is any positive constant, and } c \text{ is a constant such that } c >> \lambda. \quad (7)$$

$$T = \lceil n^\lambda \rceil \quad (8)$$

We use the symbols $\tilde{O}(.)$ and $\tilde{\Theta}(.)$ to hide $\text{poly}(\log n, 1/\epsilon)$ factors. We now state the main result.

Theorem 5.1. *Define T as in equation 8. There is an algorithm that processes a stream of T updates (starting from an empty graph), and satisfies the following properties with high probability:*

- *It uses only $\tilde{O}(n)$ bits of space.*

- *The total time taken to process the T edge insertions/deletions is $\tilde{O}(T)$. Thus, it has an amortized update time of $\tilde{O}(1)$.*

- *It maintains a value $\text{OUTPUT}^{(t)}$ such that for all time-steps $t \in [1,T]$ we have $\text{OUTPUT}^{(t)} \leq \text{OPT}^{(t)} \leq (4+\epsilon) \cdot \text{OPT}^{(t)}$. Thus, the algorithm maintains a $(4+\epsilon)$-approximation to the value of the densest subgraph while processing the stream of updates, and the query time is $O(1)$.*

Note that the algorithm in Theorem 5.1 works only for polynomially many time-steps (since $T = \Theta(n^\lambda)$ and λ is a constant). In contrast, we did not impose this restriction while presenting our semi-streaming algorithm in Section 3. To see why this is the case, recall that a semi-streaming algorithm maintains a "sketch" of the input while processing the stream of edge insertions/deletions. For our algorithm in Section 3, the sketch is simply the collection of random samples from the edge set of the input graph. Let $\text{SKETCH}^{(t)}$ denote the status of the sketch at time-step t (i.e., it corresponds to the graph $G^{(t)}$). Now, the following condition holds in Section 3:

- (P1) Fix any time-step t. With high probability, if we run the procedure in Section 3 on $\text{SKETCH}^{(t)}$, then this gives us a good approximation to the value of $\text{OPT}^{(t)}$.

A semi-streaming algorithm typically needs to invoke property (P1) *only at the end of the stream*, since it answers a query after processing all the edge insertions/deletions. In this section, however, we want an algorithm that can answer a query at any given time-instant (i.e., even in the middle of the stream). Thus, we want the stronger property stated below.

- (P2) The following event holds with high probability. For *every time-step $t \in [1,T]$*, we can get a good approximation to $\text{OPT}^{(t)}$ using $\text{SKETCH}^{(t)}$.

Intuitively, (P2) follows if we take a union bound over the complement of (P1) for all time-steps $t \in [1,T]$. But this can be done only if the length of the interval $[1,T]$ is bounded by some polynomial in n. Thus, we require that $T = \Theta(n^\lambda)$ for some constant λ.

5.1.1 Main technical challenges

At a very high level, the following approach seems natural for proving Theorem 5.1. First, using the techniques from Section 3, maintain $\tilde{O}(n)$ uniformly random samples from the edge set of the input graph. Next, using the techniques from Section 4, maintain (α, d, L)-decompositions on these randomly sampled edges. The first step should ensure that the algorithm requires only $\tilde{O}(n)$ bits

of space, while the second step should ensure that the algorithm requires $\tilde{O}(1)$ update and query times. Unfortunately, however, to implement this simple idea we need to overcome several intricate technical challenges. They are described below.

1. As stated at the end of Section 3, maintaining the random samples from the edge set E requires $\tilde{\Theta}(n)$ update time, since to process the insertion/deletion of an edge we have to update $\tilde{\Theta}(n)$ many ℓ_0-samplers. So the first challenge is to speed up the update time of the subroutine that maintains the randomly sampled edges. This is done in Section 5.2.

2. The algorithm in Section 3 uses one crucial observation that is captured in Lemma 2.3. Specifically, the maximum density of a subgraph of $G = (V, E)$ lies in the range $[m/n, n]$, where $m = |E|$ and $n = |V|$. Thus, we set $\pi = m/(2\alpha n)$ and $\sigma = 2(1 + \epsilon)n$, so that we have $\alpha \cdot \pi < d^* < \sigma/(2(1 + \epsilon))$, where $d^* = \max_{S \subseteq V} \rho(S)$ is the value of the maximum density of a subgraph of G (see the discussion immediately after the proof of Lemma 3.2). Then we discretize the range $[\pi, \sigma]$ in powers of $(1 + \epsilon)$, by defining the values $d_k, k \in [K]$, as per Corollary 2.7. For each $k \in [K]$, we construct an (α, d_k, L)-decomposition. Finally, we approximate the value of d^* by looking at the topmost levels (i.e., the node set V_L) of each of these decompositions. To implement this approach in Section 3, we wait till the end of the stream to get the value of m after all insertions/deletions. This is of crucial importance since the degree-threshold d_k (as per Corollary 2.7) for the kth (α, d, L)-decomposition depends on the value of π, which, in turn, depends on m. In this section, however, we have to maintain a solution at every time-step in the interval $[1, T]$. Consequently, we have to maintain an (α, d_k, L)-decomposition for each $k \in [K]$ throughout the interval $[1, T]$. Hence, the degree-threshold d_k of the kth decomposition changes over time as edges are inserted/deleted into the input graph. Thus, we need to extend the dynamic algorithm from Section 4.1 (which maintains an (α, d, L)-decomposition for a fixed d) so that it can handle the scenario where d changes over time. See Section 5.5 for further details.

3. Suppose that we want to construct an (α, d, L)-decomposition using $\tilde{O}(n)$ bits of space. In Section 3, to achieve this goal we used a collection of sets $\{S_i\}, i \in \{1, \dots, L - 1\}$. Recall the proof of Lemma 3.2 for details. Specifically, each S_i consisted of $\Theta(m \log n/d)$ many uniformly random samples from the edge set E. Given the subset of nodes Z_i, we used the samples S_i to construct the next subset $Z_{i+1} \subseteq Z_i$. Thus, we used different collections of sampled edges for different levels of the (α, d, L)-decomposition. The reason behind this was as follows: For the proof of Lemma 3.2 to be valid, it was crucial that the samples used for defining the set Z_{i+1} be chosen independently of the samples used for the sets $Z_1, \dots, Z_i$. This is in sharp contrast to our dynamic algorithm in Section 4.1 for maintaining an (α, d, L)-decomposition: that algorithm uses the same edge set E for different levels of the decomposition. Thus, we need to find a way to extend the potential function based analysis from Section 4.1 to a setting where different levels of the (α, d, L)-decomposition are concerned with different sets of edges (chosen uniformly at random). See Section 5.5 for further details.

Roadmap for the rest of Section 5. The rest of this section is organized as follows.

- In Section 5.2, we show how to maintain random samples from the edge set of the input graph in $\tilde{O}(n)$ space and $\tilde{O}(1)$ update time. For technical reasons, we have to run two separate algorithms for this purpose. Roughly speaking, the first algorithm (as stated in Theorem 5.2) maintains the entire edge set of the graph whenever $m^{(t)} = \tilde{O}(n)$, whereas the second algorithm (as stated in Theorem 5.3) maintains $\tilde{\Theta}(n)$ random samples from the edge set of the graph whenever $m^{(t)} = \tilde{\Omega}(n)$.

- In Section 5.3, we present a high level overview of our main algorithm. The main idea is to classify each time-step as either "dense" or "sparse", depending on the number of edges in the input graph. This classification is done in such a way that Theorem 5.2 applies to all sparse time-steps, whereas Theorem 5.3 applies to all dense time-steps.

- In Section 5.4, we present our algorithm for maintaining a $(4 + \epsilon)$-approximation to $\text{OPT}^{(t)}$ during all the sparse time-steps (see Theorem 5.4). This algorithm takes as input the set of edges maintained by the subroutine from Theorem 5.2.

- In Section 5.5, we present our algorithm for maintaining a $(4 + \epsilon)$-approximation to $\text{OPT}^{(t)}$ during all the dense time-steps (see Theorem 5.5). This algorithm takes as input the set of edges maintained by the subroutine from Theorem 5.3.

- Theorem 5.1 follows from Theorem 5.2 and Theorem 5.3.

5.2 Maintaining the randomly sampled edges in $\tilde{O}(1)$ update time

Intuitively, we want to maintain s uniformly random samples from the edge-set E of the input graph $G = (V, E)$, for some $s = \tilde{\Theta}(n)$. In Section 3, we achieved this by running $\tilde{\Theta}(n)$ mutually independent copies of the ℓ_0-sampler. This ensured a space complexity of $\tilde{\Theta}(n)$. However, after each edge insertion/deletion in the input graph, we had to update each of the ℓ_0-samplers. So the update time of our algorithm became $\tilde{\Theta}(n)$. In this section, we show how to bring down this update time (for maintaining the randomly sampled edges) to $\tilde{\Theta}(1)$ without compromising on the space complexity.

To see the high level idea behind our approach, suppose that the input graph has a large number of edges, i.e., $s \ll m = |E|$. We maintain s "buckets" $B_1, \ldots, B_s$. Whenever an edge e is inserted into the input graph, we insert the edge into a bucket chosen uniformly at random. When the edge gets deleted from the input graph, we also delete it from the bucket it was assigned to. Thus, the buckets $B_1, \ldots, B_s$ give a random partition of the edge set of the input graph $G = (V, E)$. We can maintain this partition using an appropriate hash function. Now, we run s mutually independent copies of the ℓ_0 sampler, *one for each bucket* $B_i, i \in \{1, \ldots, s\}$. Let S be the collection of edges returned by these ℓ_0-samplers. Thus, we have $S \subseteq E$, $|S| = s$, and any given edge $e \in E$ belongs to S with a probability that is very close to s/m. In other words, the set S is a set of s edges chosen uniformly at random from the edge set E (without replacement). This solves our problem. The space requirement is $\tilde{\Theta}(s) = \tilde{\Theta}(n)$ since we run s copies of the ℓ_0-sampler and each of this samplers needs $\tilde{\Theta}(1)$ space. Furthermore, unlike our algorithm in Section 3, here if an edge insertion/deletion takes place in the input graph G, then we only need to update a single ℓ_0 sampler (the one running on the bucket that edge was assigned to). This improves the update time to $\tilde{\Theta}(1)$.

To be more specific, we present two results in this section. In Theorem 5.2, we show how to maintain the edge set of the input graph $G = (V, E)$ at all time-steps where it is "sparse" (i.e., $m = |E|$ is small). Next, in Theorem 5.3, we show how to maintain $\tilde{\Theta}(n)$ uniformly random samples (without replacement) from the edge set E at all time-steps where the input graph is "dense" (i.e., $m = |E|$ is large). The proofs of Theorems 5.2 and 5.3 appear in Sections 5.2.1 and 5.2.2 respectively. Both the proofs crucially use well known results on ℓ_0-sampling in a streaming setting (see Lemma 2.8) and w-wise independent hash functions (see Lemma 2.9).

Theorem 5.2. *Starting from an empty graph, we can process the first T updates in a dynamic stream so as to maintain a random subset of edges $F^{(t)} \subseteq E^{(t)}$ at each time-step $t \leq T$. This requires $\tilde{O}(n)$ space and $\tilde{O}(1)$ worst case update time. Furthermore, the following conditions hold*

with high probability.

$$F^{(t)} = E^{(t)} \text{ at each time-step } t \leq T \text{ with } m^{(t)} \leq 8\alpha c^2 n \log^2 n. \tag{9}$$

Note that the proof of Theorem 5.2 is by no means obvious, for the following reason. It might happen that we have not included an edge e from the input graph in our sample (since the graph currently contains many edges). However, with the passage of time, a lot of edges get deleted from the graph so as to make it sparse, and at that time we might need to recover the edge e.

Theorem 5.3. *Fix any positive integer $s \leq 2\alpha cn \log n$. Starting from an empty graph, we can process the first T updates in a dynamic stream so as to maintain a random subset of edges $S^{(t)} \subseteq E^{(t)}$ at each time-step $t \in [T]$. This requires $\tilde{O}(n)$ space and $\tilde{O}(1)$ worst case update time. For every edge $e \in E^{(t)}$, let $X_e^{(t)} \in \{0,1\}$ be an indicator random variable that is set to one iff $e \in S^{(t)}$. Then we have:*

1. *The following condition holds with high probability.*

$$E\left[X_e^{(t)}\right] \in \left[(1 \pm \epsilon) \cdot \frac{s}{m^{(t)}}\right] \text{ for all edges } e \in E^{(t)}, \text{ at each } t \leq T \text{ with } m^{(t)} \geq 4\alpha c^2 n \log^2 n. \tag{10}$$

2. *At each time-step $t \leq T$, the random variables $\{X_e^{(t)}\}, e \in E^{(t)}$, are negatively associated.*

3. *Insertion/deletion of an edge in G leads to at most two insertion/deletions in the set S.*

5.2.1 Proof of Theorem 5.2

The algorithm. We define $w = q = 8\alpha c^2 n \log^2 n$, and build a w-wise independent hash function $h : E^* \to [w]$ as per Lemma 2.9. 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 $i \in [w]$, let $B_i^{(t)}$ denote the set of edges $e \in E^{(t)}$ with $h(e) = i$. So the edge set $E^{(t)}$ is partitioned into w random subsets $B_1^{(t)}, \ldots, B_w^{(t)}$.

As per Lemma 2.8, for each $i \in [w]$ we run $r = c^2 \log^2 n$ copies of a subroutine called STREAMING-SAMPLER. Specifically, for every $i \in [w]$ and $j \in [r]$, the subroutine STREAMING-SAMPLER(i, j) maintains a uniformly random sample from the set $B_i^{(t)}$ in $\tilde{O}(1)$ space and $\tilde{O}(1)$ worst case update time. Furthermore, the subroutines $\{$STREAMING-SAMPLERS$(i, j)\}, i \in [w], j \in [r]$, 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 $B_i^{(t)}, i \in [w]$, a given edge can occur multiple times in $Y^{(t)}$. We define $F^{(t)} \subseteq E^{(t)}$ to be the collection of those edges in $E^{(t)}$ that appear at least once in $Y^{(t)}$. Our algorithm maintains the subset $F^{(t)}$ at each time-step $t \in T$.

Update time. 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 $\tilde{O}(1)$ 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. We need $\tilde{O}(1)$ space to implement each STREAMING-SAMPLER(i, j), $i \in [w], j \in [r]$. Since $w = O(n \log^2 n)$ and $r = O(\log^2 n)$, the total space required by all the streaming samplers is $O(wr \text{ poly} \log n) = \tilde{O}(n)$. 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. 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 $i \in [w]$. The probability that any given edge $e \in E^{(t)}$ has $h(e) = i$ is equal to $1/w$. Since $w = 8\alpha c^2 n \log^2 n$, the linearity of expectation implies that $E\left[|B_i^{(t)}|\right] = m^{(t)}/w \leq 1$. Since the hash function h is w-wise independent, and since $m^{(t)} \leq w$, we can apply the Chernoff bound and infer that $|B_i^{(t)}| \leq c \log n$ with high probability. Now, a union bound over all $i \in [w]$ shows that with high probability, we have $|B_i^{(t)}| \leq c \log n$ for all $i \in [w]$. 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 $i \in [w]$. 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 $m^{(t)} \leq 8\alpha c^2 n \log^2 n$.

5.2.2 Proof of Theorem 5.3

We define $w = 2cs \log n$, $q = s$, and build a w-wise independent hash function $h : E^* \to [s]$ as per Lemma 2.9. 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.

This hash function partitions the edge set $E^{(t)}$ into s mutually disjoint buckets $\{B_j^{(t)}\}, j \in [s]$, where the bucket $B_j^{(t)}$ consists of those edges $e \in E^{(t)}$ with $h(e) = j$. For each $j \in [s]$, we run an independent copy of ℓ_0-SAMPLER, as per Lemma 2.8, that maintains a uniformly random sample from $B_j^{(t)}$. 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 [37], (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 $s = \tilde{O}(n)$, the procedure can be implemented in $\tilde{O}(n)$ space. We now show that this algorithm satisfies the three properties stated in Theorem 5.3.

1. Fix any time-step $t \in [1, T]$ where $m^{(t)} \geq 4\alpha c^2 n \log^2 n$. Since $s \leq 2\alpha cn \log n$, we infer that $m^{(t)} = |E^{(t)}| \geq 2cs \log n$. Hence, 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 $cs \log n$ and $2cs \log n$. Thus, for any $j \in [s]$ and any j', we have $|H_{j'}^{(t)} \cap B_j^{(t)}| \in [c \log n, 2c \log n]$ in expectation. Since the hash function h is $(2cs \log n)$-wise independent, by applying a Chernoff bound we infer that with high probability, the value $|H_{j'}^{(t)} \cap B_j^{(t)}|$ is within a $(1 \pm \epsilon)$ factor of its expectation. Applying the union bound over all j, j', we infer that with high probability, the sizes of all the sets $\left\{H_{j'}^{(t)} \cap B_j^{(t)}\right\}$ are within a $(1 \pm \epsilon)$ factor of their expected values – let us call this event $\mathcal{R}^{(t)}$. Note that $E[|B_j^{(t)}|] = m^{(t)}/s$

22

and $|B_j^{(t)}| = \sum_{j'} |B_j^{(t)} \cap H_{j'}^{(t)}|$. Hence, under the event $\mathcal{R}^{(t)}$, for all $j \in [s]$ the quantity $|B_j^{(t)}|$ is within a $(1 \pm \epsilon)$ factor of $m^{(t)}/s$. Under the same event $\mathcal{R}^{(t)}$, due to the ℓ_0-SAMPLERs, the probability that a given edge $e \in E^{(t)}$ with $h(e) = j$ (say) becomes part of $S^{(t)}$ is within a $(1 \pm \epsilon)$ factor of $1/|B_j^{(t)}|$, which, in turn, is within a $(1 \pm \epsilon)$ factor of $s/m^{(t)}$. This implies that for any given edge $e \in E^{(t)}$, we have $E\left[X_e^{(t)}\right] = \Pr\left[e \in S^{(t)}\right] \in \left[(1 \pm \epsilon) \cdot s/m^{(t)}\right]$ with high probability.

2. The property of negative association follows from the facts 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 $S_i^{(t)}$.

3. Finally, when an edge e is inserted into (resp. deleted from) the input graph, only the ℓ_0-sampler running on the bucket B_j, for $j = h(e)$ gets affected. This implies that a single edge insertion/deletion in the input graph leads to at most two edge insertions/deletions in the random subset of edges $S \subseteq E$.

5.3 A high level overview of our algorithm: Sparse and Dense Intervals

Our algorithm for Theorem 5.1 will consist of four different components. They are described below.

- (P1) This subroutine implements an algorithm as per Theorem 5.2.

- (P2) This subroutine implements $\tilde{\Theta}(1)$ independent copies of the algorithm in Theorem 5.3.

- (P3) For each $t \in [1, T]$, this subroutine classifies time-step t as either "dense" or "sparse". This is done on the fly, i.e., immediately after receiving the tth edge insertion/deletion in G. Consequently, the range $[1, T]$ is partitioned into "dense" and "sparse" "intervals", where a dense (resp. sparse) interval is a maximal and contiguous block of dense (resp. sparse) time-steps. For example, we say that $[t_0, t_1] \subseteq [1, T]$ is a dense 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. The sparse time-intervals are defined analogously. The subroutine ensures the following properties.

  1. We have $m^{(t)} \leq 8\alpha c^2 n \log^2 n$ for every sparse time-step $t \in [1, T]$. In other words, the input graph has a small number of edges in a sparse time-step. Note that $8\alpha c^2 n \log^2 n$ is also the threshold used in Theorem 5.2. Thus, with high probability, equation 9 holds at all sparse time-steps.
  2. We have $m^{(t)} \geq 4\alpha c^2 n \log^2 n$ for every dense time-step $t \in [1, T]$. In other words, the input graph has a large number of edges in a dense time-step. Note that $4\alpha c^2 n \log^2 n$ is also the threshold used in Theorem 5.3 (part 1). Thus, with high probability, equation 10 holds at all dense time-steps.
  3. If a dense interval begins at a time-step t, then we have $m^{(t)} = 1 + 8\alpha c^2 n \log^2 n$.
  4. Every dense (resp. sparse) interval spans at least $4\alpha c^2 n \log^2 n$ time-steps, unless it is the interval ending at T.

  The classification of a time-step as dense or sparse is done according to the procedure outlined in Figure 2. This procedure can be easily implemented in $\tilde{\Theta}(1)$ space and $\Theta(1)$ update time, since all we need is a counter that keeps track of the number of edges in the input graph while processing the stream of updates. Furthermore, it is easy to check that the procedure in Figure 2 ensures all the four properties described above.

- (P4) This subroutine takes as input the set of edges maintained by (P1). Furthermore, it also has access to the output of the subroutine (P3). At all sparse time-steps, it maintains a value $\text{OUTPUT}^{(t)}$ that gives a $(4 + \epsilon)$-approximation of $\text{OPT}^{(t)}$. See Section 5.4 for further details.

- (P5) This subroutine takes as input the subsets of edges maintained by (P2). Furthermore, it also has access to the output of the subroutine (P3). At all dense time-steps, it maintains a value $\text{OUTPUT}^{(t)}$ that gives a $(4 + \epsilon)$-approximation of $\text{OPT}^{(t)}$. See Section 5.5 for further details.

- Theorem 5.1 follows from Theorem 5.4 and Theorem 5.5.

<div style="border:1px solid;">

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 $m^{(t)} \leq 8\alpha c^2 n \log^2 n$, THEN
05.	Classify time-step t as sparse.
06.	ELSE IF $m^{(t)} > 8\alpha c^2 n \log^2 n$, THEN
07.	Classify time-step t as dense.
08.	ELSE IF time-step $(t-1)$ was dense, THEN
09.	IF $m^{(t)} \geq 4\alpha c^2 n \log^2 n$, THEN
10.	Classify time-step t as dense.
11.	ELSE IF $m^{(t)} < 4\alpha c^2 n \log^2 n$, THEN
12.	Classify time-step t as sparse.

</div>

Figure 2: CLASSIFY-TIME-STEPS(.).

5.4 Algorithm for sparse intervals

In this section, we show how to maintain a $(4 + \epsilon)$-approximation to the value of the densest subgraph during the sparse time-intervals. Specifically, we prove the following theorem.

Theorem 5.4. *There is an algorithm that uses $\tilde{O}(n)$ space and maintains a value $\text{OUTPUT}^{(t)}$ at every sparse time-step $t \leq T$. The algorithm gives the following two guarantees with high probability.*

- $\text{OPT}^{(t)}/(4 + \epsilon) \leq \text{OUTPUT}^{(t)} \leq \text{OPT}^{(t)}$ *at every sparse time-step $t \leq T$.*

- *The algorithm takes $\tilde{O}(T)$ time to process the stream of T updates in G. In other words, the amortized update time is $\tilde{O}(1)$.*

The algorithm for Theorem 5.4 consists of two major ingredients.

- First, we run a subroutine as per Theorem 5.2 while processing the stream of T updates.[2] This ensures that with high probability, we maintain a subset $F^{(t)} \subseteq E^{(t)}$ such that $F^{(t)} = E^{(t)}$ at every sparse time-step $t \leq T$. The worst case update time and space complexity of this subroutine are $\tilde{O}(1)$ and $\tilde{O}(n)$ respectively.

---

[2]Note that this is the same subroutine (P1) from Section 5.3.

- Second, we run our dynamic algorithm – which we refer to as DYNAMIC-ALGO – from Section 4 on the graph $(V, F^{(t)})$ during every sparse time-interval.[3] Since $F^{(t)} = E^{(t)}$ throughout the duration of such an interval (with high probability), this allows us to maintain an OUTPUT$^{(t)} \in \left[\text{OPT}^{(t)}/(4 + \epsilon), \text{OPT}^{(t)} \right]$ at every sparse time-step $t \leq T$. As the input 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. Towards this end, fix any sparse time-interval $[t_0, t_1]$, and let $C(t_0, t_1)$ denote the amount of computation performed during this interval by the subroutine DYNAMIC-ALGO. Consider two possible cases.

- *Case 1.* $(t_1 < T)$

  In this case, our analysis from Section 4 implies that $C(t_0, t_1) = \tilde{O}(n + (t_1 - t_0))$. Since $t_1 < T$, the subroutine (P3) from Section 5.3 ensures that $(t_1 - t_0) = \Omega(n)$.[4] This gives us the guarantee that $C(t_0, t_1) = \tilde{O}(t_1 - t_0)$.

- *Case 2.* $(t_1 = T)$

  In this case, the sparse time-interval under consideration ends at T. Thus, if $(t_1 - t_0) = o(n)$, then we would have $C(t_0, t_1) = \tilde{O}(n)$. Else if $(t_1 - t_0) = \Omega(n)$, then applying a similar argument as in Case 1, we get $C(t_0, t_1) = \tilde{O}(t_1 - t_0)$.

Let $\left[t_0^i, t_1^i \right]$ denote the ith 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.

$$C = \left(\sum_i \tilde{O}((t_1^i - t_0^i)) \right) + \tilde{O}(n) \leq \tilde{O}(T) + \tilde{O}(n) = \tilde{O}(T) \tag{11}$$

The last equality holds as $T = \Theta(n^\lambda)$ and $\lambda \geq 1$ (equation 7). This shows that the amortized update time of the algorithm is $\tilde{O}(1)$, thereby concluding the proof of Theorem 5.4.

5.5   Algorithm for dense intervals

In this section, we show how to maintain a $(4 + \epsilon)$-approximation to the value of the densest subgraph during the dense time-intervals. Specifically, we prove the following theorem.

Theorem 5.5. *There is an algorithm that uses $\tilde{O}(n)$ space and maintains a value OUTPUT$^{(t)}$ at every dense time-step $t \leq T$. The algorithm gives the following two guarantees with high probability.*

- OPT$^{(t)}/(4 + \epsilon) \leq$ OUTPUT$^{(t)} \leq$ OPT$^{(t)}$ *at every dense time-step $t \leq T$.*

- *The algorithm takes $\tilde{O}(T)$ time to process the stream of T updates in G. In other words, the amortized update time is $\tilde{O}(1)$.*

---

[3]We can identify each sparse time-interval using the subroutine (P3) from Section 5.3.
[4]See item 4 in the description of the subroutine (P3).

5.5.1 Basic building block for proving Theorem 5.5

For every time-step $t \in [1, T]$, we define:

$$\pi^{(t)} = m^{(t)}/(2\alpha n), \text{ and } \sigma = 2(1+\epsilon)n \tag{12}$$

Accordingly, Lemma 2.3 implies that:

$$\alpha \cdot \pi^{(t)} < \text{OPT}^{(t)} < \sigma/(2(1+\epsilon)) \tag{13}$$

We now discretize the range $[\pi^{(t)}, \sigma]$ in powers of $(1+\epsilon)$ as in Corollary 2.7. Specifically, we define:

$$K = 2 + \lceil \log_{(1+\epsilon)}(\sigma \cdot (2\alpha n)) \rceil \tag{14}$$

$$d_k^{(t)} = (1+\epsilon)^{k-1} \cdot \pi^{(t)} \text{ for all } k \in [K] \tag{15}$$

Note that $m^{(t)} \geq 1$ at every dense time-step $t \leq T$. This ensures that $\pi^{(t)} \geq 1/(2\alpha n)$, and hence we have $K \geq 2 + \lceil \log_{(1+\epsilon)}(\sigma/\pi^{(t)}) \rceil$ during those time-steps. In other words, the values of $\pi^{(t)}, \sigma$ and K satisfy the condition stated in Corollary 2.7 during the dense time-intervals. Also note that the value of K *does not* depend on the time-step t under consideration.[5] Furthermore, we have $K = \tilde{O}(1)$.

We want to maintain a $2\alpha(1+\epsilon)^3 = (4 + \Theta(\epsilon))$-approximation of the maximum density during the dense time-intervals. For this purpose it suffices to maintain an $(\alpha, d_k^{(t)}, L)$-decomposition of $G^{(t)}$ for each $k \in [K]$ (see Corollary 2.7).

Thus, we conclude that Theorem 5.5 follows from Theorem 5.6. Accordingly, for the rest of Section 5.5, we fix an index $k \in [K]$, and focus on proving Theorem 5.6.

Theorem 5.6. *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 $V = Z_1^{(t)} \supseteq \cdots \supseteq Z_L^{(t)}$ at every dense time-step $t \in [T]$. The algorithm is randomized and gives the following two guarantees with high probability.*

- *The tuple $(Z_1^{(t)} \ldots Z_L^{(t)})$ is an $(\alpha, d_k^{(t)}, L)$-decomposition of $G^{(t)}$ at every dense time-step $t \in [T]$.*

- *The algorithm takes $\tilde{O}(T)$ time to process the stream of T updates. In other words, the algorithm has an amortized update time of $\tilde{O}(1)$.*

5.5.2 Overview of our algorithm for Theorem 5.6

Our algorithm for proving Theorem 5.6 consists of two major ingredients.

- First, at each dense time-step $t \leq T$ we maintain a collection of $(L-1)$ random subsets of edges $S_1^{(t)}, \ldots, S_{L-1}^{(t)} \subseteq E^{(t)}$. To maintain these random subsets, we need to run $(L-1)$ mutually independent copies of the algorithm in Theorem 5.3 (for an appropriate value of s).

- Second, using the random subsets $S_1^{(t)}, \ldots, S_{L-1}^{(t)} \subseteq E^{(t)}$, we maintain an $(\alpha, d_k^{(t)}, L)$ decomposition $(Z_1^{(t)}, \ldots, Z_L^{(t)})$ during the dense time-steps. Specifically, we set $Z_1^{(t)} = V$. Next, for $i = 1$ to $(L-1)$, we construct the node set $Z_{i+1}^{(t)} \subseteq Z_i^{(t)}$ by looking at the degrees $D_v(Z_i^{(t)}, S_i^{(t)})$ of the nodes $v \in Z_i^{(t)}$ among the edges $e \in S_i^{(t)}$. This follows the spirit of the construction in Section 3 (but note that there we did not concern ourselves with the update time).

---

[5]This is especially important since we want to maintain an $(\alpha, d_k^{(t)}, L)$-decomposition for each $k \in [K]$ during the dense time-intervals. If K were a function of t, then the number of such decompositions would have varied over time.

From the proof of Lemma 3.2, we infer that each $S_i^{(t)}$ should contain $s^{(t)} = cm^{(t)} \log n / d_k^{(t)}$ random samples from the edge set $E^{(t)}$. Equations 12 and 15 ensure that $s^{(t)} = 2\alpha cn \log n / (1 + \epsilon)^{k-1}$, which means that the value of $s^{(t)}$ is independent of the time-step t under consideration. Accordingly, for the rest of Section 5.5, we omit the superscript t from $s^{(t)}$ and define:

$$s = 2\alpha cn \log n / (1 + \epsilon)^{k-1} = cm^{(t)} \log n / d_k^{(t)} \text{ for each dense time-step } t \leq T. \quad (16)$$

Furthermore, since $k \geq 1$, we observe that:

$$s \leq 2\alpha cn \log n \quad (17)$$

Thus, the value assigned to s satisfies the condition dictated by Theorem 5.3.

To summarize, our first subroutine maintains $(L-1)$ mutually independent copies of the algorithm in Theorem 5.3 (for the value of s as defined by equations 16, 17). The random subsets of edges maintained by them are denoted as $S_1^{(t)}, \ldots, S_{L-1}^{(t)}$. Since $L = \tilde{\Theta}(1)$ (see equation 6), Theorem 5.3 ensures that this subroutine has $\tilde{O}(1)$ worst case update time and $\tilde{O}(n)$ space complexity.

Next, we present another subroutine DYNAMIC-STREAM (see Section 5.5.3) that is invoked only during the dense time-intervals.[6] This subroutine can access the random subsets $\left\{ S_i^{(t)} \right\}, i \in [L-1]$, and it maintains L subsets of nodes $V = Z_1^{(t)} \supseteq \cdots \supseteq Z_L^{(t)}$ at each dense time-step $t \in [T]$.

Roadmap. The rest of Section 5.5 is organized as follows.

- In Section 5.5.3, we fix a dense time-interval $[t_0, t_1]$, and describe how the subroutine DYNAMIC-STREAM processes the edge insertions/deletions in the input graph during this interval. We show that DYNAMIC-STREAM maintains an $(\alpha, d_k^{(t)}, L)$-decomposition of the input graph $G^{(t)}$ throughout the duration of this dense time-interval (see Lemma 5.8).

- Section 5.5.4 presents the data structures for implementing the subroutine DYNAMIC-STREAM.

- In Section 5.5.5, we make some preliminary observations about the running time of the subroutine DYNAMIC-STREAM, and analyze its space complexity (see Lemma 5.9).

- In Section 5.5.6, we analyze the amortized update time of the subroutine DYNAMIC-STREAM (see Corollary 5.14). Theorem 5.6 follows from Lemma 5.8, Lemma 5.9 and Corollary 5.14.

- In Sections 5.5.7 and 5.5.8 are devoted to the proof of two lemmas stated in Section 5.5.6.

5.5.3 The subroutine DYNAMIC-STREAM for a dense time-interval $[t_0, t_1]$

Fix any dense time-interval $[t_0, t_1] \subseteq [1, T]$. We will maintain L subsets of nodes $V = Z_1^{(t)} \supseteq \cdots \supseteq Z_L^{(t)}$ at each time-step $t \in [t_0, t_1]$. With high probability, we will prove that throughout the interval the tuple $(Z_1^{(t)}, \ldots, Z_L^{(t)})$ remains a valid $(\alpha, d_k^{(t)}, L)$-decomposition of $G^{(t)} = (V, E^{(t)})$.

We run $(L-1)$ mutually independent copies of the algorithm stated in Theorem 5.3 (for the value of s as defined by equations 16, 17). Hence, at each time-step $t \in [t_0, t_1]$, we can access the mutually independent random subsets of edges $S_1^{(t)}, \ldots, S_{L-1}^{(t)} \subseteq E^{(t)}$ as defined in Section 5.5.2.

Initialization in the beginning of the dense time-interval $[t_0, t_1]$.

Just before time-step t_0, we perform the initialization step outlined in Figure 3. It ensures that $Z_1^{(t_0-1)} = V$ and $Z_i^{(t_0-1)} = \emptyset$ for all $i \in \{2, \ldots, L\}$.

---

[6]The dense time-intervals can be easily identified using the subroutine (P3) from Section 5.3.

```
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 3: INITIALIZE(.).

Handling an update in G during the dense time-interval $[t_0, t_1]$.

Consider an edge insertion/deletion in the input graph during time-step $t \in [t_0, t_1]$. The edge set $E^{(t)}$ is different from the edge set $E^{(t-1)}$. Accordingly, for all $i \in [L-1]$, the subset of edges $S_i^{(t)}$ may differ from the subset of edges $S_i^{(t-1)}$. Therefore, at this time we call the subroutine RECOVER-SAMPLE(t) in Figure 4. 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 $(Z_1^{(t)}, \ldots, Z_L^{(t)})$.

As in Section 3, we want to ensure that the node set Z_i is completely determined by the outcomes of the random samples in $\{S_j\}, j < i$ (see the proof of Lemma 3.2). Towards this end, we observe the following lemma.

```
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 $D_y(Z_i^{(t)}, S_i^{(t)}) > (1-\epsilon)^2 \alpha c \log n$.
06.             Let $B_i^{(t)}$ be the set of nodes $y \in Z_i^{(t)}$ having $D_y(Z_i^{(t)}, S_i^{(t)}) < (1+\epsilon)^2 c \log n$.
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 4: RECOVER-SAMPLE(t).

Lemma 5.7. *Fix any time-step $t \in [t_0, t_1]$, any index $i \in [L-1]$, and consider the set $Z_i^{(t)}$ as defined by the procedure in Figure 4.*

1. *The node set $Z_i^{(t)}$ is completely determined by the contents of the sets $\left\{ S_j^{(t)} \right\}, j < i$.*

2. *The contents of the random sets $\left\{ S_j^{(t)} \right\}, j \geq i$, are independent of the contents of the set $Z_i^{(t)}$.*

Proof. Follows from the description of the procedure in Figure 4. $\qquad\square$

We now prove the correctness of our algorithm. Specifically, we show that with high probability the tuple $(Z_1^{(t)}, \ldots, Z_L^{(t)})$ remains a valid $(\alpha, d_k^{(t)}, L)$-decomposition of the input graph throughout the dense time-interval under consideration.

Lemma 5.8. *With high probability, at each time-step $t \in [t_0, t_1]$ the tuple $(Z_1^{(t)} \ldots Z_L^{(t)})$ maintained by the subroutine DYNAMIC-STREAM gives an $(\alpha, d_k^{(t)}, L)$-decomposition of the input graph $G^{(t)}$.*

28

Proof. For each time-step $t \in [t_0, t_1]$ and index $i \in [L-1]$, we define an event $\mathcal{E}_i^{(t)}$ as follows.

- The event $\mathcal{E}_i^{(t)}$ occurs iff the following two conditions hold simultaneously.
  - $Z_{i+1}^{(t)} \supseteq \{v \in Z_i^{(t)} : D_v(Z_i^{(t)}, E^{(t)}) > \alpha d_k^{(t)}\}$, and
  - $Z_{i+1}^{(t)} \cap \{v \in Z_i^{(t)} : D_v(Z_i^{(t)}, E^{(t)}) < d_k^{(t)}\} = \emptyset$.

We now define another event $\mathcal{E}^{(t)}$ for each time-step $t \in [t_0, t_1]$.

- $\mathcal{E}^{(t)} = \bigcap_{i=1}^{L-1} \mathcal{E}_i^{(t)}$.

By Definition 2.5, the tuple $(Z_1^{(t)} \ldots Z_L^{(t)})$ is an $(\alpha, d_k^{(t)}, L)$-decomposition of $G^{(t)}$ iff the event $\mathcal{E}^{(t)}$ occurs. Below, we show that $\Pr\left[\mathcal{E}_i^{(t)}\right] \geq 1 - 1/(\text{poly } n)$ for any given $i \in \{1, \ldots, L-1\}$ and $t \in [t_0, t_1]$. Taking a union bound over all $i \in \{1, \ldots, L-1\}$, we get $\Pr\left[\mathcal{E}^{(t)}\right] \geq 1 - 1/(\text{poly } n)$ at each time-step $t \in [t_0, t_1]$. Hence, the lemma follows by taking a union bound over all $t \in [t_0, t_1]$.

For the rest of the proof, fix any time-step $t \in [t_0, t_1]$ and index $i \in \{1, \ldots, L-1\}$.

- Fix any instance of the random set $Z_i^{(t)}$ and condition on this event.

  - By Theorem 5.3 and equation 16, each edge $e \in E^{(t)}$ appears in $S_i^{(t)}$ with probability $(1 \pm \epsilon)s/m^{(t)} = (1 \pm \epsilon)c \log n / d_k^{(t)}$. Furthermore, these events are negatively associated (see Section 2.4).

  Consider any node $v \in Z_i^{(t)}$ with $D_v(Z_i^{(t)}, E^{(t)}) > \alpha d_k^{(t)}$. By linearity of expectation:

  $$E\left[D_v(Z_i^{(t)}, S_i^{(t)})\right] \geq \alpha d_k^{(t)} \cdot (1-\epsilon)c \log n / d_k^{(t)} = (1-\epsilon)\alpha c \log n.$$

  Since the contents of the random set $S_i^{(t)}$ are independent of the contents of $Z_i^{(t)}$ (see Lemma 5.7), we can apply a Chernoff bound on this expectation, and derive that:

  $$\Pr\left[D_v(Z_i^{(t)}, S_i^{(t)}) > (1-\epsilon)^2 \alpha c \log n \mid Z_i^{(t)}\right] \geq 1 - 1/(\text{poly } n) \tag{18}$$

  Now, Figure 4 implies that if $D_v(Z_i^{(t)}, S_i^{(t)}) > (1-\epsilon)^2 \alpha c \log n$, then the node v becomes part of $Z_{i+1}^{(t)}$. Thus, applying equation 18 we get:

  $$\Pr\left[v \in Z_{i+1}^{(t)} \mid Z_i^{(t)}\right] \geq \Pr\left[D_v(Z_i^{(t)}, S_i^{(t)}) > (1-\epsilon)^2 \alpha c \log n \mid Z_i^{(t)}\right] \geq 1 - 1/(\text{poly } n) \tag{19}$$

  Note that equation 19 would have been true even if v did not belong to $Z_i^{(t)}$. Furthermore, equation 19 holds regardless of the event $\mathcal{E}^{(t-1)}$.

  Next, consider any node $u \in Z_i^{(t)}$ with $D_u(Z_i^{(t)}, E^{(t)}) < d_k^{(t)}$. A similar argument shows:

  $$\Pr\left[u \notin Z_{i+1}^{(t)} \mid Z_i^{(t)}\right] \geq \Pr\left[D_u(Z_i^{(t)}, E^{(t)}) < (1+\epsilon)^2 c \log n \mid Z_i^{(t)}\right] \geq 1 - 1/(\text{poly } n) \tag{20}$$

  Note that equation 20 would have been true even if u did not belong to $Z_i^{(t)}$. Furthermore, equation 20 holds regardless of the event $\mathcal{E}^{(t-1)}$.

  Thus, applying a union bound on equations 19, 20 over all nodes in $Z_i^{(t)}$, we infer that:

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

29

Since the guarantee $\Pr\left[\mathcal{E}_i^{(t)} \mid Z_i^{(t)}\right] \geq 1 - 1/(\text{poly } n)$ holds for every possible instantiation of $Z_i^{(t)}$, we get $\Pr\left[\mathcal{E}_i^{(t)}\right] \geq 1 - 1/(\text{poly } n)$. Taking a union bound over all indices $i \in \{1, \ldots, L-1\}$, we infer that $\Pr\left[\mathcal{E}^{(t)}\right] = \bigcap_{i=1}^{L-1} \mathcal{E}_i^{(t)} \geq 1 - 1/(\text{poly } n)$.

In other words, at every time-step $t \in [t_0, t_1]$ the event $\mathcal{E}^{(t)}$ occurs with high probability, and this holds *regardless of the past events* $\mathcal{E}^{(t')}, t' < t$. Hence, taking a union bound over all time-steps in the interval $[t_0, t_1]$, we get: With high probability, for all $t \in [t_0, t_1]$ the tuple $(Z_1^{(t)} \ldots Z_L^{(t)})$ maintained by the subroutine DYNAMIC-STREAM gives an $(\alpha, d_k^{(t)}, L)$-decomposition of the input graph $G^{(t)}$. This concludes the proof of the lemma. $\qquad\square$

5.5.4 Data structures for implementing the procedure in Figure 4

Recall the notations introduced in Section 2.

- Consider any node $v \in V$ and any $i \in \{1, \ldots, L-1\}$. We maintain the doubly linked lists $\{\text{FRIENDS}_i[v, j]\}, 1 \leq j \leq L-1$ as defined below. Each of these lists is defined by the neighborhood of v induced by the sampled edges in S_i. Recall Definition 2.5.

    - If $i \leq \ell(v)$, then we have:
        * $\text{FRIENDS}_i[v, j]$ is empty for all $j > i$.
        * $\text{FRIENDS}_i[v, j] = \mathcal{N}_v(Z_j, S_i)$ for $j = i$.
        * $\text{FRIENDS}_i[v, j] = \mathcal{N}_v(V_j, S_i)$ for all $j < i$.
    - Else if $i > \ell(v)$, then we have:
        * $\text{FRIENDS}_i[v, j]$ is empty for all $j > \ell(v)$.
        * $\text{FRIENDS}_i[v, j] = \mathcal{N}_v(Z_j, S_i)$ for $j = \ell(v)$.
        * $\text{FRIENDS}_i[v, j] = \mathcal{N}_v(V_j, S_i)$ for all $j < \ell(v)$.

For every node $v \in V$, we maintain a counter $\text{DEGREE}_i[v]$, which keeps track of the number of nodes in $\text{FRIENDS}_i[v, i]$. Note that if $\ell(v) < i$, then this counter equals zero. Further, we maintain a doubly linked list $\text{DIRTY-NODES}[i]$. This consists of all the nodes $v \in V$ having either $\left\{\text{DEGREE}_i[v] > (1-\epsilon)^2 \alpha c \log n \text{ and } \ell(v) = i\right\}$ or $\left\{\text{DEGREE}_i[v] < (1+\epsilon)^2 c \log n \text{ and } \ell(v) > i\right\}$.

Remark. Note that for any given index $i \in \{1, \ldots, L-1\}$ and any time-step $t \leq T$, an edge $e \in E^{(t)}$ of the input graph appears at most once among the samples in $S_i^{(t)}$ (see Theorem 5.3). Thus, the number of occurrences of an edge among the samples $S_1^{(t)}, \ldots, S_{L-1}^{(t)}$ is at most $(L-1)$.

5.5.5 Implementing the procedure in Figure 4 during a dense time-interval $[t_0, t_1]$

Consider any dense time-interval $[t_0, t_1] \subseteq [1, T]$, and fix any time-step $t \in [t_0, t_1]$. The tth edge insertion/deletion in the input graph might lead to some changes in the random subsets of edges $S_1, \ldots, S_{L-1} \subseteq E$. However, Theorem 5.3 implies that an edge insertion/deletion in G can lead to at most two edge insertions/deletions in S_i, for all $i \in \{1, \ldots, L-1\}$. Thus, due to the tth update in the stream, there can be at most $O(L) = \tilde{O}(1)$ insertions/deletions in the random sets $S_1, \ldots, S_{L-1}$ (see equation 6). After each such edge insertion/deletion in an $S_i, i \in \{1, \ldots, L-1\}$, we update the relevant data structures described in Section 5.5.4. Since an edge $(u, v) \in S_i$ can potentially appear only in the lists $\text{FRIENDS}_i[x, j]$ with $x \in \{u, v\}$ and $j \in \{1, \ldots, L-1\}$ (see Section 5.5.4), we reach the following conclusion:

- When an edge insertion/deletion in G leads to changes in the random edge sets $S_i, \ldots, S_{L-1} \subseteq E$, we can update the FRIENDS and DIRTY-NODES lists from Section 5.5.4 in $\tilde{O}(1)$ time.

After updating the edge sets $S_1, \ldots, S_{L-1}$ and the FRIENDS and DIRTY-NODES lists, we run the procedure described in Figure 4. Now, consider the ith iteration of the main FOR loop (Steps 05-10) in Figure 4, for some index $i \in \{1, \ldots, L-1\}$. 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 DIRTY-NODES[i] just before the ith iteration of the main FOR loop. Thus, we can implement Steps (05-10) as follows: Scan through the nodes y in DIRTY-NODES[i] one after another. While considering any such node y, change its level as per Figure 4, and then update the relevant data structures to reflect this change.

The next lemma states the space complexity of this procedure.

Lemma 5.9. *Our algorithm in Figure 4 can be implemented in $\tilde{O}(n)$ space.*

Proof. The amount of space needed is dominated by the number of edges in $\left\{ S_i^{(t)} \right\}, i \in [L-1]$. Since $|S_i^{(t)}| \leq s$ for each $i \in [L-1]$, the space complexity is $(L-1) \cdot s = \tilde{O}(n)$ (see equations 6, 17). $\square$

The claim below bounds the time taken by a single iteration of the main FOR loop in Figure 4. This will be crucially used to analyze the overall update time of our algorithm in Section 5.5.6.

Claim 5.10. *Fix any time-step $t \in [t_0, t_1]$, and consider the ith iteration of the main FOR loop in Figure 4 for some $i \in \{1, \ldots, L-1\}$. Consider 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 for this step 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. Follows from the fact that we only need to update the lists FRIENDS$_{i'}[x, j]$ where $i' > i$, $x \in \{y\} \cup \mathcal{N}_y(Z_i^{(t)}, S_{i'}^{(t)})$ and $j \in \{i, i+1\}$. $\square$

5.5.6 The amortized update time of DYNAMIC-STREAM during a dense time-interval

In this section, we bound the amortized update time of the subroutine DYNAMIC-STREAM during a dense time-interval $[t_0, t_1]$. Recall that the subroutine DYNAMIC-STREAM maintains an $(\alpha, d_k^{(t)}, L)$-decomposition $(V_1^{(t)}, \ldots, V_L^{(t)})$ of the input graph $G^{(t)}$ throughout the duration of such an interval. To bound the amortized update time, we use a potential function $\mathcal{B}$ as defined in equation 24. Note that the potential $\mathcal{B}$ is uniquely determined by the assignment of the nodes $v \in V$ to the levels

$\{1, \ldots, L\}$ and by the contents of the random sets $S_1, \ldots, S_{(L-1)}$. For all nodes $v \in V$, we define:

$$\Gamma_i(v) \;=\; \max(0, (1-\epsilon)^2 \alpha c \log n - D_v(Z_i, S_i)) \text{ for all } i \in \{1, \ldots, \ell(v)-1\} \tag{21}$$

$$\Phi(v) \;=\; (L/\epsilon) \cdot \sum_{i=1}^{\ell(v)-1} \Gamma_i(v) \tag{22}$$

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 $i \in \{1, \ldots, L-1\}$, $(u,v) \in S_i$, we define:

$$\Psi_i(u,v) \;=\; \begin{cases} 0 & \text{if } r_{uv} \geq i; \\ 2 \cdot (i - r_{uv}) + f(u,v) & \text{otherwise.} \end{cases} \tag{23}$$

The potential $\mathcal{B}$ is defined as the sum of the potentials associated with all the nodes and edges.

$$\mathcal{B} \;=\; \sum_{v \in V} \Phi(v) + \sum_{i=1}^{(L-1)} \sum_{e \in S_i} \Psi_i(e) \tag{24}$$

It might be instructive to contrast this potential function with the one used to analyze the dynamic algorithm in Section 4.

Roadmap. Our analysis works in three steps.

1. In Definition 5.11, we describe an event $\mathcal{F}$. To understand the intuition behind this definition, recall the discussion on the third technical challenge in Section 5.1.1: We have to overcome the apparent obstacle that different levels of the $(\alpha, d_k^{(t)}, L)$-decomposition are constructed using different subsets of randomly sampled edges. Intuitively, the event F guarantees that the degrees of a node among these different subsets of edges are approximately the same with high probability. This helps in extending the ideas from the potential function based analysis in Section 4 to the current setting.

2. In Lemma 5.12, we show that the event $\mathcal{F}$ holds with high probability. The proof of Lemma 5.12 appears in Section 5.5.7.

3. Conditioned on the event $\mathcal{F}$, we show that our algorithm has $\tilde{O}(1)$ amortized update time (see Lemma 5.13 and Corollary 5.14). The proof of Lemma 5.13 appears in Section 5.5.8.

Definition 5.11. *Recall the procedure in Figure 4. For all levels $i, i' \in \{1, \ldots, L-1\}$ with $i < i'$, and time-steps $t \in [t_0, t_1]$, we define an $\mathcal{F}_{i,i'}^{(t)}$ as follows.*

- *The event $\mathcal{F}_{i,i'}^{(t)}$ occurs iff the following conditions are satisfied.*

  - $\left\{ D_v(Z_i^{(t)}, S_{i'}^{(t)}) \geq \frac{(1-\epsilon)^4}{(1+\epsilon)^2} \cdot (\alpha c \log n) \text{ for all } v \in A_i^{(t)} \right\}$, *and*
  - $\left\{ D_v(Z_i^{(t)}, S_{i'}^{(t)}) \leq \frac{(1+\epsilon)^4}{(1-\epsilon)^2} \cdot c \log n \text{ for all } v \in B_i^{(t)} \right\}$.

Now, define the event $\mathcal{F}^{(t)} = \bigcap_{i,i'} \mathcal{F}_{i,i'}^{(t)}$ and the event $\mathcal{F} = \bigcap_{t=T'}^{T} \mathcal{F}^{(t)}$.

Lemma 5.12. *The event $\mathcal{F}$ holds with high probability.*

Lemma 5.13. *Conditioned on the event $\mathcal{F}$, we have:*

- *(a) $0 \le \mathcal{B} = \tilde{O}(n)$ at each time-step $t \in [t_0, t_1]$.*

- *(b) Insertion/deletion of an edge in G (ignoring the call to the procedure in Figure 4) changes the potential $\mathcal{B}$ by $\tilde{O}(1)$.*

- *(c) For every constant amount of computation performed while implementing the procedure in Figure 4, the potential $\mathcal{B}$ drops by $\Omega(1)$.*

Corollary 5.14. *With high probability, subroutine* DYNAMIC-STREAM *spends $\tilde{O}(n + (t_1 - t_0))$ time during the dense time-interval $[t_0, t_1]$. So its amortized update time is $\tilde{O}(1)$ with high probability.*

Proof. Condition on the event $\mathcal{F}$ (which occurs with high probability by Lemma 5.12). At each time-step $t \in [1, T]$, we maintain the random sets of edges $\{S_i^{(t)}\}$ as per Theorem 5.3. This takes $\tilde{O}(1)$ worst case update time. Further, a single edge insertion/deletion in the input graph leads to at most two edge insertions/deletions in each of these sets $\{S_i^{(t)}\}, i \in \{1, \dots, L - 1\}$.

Now, using the random sets $\{S_i^{(t)}\}$, at each time-step $t \in [t_0, t_1]$ we maintain an $(\alpha, d_k^{(t)}, L)$-decomposition of the input graph (see Lemma 5.8). Lemma 5.13 implies that with high probability, this requires a total update time of $\tilde{O}(n + (t_1 - t_0))$ for the entire duration of the interval $[t_0, t_1]$.

Finally, recall that either the dense time-interval spans $\tilde{\Omega}(n)$ time-steps, or it ends at time-step T (see the discussion on the subroutine (P3) in Section 5.3). Hence, applying an argument similar to the one used in Section 5.4 (see the discussions preceding equation 11), we conclude that with high probability the subroutine DYNAMIC-STREAM spends $\tilde{O}(T)$ total time during the first T updates in the dynamic stream. In other words, with high probability the subroutine DYNAMIC-STREAM has an amortized update time of $\tilde{O}(1)$. $\square$

5.5.7 Proof of Lemma 5.12

Fix any $1 \le i < i' \le L - 1$, and any time-step $t \in [t_0, t_1]$.

- Condition on an instantiation of the random set $Z_i^{(t)}$. Note that this also determines the sets $A_i^{(t)}$ and $B_i^{(t)}$ (see Figure 4).

  - Let $W_i^{(t)} \subseteq Z_i^{(t)}$ be the subset of nodes v with small degrees $D_v(Z_i^{(t)}, E^{(t)})$. Specifically,

  $$W_i^{(t)} = \left\{ v \in Z_i^{(t)} : D_v(Z_i^{(t)}, E^{(t)}) < \frac{(1 - \epsilon)^2}{(1 + \epsilon)^2} \cdot \alpha d_k^{(t)} \right\} \tag{25}$$

  The node set $W_i^{(t)}$ is uniquely determined by the node set $Z_i^{(t)}$ (which we are conditioning upon) and the edge set $E^{(t)}$ (which is given by the stream of updates in the input graph).

  Now, consider any node $v \in W_i^{(t)}$. By Lemma 5.7, the contents of the random set $S_i^{(t)}$ are independent of $Z_i^{(t)}$. By Theorem 5.3, each edge $(u, v) \in E^{(t)}$ is included in $S_i^{(t)}$ with probability $(1 \pm \epsilon) \cdot (s/m^{(t)})$. Applying Linearity of expectation and equation 16, we get:

  $$E\left[D_v(Z_i^{(t)}, S_i^{(t)}) \right] \le (1 + \epsilon) \cdot (c \log n / d_k^{(t)}) \cdot D_v(Z_i^{(t)}, E^{(t)}) \tag{26}$$

  $$\le \frac{(1 - \epsilon)^2}{(1 + \epsilon)} \cdot (\alpha c \log n) \tag{27}$$

  Equation 27 follows from equations 25 and 26. Next, for each edge $(u, v) \in E^{(t)}$ incident upon the node v, consider the random event that $(u, v) \in S_i^{(t)}$. By Theorem 5.3, these

random events are negatively associated (see Section 2.4). Thus, applying Chernoff bound on equation 27 and recalling the definition of $A_i^{(t)}$ from Figure 4, we get:

$$\Pr\left[v \in A_i^{(t)}\right] = \Pr\left[D_v(Z_i^{(t)}, S_i^{(t)}) > (1-\epsilon)^2 \cdot (\alpha c \log n)\right] \leq 1/(\text{poly } n) \tag{28}$$

Applying a union bound on equation 28 over all the nodes in $W_i^{(t)}$, we get:

$$\Pr\left[W_i^{(t)} \bigcap A_i^{(t)} \neq \emptyset\right] \leq 1/(\text{poly } n) \tag{29}$$

In other words, with high probability no node in $W_i^{(t)}$ belongs to the set $A_i^{(t)}$.

We will now bound the degrees of the nodes in $Z_i^{(t)} \setminus W_i^{(t)}$ with respect to the random edge set $S_{i'}^{(t)}$. Towards this end, consider any node $x \in Z_i^{(t)} \setminus W_i^{(t)}$. By Lemma 5.7, the contents of the random set $S_{i'}^{(t)}$ are independent of $Z_i^{(t)}$. By Theorem 5.3, each edge $(u, v) \in E^{(t)}$ is included in $S_{i'}^{(t)}$ with probability $(1 \pm \epsilon) \cdot (s/m^{(t)})$. Applying Linearity of expectation and equations 16, 25 we get:

$$E\left[D_x(Z_i^{(t)}, S_{i'}^{(t)})\right] \geq (1-\epsilon) \cdot (c \log n / d_k^{(t)}) \cdot D_v(Z_i^{(t)}, E^{(t)}) \tag{30}$$

$$\geq \frac{(1-\epsilon)^3}{(1+\epsilon)^2} \cdot (\alpha c \log n) \tag{31}$$

Equation 31 follows from equations 25 and 30. Next, for each edge $(u, x) \in E^{(t)}$ incident upon the node x, consider the random event that $(u, x) \in S_{i'}^{(t)}$. By Theorem 5.3, these random events are negatively associated (see Section 2.4). Thus, applying Chernoff bound on equation 31, we get:

$$\Pr\left[D_x(Z_i^{(t)}, S_{i'}^{(t)}) < \frac{(1-\epsilon)^4}{(1+\epsilon)^2} \cdot (\alpha c \log n)\right] \leq 1/(\text{poly } n) \tag{32}$$

Now, taking an union bound on equation 32 over all the nodes in $Z_i^{(t)} \setminus W_i^{(t)}$, we get:

$$\Pr\left[D_x(Z_i^{(t)}, S_{i'}^{(t)}) < \frac{(1-\epsilon)^4}{(1+\epsilon)^2} \cdot (\alpha c \log n) \text{ for some } x \in Z_i^{(t)} \setminus W_i^{(t)}\right] \leq 1/(\text{poly } n) \tag{33}$$

In other words, with high probability every node $x \in Z_i^{(t)} \setminus W_i^{(t)}$ has a high degree $D_x(Z_i^{(t)}, S_{i'}^{(t)})$. Now, taking an union bound over equations 29 and 33, we conclude that:

With high probability, $D_x(Z_i^{(t)}, S_{i'}^{(t)}) \geq \frac{(1-\epsilon)^4}{(1+\epsilon)^2} \cdot (\alpha c \log n)$ for every node $x \in A_i^{(t)}$. (34)

Using a similar argument for the node set $B_i^{(t)}$, we can infer that:

With high probability, $D_x(Z_i^{(t)}, S_{i'}^{(t)}) \leq \frac{(1+\epsilon)^4}{(1-\epsilon)^2} \cdot (c \log n)$ for every node $x \in B_i^{(t)}$. (35)

Taking an union bound over equations 34 and 35, we infer that:

Given any instantiation of $Z_i^{(t)}$, the event $\mathcal{F}_{i,i'}^{(t)}$ occurs with high probability. (36)

- From equation 36, we infer that:

The event $\mathcal{F}_{i,i'}^{(t)}$ occurs with high probability. (37)

The lemma follows by applying an union bound on equation 37 over all indices $i, i' \in \{1, \ldots, L-1\}$ with $i < i'$ and time-steps $t \in [t_0, t_1]$.

5.5.8 Proof of Lemma 5.13

Part (a) and part (b) of the lemma hold independently of the event $\mathcal{F}$. It is only during the proof of part (c) that we have to condition of the event $\mathcal{F}$.

Proof of part (a). Fix any time-step $t \in [t_0, t_1]$. The proof follows from three facts.

1. We have $0 \leq \Phi(v) \leq (L/\epsilon) \cdot L \cdot (1 - \epsilon)^2 \alpha c \log n = \tilde{O}(1)$ for all $v \in V$.

2. We have $0 \leq \Psi_i(u, v) \leq 3L = \tilde{O}(1)$ for all $i \in [L - 1], (u, v) \in S_i^{(t)}$.

3. We have $|S_i^{(t)}| \leq s = \tilde{O}(n)$ for all $i \in [L - 1]$. This follows from equation 17.

Proof of part (b). By Theorem 5.3, insertion/deletion of an edge in G leads to at most two insertions/deletions in each of the random sets $S_1^{(t)}, \ldots, S_{L-1}^{(t)} \subseteq E^{(t)}$. Since $L = \tilde{O}(1)$ (see equation 6), it suffices to show that for a single edge insertion/deletion in any given $S_i^{(t)}$, the potential $\mathcal{B}$ changes by at most $\tilde{O}(1)$ (ignoring the call to the procedure in Figure 4).

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 endpoint $x \in \{u, v\}$, this changes the potential $\Phi(x)$ by at most $O(L/\epsilon)$. The potentials $\Phi(y)$ for all other nodes $y \in V \setminus \{u, v\}$ remain unchanged. Additionally, the potential $\Psi_i(u, v) \in [0, 3L]$ is created (resp. destroyed). Thus, we infer that the absolute value of the change in the overall potential $\mathcal{B}$ is at most $O(3L + 2L/\epsilon) = \tilde{O}(1)$.

Proof of part (c). Fix any time-step $t \in [t_0, t_1]$, and any iteration of the FOR loop in Figure 4 while processing the update in time-step t. Consider two possible events.

Case 1: A node $v \in Z_i^{(t)}$ is promoted from level i to level $(i + 1)$ in Step 07 of Figure 4. This happens only if $v \in A_i^{(t)}$. Let C be the amount of computation performed during this step. By Claim 5.10, we have:

$$C = \sum_{i'=(i+1)}^{(L-1)} O\left(1 + D_v(Z_i^{(t)}, S_{i'}^{(t)})\right) \tag{38}$$

Let Δ be the net decrease in the overall potential $\mathcal{B}$ due to this step. We observe that:

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 (this holds since the degree $D_u(Z_j, S_j)$, for any level j, can only increase as node v increases its level from i to $i + 1$).

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

$$\Delta \geq \sum_{i'=(i+1)}^{(L-1)} D_v(Z_i^{(t)}, S_{i'}^{(t)}) \tag{39}$$

Since $v \in A_i^{(t)}$, and since we have conditioned on the event $\mathcal{F}^{(t)}$ (see Definition 5.11), we get:

$$D_v(Z_i^{(t)}, S_{i'}^{(t)}) \quad > \quad 0 \quad \text{for all } i' \in [i+1, L-1]. \tag{40}$$

Eq. (38), (39), (40) 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 to level i in Steps (08-09) of Figure 4.
This can happen only if $j > i$ and $v \in B_i^{(t)}$. Let C denote the amount of computation performed during this step. By Claim 5.10, we have

$$C \quad = \quad \sum_{i'=(i+1)}^{(L-1)} O(1 + D_v(Z_i^{(t)}, S_{i'}^{(t)})) \tag{41}$$

Let $\gamma = (1+\epsilon)^4/(1-\epsilon)^2$. Equation (42) holds since $v \in B_i^{(t)}$ and since we conditioned on the event $\mathcal{F}$ (see Definition 5.11). Equation (43) follows from equations (41), (42) and since γ, c are constants.

$$D_v(Z_i^{(t)}, S_{i'}^{(t)}) \quad \leq \quad \gamma c \log n \text{ for all } i' \in [i, L-1] \tag{42}$$
$$C \quad = \quad O(L \log n) \tag{43}$$

Let Δ be the net decrease in the overall potential $\mathcal{B}$ due to this step. We observe that:

1. By eq. (42), 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 (42), 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 (42), 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:

$$
\begin{aligned}
\Delta \quad &\geq \quad (j-i)(L/\epsilon)((1-\epsilon)^2 \alpha - \gamma)(c \log n) \\
&\quad -(j-i)(L/\epsilon)(\gamma c \log n) - (j-i)(3L)(\gamma c \log n) \\
&= \quad (j-i) \cdot (L/\epsilon) \cdot ((1-\epsilon)^2 \alpha - 2\gamma - 3\epsilon\gamma) \cdot (c \log n) \\
&\geq \quad Lc \log n
\end{aligned}
\tag{44}
$$

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. (43) and (44), we conclude that the net decrease in the overall potential $\mathcal{B}$ is sufficient to pay for the cost of the computation performed.

6 Extension to Directed Graphs

In this section, we extend our dynamic algorithm from Section 4 to directed graphs. The notion of "density" of a directed graph was introduced by Kannan et al. [25]. We first recall their definition.

Definition 6.1. *[25] Consider two subsets of nodes $X, Y \subseteq V$ in a directed graph $G = (V, E)$. The "density" of this pair equals $\rho(X, Y) = |E(X, Y)|/\sqrt{|X||Y|}$. Here, $E(X, Y) = \{(u, v) : u \in X, v \in Y\}$ is the set of edges going from X to Y. The value of the densest subgraph is given by $\rho(G) = \max_{X,Y \subseteq V} \rho(X, Y)$. Note that we do not require the sets X and Y to be mutually disjoint.*

Throughout this section, we denote the input graph by $G = (V, E)$. In the beginning, the input graph is empty, i.e., we have $E = \emptyset$. Subsequently, at each time-step, either a directed edge is inserted into the graph, or an already existing edge is deleted from the graph. The set of nodes, on the other hand, remains unchanged. Our goal is to maintain a good approximation of $\rho(G)$ in this dynamic setting. The main result is stated below.

Theorem 6.2. *We can deterministically maintain a $(8 + \epsilon)$-approximation to the value of the densest subgraph of a directed graph $G = (V, E)$. The algorithm requires $\tilde{O}(m + n)$-space, where m (resp. n) denotes the number of nodes (resp. edges) in the graph. Furthermore, the algorithm has an amortized update time of $\tilde{O}(1)$ and a query time of $O(1)$.*

We devote the rest of Section 6 to the proof of Theorem 6.2. We first define the preliminary concepts and notations in Section 6.1. Next, we extend the notion of an (α, d, L)-decomposition to directed graphs in Section 6.2. In Section 6.3, we present our main algorithm. Finally, in Section 6.4, we combine all these ingredients together and conclude the proof of Theorem 6.2.

6.1 Notations and Preliminaries

We first define the notion of a "*derived graph*", which will be crucially used in our algorithm.

Definition 6.3. *The "derived graph" $G' = (V', E')$ of the input graph $G = (V, E)$ is constructed as follows. For each node $v \in V$, we create two nodes s_v and t_v. We define the node set $V' = S' \cup T'$, where $S' = \{s_v : v \in V\}$ and $T' = \{t_v : v \in V\}$. Next, for each directed edge $(u, v) \in E$, we create the directed edge (s_u, t_v), and define the directed edge set $E' = \{(s_u, t_v) : (u, v) \in E\}$. Thus, in the derived graph $G' = (V', E')$, each node in S' (resp. T') has zero in-degree (resp. out-degree).*

It is easy to check that the derived graph $G' = (V', E')$ can be maintained in the dynamic setting using $O(m + n)$ space and $O(1)$ update time: Fix the set of nodes $V' = S' \cup T'$, and whenever an edge (u, v) is inserted into (resp. deleted from) $G = (V, E)$, insert (resp. delete) the corresponding derived edge (s_u, t_v) in G'. From now on, unless explicitly mentioned otherwise, our main algorithm will work on the derived graph $G' = (V', E')$. Before proceeding any further, we introduce some notations that will be used throughout the rest of this section.

- Consider the derived graph $G' = (V', E')$. Given any node $s \in S'$ and any subset of nodes $T \subseteq T'$, we let $\mathcal{N}_s(T) = \{t \in T : (s, t) \in E'\}$ denote the set of neighbors of s among the nodes in T. Furthermore, we let $D_s(T) = |\mathcal{N}_s(T)|$ denote the degree of s among the nodes in T. For a node $t \in T'$ and a subset $S \subseteq S'$, the notations $\mathcal{N}_t(S) = \{(s, t) \in E' : s \in S\}$ and $D_t(S) = |\mathcal{N}_t(S)|$ are defined analogously. Next, for any two subsets of nodes $S \subseteq S'$ and $T \subseteq T'$, we let $E'(S, T) = \{(s, t) \in E' : s \in S, t \in T\}$ denote the set of edges in the derived graph that are going from S to T. We also define $\rho'(S, T) = |E'(S, T)|/\sqrt{|S||T|}$ for all $S \subseteq S', T \subseteq T'$. Hence, from Definition 6.1, it follows that $\rho'(S, T) = \rho(S, T)$.

Accordingly, the value of the densest subgraph of the input graph $G = (V, E)$ is given by $\rho(G) = \max_{S \subseteq S', T \subseteq T'} \rho'(S, T)$. We will denote the densest subgraph of $G = (V, E)$ by the pair (S^*, T^*), where $S^* \subseteq S'$ and $T^* \subseteq T'$. Thus, we have $\rho'(S^*, T^*) = \rho(G)$. Finally, we define the parameters $\lambda_{S'}, \lambda_{T'}$ as follows.

$$\lambda_{S'} = |E(S^*, T^*)| \cdot (1 - \sqrt{1 - 1/|S^*|}) \text{ and } \lambda_{T'} = |E(S^*, T^*)| \cdot (1 - \sqrt{1 - 1/|T^*|}). \quad (45)$$

We now state one crucial lemma that will be used in the analysis of our algorithm. This lemma was proved by Khuller et al. [29]. For the sake of completeness, we state their proof below.

Lemma 6.4. *[29] Consider the densest subgraph (S^*, T^*) of the input graph $G = (V, E)$. We have:*

1. *$D_s(T^*) \geq \lambda_{S'}$ for all nodes $s \in S^*$, and $D_t(S^*) \geq \lambda_{T'}$ for all nodes $t \in T^*$.*

2. *$\rho(G)/2 \leq \sqrt{\lambda_{S'} \lambda_{T'}} \leq \rho(G)$.*

Proof.

1. For the sake of contradiction, suppose that there exists a node $s \in S^*$ with $D_s(T^*) < \lambda_{S'}$. Then we can show that the density of the pair $(S^* \setminus \{s\}, T^*)$ is strictly larger than the density of the pair (S^*, T^*). Specifically, we can infer the following guarantee:

$$\rho'(S^* \setminus \{s\}, T^*) > \frac{|E'(S^*, T^*)| - \lambda_{S'}}{\sqrt{(|S^*| - 1) \cdot |T^*|}} = \frac{|E'(S^*, T^*)|}{\sqrt{|S^*| \cdot |T^*|}} = \rho'(S^*, T^*) = \rho(G).$$

Since $\rho(G)$ denotes the value of the densest subgraph of G, the above inequality leads to a contradiction. Thus, we conclude that $D_s(T^*) \geq \lambda_{S'}$ for all nodes $s \in S^*$. Using the same line of reasoning, we can conclude that $D_t(S^*) \geq \lambda_{T'}$ for all nodes $t \in T^*$.

2. From part I of the lemma, we have the following guarantee: Every node in S^* (resp. T^*) has a degree of at least $\lambda_{S'}$ (resp. $\lambda_{T'}$) among the nodes in T^* (resp. S^*). This implies that $|E'(S^*, T^*)| \geq |S^*| \cdot \lambda_{S'}$ and $|E'(S^*, T^*) \geq |T^*| \cdot \lambda_{T'}$. Thus, we get:

$$|E'(S^*, T^*)| \geq \sqrt{|S^*| \cdot |T^*|} \cdot \sqrt{\lambda_{S'} \cdot \lambda_{T'}} \quad (46)$$

Since $\rho(G) = \rho'(S^*, T^*) = |E'(S^*, T^*)|/\sqrt{|S^*| \cdot |T^*|}$, from equation 46 we conclude that:

$$\rho(G) = \frac{|E'(S^*, T^*)|}{\sqrt{|S^*| \cdot |T^*|}} \geq \sqrt{\lambda_{S'} \cdot \lambda_{T'}} \quad (47)$$

Next, putting $|S^*| = 1/\sin^2 \theta_1$ and $|T^*| = 1/\sin^2 \theta_2$, and recalling equation 45, we get:

$$
\begin{aligned}
\lambda_{S'} \cdot \lambda_{T'} &= \frac{|E'(S^*, T^*)|^2}{|S^*| \cdot |T^*|} \cdot (|S^*| \cdot |T^*|) \cdot \left(1 - \sqrt{1 - 1/|S^*|}\right) \cdot \left(1 - \sqrt{1 - 1/|T^*|}\right) \\
&= \left(\rho'(S^*, T^*)\right)^2 \cdot \frac{(1 - \cos \theta_1) \cdot (1 - \cos \theta_2)}{\sin^2 \theta_1 \sin^2 \theta_2} \\
&= \frac{\left(\rho'(S^*, T^*)\right)^2}{4 \cos^2(\theta_1/2) \cos^2(\theta_2/2)} \\
&\geq \frac{\left(\rho'(S^*, T^*)\right)^2}{4} = \frac{(\rho(G))^2}{4} \quad (48)
\end{aligned}
$$

From equation 48 we conclude that:

$$\sqrt{\lambda_{S'} \cdot \lambda_{T'}} \geq \rho(G)/2 \quad (49)$$

The part II of the lemma follows from equations 47 and 49.

$\square$

6.2 $(\alpha, d_{S'}, d_{T'}, L)$-Decomposition and Its Properties

We extend the concept of an (α, d, L)-decomposition (see Definition 2.5) to directed graphs. This requires us to introduce one additional parameter in the definition, and we call the corresponding entity an $(\alpha, d_{S'}, d_{T'}, L)$-decomposition. Specifically, an $(\alpha, d_{S'}, d_{T'}, L)$-decomposition of the derived graph $G' = (V', E')$, where $V' = S' \cup T'$, is given by two laminar families of subsets of nodes $S' = S_1 \supseteq S_2 \supseteq \cdots \supseteq S_L$ and $T' = T_1 \supseteq T_2 \supseteq \cdots \supseteq T_L$. These subsets are iteratively constructed as follows. First, we set $S_1 = S$ and $T_1 = T$. Next, for each $i \in \{1, \ldots, L-1\}$, while constructing the subsets S_{i+1}, T_{i+1} from the subsets S_i, T_i, we ensure the following conditions:

- Every node $s \in S_i$ with $D_s(T_i) > \alpha \cdot d_{S'}$ is included in the subset S_{i+1}. On the other hand, every node $s \in S_i$ with $D_s(T_i) < d_{S'}$ is excluded from the subset S_{i+1}.

- Every node $t \in T_i$ with $D_t(S_i) > \alpha \cdot d_{T'}$ is included in the subset T_{i+1}. On the other hand, every node $t \in T_i$ with $D_t(S_i) < d_{T'}$ is excluded from the subset T_{i+1}.

We are now ready to formally define the concept of an $(\alpha, d_{S'}, d_{T'}, L)$-decomposition of G'.

Definition 6.5. *Fix any $\alpha \geq 1$, $d_{S'}, d_{T'} \geq 0$, and any positive integer L. Consider a family of subsets $S' = S_1 \supseteq \cdots \supseteq S_L$ and $T' = T_1 \supseteq \cdots \supseteq T_L$. The tuples $(S_1, \ldots, S_L)$ and $(T_1, \ldots, T_L)$ form an $(\alpha, d_{S'}, d_{T'}, L)$-decomposition of the derived graph G' iff for every $i \in \{1, \ldots, L-1\}$, we have:*

1. $S_{i+1} \supseteq \{s \in S_i : D_s(T_i) > \alpha \cdot d_{S'}\}$ and $S_{i+1} \cap \{s \in S_i : D_s(T_i) < d_{S'}\} = \emptyset$.

2. $T_{i+1} \supseteq \{t \in T_i : D_t(S_i) > \alpha \cdot d_{T'}\}$ and $T_{i+1} \cap \{t \in T_i : D_t(S_i) < d_{T'}\} = \emptyset$.

Let $V_i' = (S_i \cup T_i) \setminus (S_{i+1} \cup T_{i+1})$ for all $i \in \{1, \ldots, L-1\}$, and $V_i' = S_i \cup T_i$ for $i = L$. We say that the nodes in V_i' constitute the ith 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 $v \in V_i'$, and the set of nodes V' is partitioned into L subsets $V_1', \ldots, V_L'$.

Theorem 6.6 and Corollary 6.7 will play the main role in the rest of this section. Roughly speaking, they are generalizations of Theorem 2.6 and Corollary 2.7 from Section 2.2, and they state that we can use the $(\alpha, d_{S'}, d_{T'}, L)$-decomposition to $4\alpha(1+\epsilon)^{3/2}$-approximate of the value of the densest subgraph of $G = (V, E)$. All we need to do is to set $L = O(\log n/\epsilon)$ and try different values of $d_{S'}, d_{T'}$ in powers of $(1+\epsilon)$.

Theorem 6.6. *Fix any $\alpha \geq 1$, $d_{S'}, d_{T'} \geq 0$, $\epsilon \in (0, 1)$, and $L \leftarrow 2 \cdot (2 + \lceil \log_{(1+\epsilon)} n \rceil)$. Let $(S_1, \ldots, S_L), (T_1, \ldots, T_L)$ be an $(\alpha, d_{S'}, d_{T'}, L)$-decomposition of the derived graph $G' = (V', E')$ as per Definition 6.5. Then the following conditions are satisfied.*

1. If $d_{S'} d_{T'} > 4(1+\epsilon)\lambda_{S'}\lambda_{T'}$, then $V_L' = \emptyset$ (i.e., the topmost level of the decomposition is empty).

2. If $d_{S'} < \lambda_{S'}/\alpha$ and $d_{T'} < \lambda_{T'}/\alpha$, then $V_L' \neq \emptyset$ (i.e., the topmost level is nonempty).

Proof.

1. For any level $k \in \{2, \ldots, L\}$. Definition 6.5 states that for every node $s \in S_k$, we have $D_s(T_{k-1}) > d_{S'}$. We thus get a lower bound on the number edges going from S_k to T_{k-1}.

$$|E'(S_k, T_{k-1})| \geq |S_k| \cdot d_{S'} \tag{50}$$

Recall that by Lemma 6.4, we have $\rho(G) \leq 2 \cdot \sqrt{\lambda_{S'} \lambda_{T'}}$. This implies that $\rho'(S_k, T_{k-1}) = |E'(S_k, T_{k-1})|/\sqrt{|S_k||T_{k-1}|} \leq \rho(G) \leq 2 \cdot \sqrt{\lambda_{S'} \lambda_{T'}} \leq \sqrt{d_{S'} d_{T'}/(1+\epsilon)}$. Hence, we get:

$$|E'(S_k, T_{k-1})| \leq \sqrt{\frac{|S_k| \cdot |T_{k-1}| \cdot |d_{S'}| \cdot |d_{T'}|}{(1+\epsilon)}} \tag{51}$$

From equations 50 and 51, we get:

$$|S_k| \cdot d_{S'} \leq |T_{k-1}| \cdot d_{T'}/(1+\epsilon) \tag{52}$$

Replacing S by T in the above argument, we can analogously show that:

$$|T_k| \cdot d_{T'} \leq |S_{k-1}| \cdot d_{S'}/(1+\epsilon) \tag{53}$$

Now, we consider two possible cases.

- If $|S_{k-1}| \cdot d_{S'} \geq |T_{k-1}| \cdot d_{T'}$, then equation 52 implies that $|S_k| \leq |S_{k-1}|/(1+\epsilon)$.
- Else if $|S_{k-1}| \cdot d_{S'} < |T_{k-1}| \cdot d_{T'}$, then equation 53 implies that $|T_k| < |T_{k-1}|/(1+\epsilon)$.

In other words, when going from level $k-1$ to level k, either the size of S or the size of T reduces by a factor of $(1+\epsilon)$. Since $|S_1| = |T_1| = n$ and $L = 2 \cdot (2 + \lceil \log_{(1+\epsilon)} n \rceil)$, after L levels both the sets S and T are empty. We thus have $S_L = T_L = \emptyset$.

2. Consider the densest subgraph of the input graph $G = (V, E)$, given by the pair (S^*, T^*) where $S^* \subseteq S'$ and $T^* \subseteq T'$. Lemma 6.4 states that each node $s \in S^*$ (resp. $t \in T^*$) has out-degree (resp. in-degree) at least $\lambda_{S'}$ (resp. $\lambda_{T'}$) with respect to the nodes in T^* (resp. S^*). Since $\lambda_{S'} > \alpha \cdot d_{S'}$ and $\lambda_{T'} > \alpha \cdot d_{T'}$, Definition 6.5 implies the following guarantee:

   - If $S^* \subseteq S_i$, $T^* \subseteq T_i$ for some $i \in \{1, \ldots, L-1\}$, then we also have $S^* \subseteq S_{i+1}$, $T^* \subseteq T_{i+1}$.

   Since $S_1 = S'$, $T_1 = T'$, we have $S^* \subseteq S_1$ and $T^* \subseteq T_1$. Hence, applying the above guarantee in an inductive fashion, we conclude that $S^* \subseteq S_L$ and $T^* \subseteq T_L$. Thus, $V'_L = S_L \cup T_L \neq \emptyset$.

$\square$

Corollary 6.7. *As in Theorem 6.6, fix any $\alpha \geq 1$, $\epsilon \in (0,1)$ and $L = 2 \cdot (2 + \lceil \log_{(1+\epsilon)} n \rceil)$. Define $\lambda^* = 1 - \sqrt{1 - 1/n}$. Discretize the range $[\lambda^*/(\alpha(1+\epsilon)), n^2]$ in powers of $(1+\epsilon)$, by letting $q_k = (1+\epsilon)^{k-1} \lambda^*/\alpha$ for every integer $k \geq 0$. Let K be the smallest integer k for which $q_k \geq n^2$. For all $d_{S'}, d_{T'} \in \{d_0, \ldots, d_K\}$, construct an $(\alpha, d_{S'}, d_{T'}, L)$-decomposition of G' as per Definition 6.5, and let $V'_L(d_{S'}, d_{T'})$ denote the set of nodes at level L of such a decomposition. Let $P = \{(d_{S'}, d_{T'}) : d_{S'}, d_{T'} \in \{0, \ldots, K\}, V'_L(d_{S'}, d_{T'}) \neq \emptyset\}$ denote the set of those $(d_{S'}, d_{T'})$ pairs for which the topmost level of the decomposition is non-empty. If $P = \emptyset$, then define $\gamma = 0$. Else define $\gamma = \max_{(d_{S'}, d_{T'}) \in P} \{d_{S'} \cdot d_{T'}\}$. Then we have:*

$$2 \cdot \sqrt{(1+\epsilon)} \cdot \rho(G) \geq \sqrt{\gamma} \geq \frac{\rho(G)}{2 \cdot \alpha \cdot (1+\epsilon)}.$$

Proof. If the derived graph is empty, i.e., if $E' = \emptyset$, then we have $E = \emptyset$, $P = \emptyset$, $\gamma = 0$, and $\rho(G) = 0$. Thus, in this case the corollary is trivially true.

For the remainder of the proof, we assume that $E' \neq \emptyset$. Here, it is easy to see that $P \neq \emptyset$: simply consider the $(\alpha, d_{S'}, d_{T'}, L)$-decomposition with $d_{S'} = d_{T'} = q_0 < 1$. In this decomposition, every node in G' with nonzero degree will be promoted to the topmost level $V'_L(d_{S'}, d_{T'})$.

Accordingly, for the rest of the proof, we fix a $(d_{S'}, d_{T'})$-pair in P for which $d_{S'} \cdot d_{T'} = \gamma$. Let this pair be identified as $(d_{S'}^*, d_{T'}^*)$. Since $(d_{S'}^*, d_{T'}^*) \in P$, we infer that $V_L'(d_{S'}^*, d_{T'}^*) \neq \emptyset$. Hence, by the first part of Theorem 6.6, we must have:

$$\gamma = d_{S'}^* \cdot d_{T'}^* \leq 4(1 + \epsilon) \cdot \lambda_{S'} \cdot \lambda_{T'}. \tag{54}$$

Next, define $\hat{d}_{S'}$ (resp. $\hat{d}_{T'}$) to be the maximum value of $d_{S'}$ (resp. $d_{T'}$) which is less than the threshold $\lambda_{S'}/\alpha$ (resp. $\lambda_{T'}/\alpha$). Note that since $E' \neq \emptyset$, we have $\lambda^* \leq \lambda_{S'}, \lambda_{T'} < n^2$. Thus, we are guaranteed the existence of such a pair $(\hat{d}_{S'}, \hat{d}_{T'})$.

$$\hat{d}_{S'} = \max_{k \in \{0, \ldots, K\} : q_k < \lambda_{S'}/\alpha} \{q_k\} \text{ and } \hat{d}_{T'} = \max_{k \in \{0, \ldots, K\} : q_k < \lambda_{T'}/\alpha} \{q_k\} \tag{55}$$

Next, note that $q_0 < \lambda^*/\alpha$ and $q_K \geq n^2$. Since the consecutive q_k values are within a factor of $(1 + \epsilon)$ from each other, equation 55 gives us:

$$\hat{d}_{S'} \cdot \hat{d}_{T'} \geq \frac{\lambda_{S'} \cdot \lambda_{T'}}{\alpha^2 \cdot (1 + \epsilon)^2} \tag{56}$$

Since $\hat{d}_{S'} < \lambda_{S'}/\alpha$ and $\hat{d}_{T'} < \lambda_{T'}/\alpha$, Theorem 6.6 (part II) implies that $V_L'(\hat{d}_{S'}, \hat{d}_{T'}) \neq \emptyset$. Thus, we have $(\hat{d}_{S'}, \hat{d}_{T'}) \in P$. Since $(d_{S'}^*, d_{T'}^*) \in P$ maximizes the product of its two components, we get:

$$\gamma = d_{S'}^* \cdot d_{T'}^* \geq \hat{d}_{S'} \cdot \hat{d}_{T'} \tag{57}$$

From equations 54, 56 and 57, we infer that:

$$\frac{\lambda_{S'} \cdot \lambda_{T'}}{\alpha^2 \cdot (1 + \epsilon)^2} \leq \gamma \leq 4(1 + \epsilon) \cdot \lambda_{S'} \cdot \lambda_{T'} \tag{58}$$

By the second part of Lemma 6.4, we have: $\rho(G)/2 \leq \sqrt{\lambda_{S'} \lambda_{T'}} \leq \rho(G)$. Combining this observation with equation 58, we get:

$$\frac{\rho(G)}{2\alpha(1 + \epsilon)} \leq \sqrt{\gamma} \leq 2 \cdot \sqrt{(1 + \epsilon)} \cdot \rho(G) \tag{59}$$

This concludes the proof of the corollary. $\qquad\square$

6.3 The Algorithm for Maintaining an $(\alpha, d_{S'}, d_{T'}, L)$-Decomposition

Throughout this section, we fix the values of $d_{S'}, d_{T'}, L$. Furthermore, we fix an $\alpha \geq 2 + \epsilon$. We describe an algorithm for maintaining an $(\alpha, d_{S'}, d_{T'}, L)$-decomposition of the derived graph $G' = (V', E')$ in a dynamic setting (see Definition 6.3). We assume that the input graph $G = (V, E)$ is empty in the beginning, and hence, at that instant we also have $E' = \emptyset$. Subsequently, at each time-step, a directed edge (u, v) is inserted into (resp. deleted from) the graph $G = (V, E)$, and accordingly, the edge (s_u, t_v) is inserted into (resp. deleted from) the derived graph $G' = (V', E')$. Our main result is stated below.

Theorem 6.8. *For every polynomially bounded $\alpha \geq 2 + 3\epsilon$, we can deterministically maintain an $(\alpha, d_{S'}, d_{T'}, L)$-decomposition of the derived graph $G' = (V', E')$. Starting from an empty graph, we can handle a sequence of t update operations (edge insertions/deletions) in total time $O(tL/\epsilon)$. Thus, we get an amortized update time of $O(L/\epsilon)$. 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 derived graph at that time-step, and $n = |V'|$ denotes the number of nodes in the derived graph (which does not change over time). Note that $|V| = 2n$ and $|E| = m$, where $G = (V, E)$ is the input graph.*

The proof of Theorem 6.8 is very similar to the proof of Theorem 4.2 from Section 4.1. Nevertheless, for the sake of completeness, we highlight the main parts of the algorithm and its analysis.

Data Structures. Recall the concept of the level of a node from Definition 6.5. We now separately describe the data structures associated with the nodes in S' and T'.

- Every node $s \in S'$ maintains L lists $\text{FRIENDS}_i[s]$, for $i \in \{1, \ldots, L\}$. For $i < \ell(s)$, the list $\text{FRIENDS}_i[s]$ consists of the neighbors of s that are at level i: these are nodes belonging to the set $\mathcal{N}_s(V'_i \cap T_i)$. For $i = \ell(s)$, the set $\text{FRIENDS}_i[s]$ consists of the neighbors of s that are at level i or above: these are the nodes belonging to the set $\mathcal{N}_s(T_i)$. For $i > \ell(s)$, the list $\text{FRIENDS}_i[s]$ is empty. Each list is stored in a doubly linked list together with its size, $\text{COUNT}_i[s]$. Using appropriate pointers, we can insert or delete a given node to or from a concerned list in constant time. The counter $\text{LEVEL}[s]$ keeps track of the level of the node s.

- Analogously, every node $t \in T'$ maintains L lists $\text{FRIENDS}_i[t]$, for $i \in \{1, \ldots, L\}$. For $i < \ell(t)$, the list $\text{FRIENDS}_i[t]$ consists of the neighbors of t that are at level i: these are nodes belonging to the set $\mathcal{N}_t(V'_i \cap S_i)$. For $i = \ell(t)$, the set $\text{FRIENDS}_i[t]$ consists of the neighbors of t that are at level i or above: these are the nodes belonging to the set $\mathcal{N}_t(S_i)$. For $i > \ell(t)$, the list $\text{FRIENDS}_i[t]$ is empty. Each list is stored in a doubly linked list together with its size, $\text{COUNT}_i[t]$. Using appropriate pointers, we can insert or delete a given node to or from a concerned list in constant time. The counter $\text{LEVEL}[t]$ keeps track of the level of the node t.

The Algorithm. If a node violates one of the conditions of an $(\alpha, d_{S'}, d_{T'}, L)$-decomposition (see Definition 6.5), then we call the node "dirty", else the node is called "clean". Specifically, consider two possible cases depending on the type of the node.

- A node $s \in S'$ at level $\ell(s) = i$ is dirty iff either (a) $i < L$ and $D_s(T_i) > \alpha \cdot d_{S'}$, or (b) $i > 1$ and $D_s(T_{i-1}) < d_{S'}$.

- A node $t \in T'$ at level $\ell(t) = i$ is dirty iff either (a) $i < L$ and $D_t(S_i) > \alpha \cdot d_{T'}$, or (b) $i > 1$ and $D_t(S_{i-1}) < d_{T'}$.

Initially, the derived graph $G' = (V', E')$ is empty, every node is at level 1, and every node is clean. When an edge (s, t), $s \in S', t \in T'$, is inserted/deleted in the derived graph G', we first update the FRIENDS lists of s and t by adding or removing that edge in constant time. Next we check whether s or t becomes dirty due to this edge insertion/deletion. If yes, then we run the RECOVER-DIRECTED() procedure described in Figure 5. Note that a single iteration of the WHILE loop (Steps 01-15) 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.

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

Analysis of the Update Time. Handling each edge insertion/deletion takes constant time plus the time for the RECOVER-DIRECTED() procedure. We show below that the total time spent in procedure RECOVER-DIRECTED() during t update operations is $O(tL/\epsilon)$.

Potential Function. To determine the amortized update time we use a potential function $\mathcal{B}$ that depends on the state of the $(\alpha, d_{S'}, d_{T'}, L)$-decomposition. For any two nodes $s \in S', t \in T'$, let $f(s, t) = 1$ if $l(s) = l(t)$ and 0 otherwise. We define $\mathcal{B}$, the node potentials $\Phi(x)$ (for each $x \in V' = S' \cup T'$), and the edge potentials $\Psi(s, t)$ (for each $(s, t) \in E'$) as follows.

```
01.      WHILE there exists a dirty node $y \in V' = S' \cup T'$:
02.           IF $y \in S'$, THEN
03.                IF $D_y(T_{\ell(y)}) > \alpha \cdot d_{S'}$ and $\ell(y) < L$, THEN
04.                     Increment the level of $s$ by setting $\ell(y) \leftarrow \ell(y) + 1$.
05.                     Update the relevant data structures to reflect this change.
06.                ELSE IF $D_y(T_{\ell(y)-1}) < d_{S'}$ and $\ell(y) > 1$, THEN
07.                     Decrement the level of $y$ by setting $\ell(y) \leftarrow \ell(y) - 1$.
08.                     Update the relevant data structures to reflect this change.
09.           ELSE IF $y \in T'$, THEN
10.                IF $D_y(S_{\ell(y)}) > \alpha \cdot d_{T'}$ and $\ell(y) < L$, THEN
11.                     Increment the level of $s$ by setting $\ell(y) \leftarrow \ell(y) + 1$.
12.                     Update the relevant data structures to reflect this change.
13.                ELSE IF $D_y(S_{\ell(y)-1}) < d_{T'}$ and $\ell(y) > 1$, THEN
14.                     Decrement the level of $y$ by setting $\ell(y) \leftarrow \ell(y) - 1$.
15.                     Update the relevant data structures to reflect this change.
```

Figure 5: RECOVER-DIRECTED().

$$\mathcal{B} = \sum_{x \in S' \cup T'} \Phi(x) + \sum_{(s,t) \in E'} \Psi(s,t) \tag{60}$$

$$\Phi(s) = \left(\frac{1}{\epsilon}\right) \cdot \sum_{i=1}^{\ell(s)-1} \max(0, \alpha \cdot d_{S'} - D_s(T_i)) \quad \text{for all nodes } s \in S' \tag{61}$$

$$\Phi(t) = \left(\frac{1}{\epsilon}\right) \cdot \sum_{i=1}^{\ell(t)-1} \max(0, \alpha \cdot d_{T'} - D_t(S_i)) \quad \text{for all nodes } t \in T' \tag{62}$$

$$\Psi(s,t) = 2 \cdot (L - \min(\ell(s), \ell(t))) + f(s,t) \quad \text{for all edges } (s,t) \in E' \tag{63}$$

It is easy to check that all these potentials are nonnegative, and that they are uniquely defined by the $(\alpha, d_{S'}, d_{L'}, L)$-decomposition under consideration. Now, mimicking the potential function based analysis from Section 4.1, we can infer the following facts.

- (F1) In the beginning, when the derived graph $G' = (V', E')$ is empty, we have $\mathcal{B} = 0$. Subsequently, the potential $\mathcal{B}$ remains always nonnegative.

- (F2) Insertion/deletion of an edge in the derived graph $G' = (V', E')$ increases the potential $\mathcal{B}$ by at most $3L/\epsilon$.

- To analyze the amortized running time of the RECOVER-DIRECTED() procedure, we have the following claims.

  – (F3) Consider a single iteration of the WHILE loop in Figure 5 where a node $s \in S'$ with $\ell(s) = i$ changes (increments or decrements) its level by one. This takes $O(1 + D_s(T_i))$ time. On the other hand, the net drop in the overall potential $\mathcal{B}$ due to the same iteration of the WHILE loop is $\Omega(1 + D_s(T_i))$, provided $\alpha \geq 2 + 3\epsilon$.

  – (F4) Consider a single iteration of the WHILE loop in Figure 5 where a node $t \in T'$ with $\ell(t) = i$ changes (increments or decrements) its level by one. This takes $O(1 + D_t(S_i))$

time. On the other hand, the net drop in the overall potential $\mathcal{B}$ due to the same iteration of the WHILE loop is $\Omega(1 + D_t(S_i))$, provided $\alpha \geq 2 + 3\epsilon$.

Facts (F1) – (F4) imply that the RECOVER-DIRECTED() procedure takes a total time of $O(tL/\epsilon)$ to handling the first t edge insertions/deletions in the derived graph $G' = (V', E')$. This gives an amortized update time of $O(L/\epsilon)$ for our algorithm and concludes the proof of Theorem 6.8.

6.4   Wrapping Up: Proof of Theorem 6.2

We fix a sufficiently small constant $\epsilon \in (0, 1)$ and set $\alpha = 2 + 3\epsilon$, $L = 2 \cdot (2 + \lceil \log_{(1+\epsilon)} n \rceil)$, and $\lambda^* = 1 - 1/\sqrt{n}$. Next, as in Corollary 6.7, we discretize the range $[\lambda^*/(\alpha(1 + \epsilon)), n^2]$ by setting $q_k = (1+\epsilon)^{k-1} \cdot \lambda^*/\alpha$ for every integer $k \geq 0$. We then define K to be the smallest integer k for which $q_k \geq n^2$. Next, we maintain an $(\alpha, d_{S'}, d_{T'}, L)$-decomposition of the derived graph $G' = (V', E')$ for every $d_{S'}, d_{T'} \in \{d_0, \ldots, d_K\}$. By Theorem 6.8, maintaining each of these decompositions requires $O(m + n)$ space and $O(L/\epsilon)$ amortized update time. Hence, the total space requirement of our scheme is $O(K^2(m + n)) = \tilde{O}(m + n)$ and the total amortized update time is $O(K^2 L/\epsilon) = \tilde{O}(1)$. We also maintain the value of γ as defined in Corollary 6.7. Since there are $O(K^2)$ decompositions, maintaining the value of γ also requires $O(K^2) = \tilde{O}(1)$ update time. By Corollary 6.7, the quantity $\sqrt{\gamma}/(2\sqrt{1 + \epsilon})$ gives a $4\alpha \cdot (1 + \epsilon)^{3/2} = 8 \cdot (1 + O(\epsilon))$-approximation to the value of the densest subgraph $\rho(G)$. This concludes the proof of Theorem 6.2.

7   Sublinear-Time Algorithm

In this section, we focus on sublinear time algorithms for the approximate densest subgraph problem. Our main results are summarized in Theorems 7.1 and 7.3.

If we assume that an algorithm has to read *all* of its input, then no sublinear (in the input size) time algorithm is possible. However, if we assume that the input is given by an *oracle* that gives efficient access to the input, then sublinear time algorithms might exist. We present in the following such an oracle that allows us to turn our algorithm from Section 3 into a sublinear time algorithm. Specifically, we will give an $\tilde{O}(n)$ time algorithm that requires $\tilde{O}(n)$ oracle queries and space. Afterwards we will also show that with this oracle no further assymptotic improvement is possible.

Oracle model.   We first present the oracle model for the input graph. It is a standard representation that is, e.g., assumed in the sublinear time algorithms of [10, 19] and is called *incident-list model*. In this representation, we allow two types of accesses to the input graph (called *oracle queries*): (1) the *degree query* which asks for the degree of some node v, and (2) the *neighbor query* which asks for the ith neighbor of v (i.e. the ith element in the incidence list corresponding to the neighbors of v). See, e.g., [12, 21, 22, 36, 38, 39] for further surveys.

Upper Bound.   In Section 3, we showed how to compute a $(2 + \epsilon)$-approximate solution to the densest subgraph problem using only $\tilde{O}(n)$ edges sampled uniformly at random. In the above oracle model, sampling an edge can be done using one neighbor query. Thus, the algorithm needs only $\tilde{O}(n)$ queries. After the sampling is completed we can process the collection of sampled edges using $\tilde{O}(n)$ time and space, as in the proof of Theorem 3.1, simply by computing the $(1 + \epsilon, d, \tilde{O}(1))$-decomposition for different $\tilde{O}(1)$ values of d, to get the desired $(2 + \epsilon)$-approximate solution. This leads to the following theorem.

Theorem 7.1. *There is a sub-linear time algorithm for computing a $(2+\epsilon)$-approximate solution to the densest subgraph problem in the incidence-list model. The algorithm makes $\tilde{O}(n)$ oracle queries, and requires $\tilde{O}(n)$ time and $\tilde{O}(n)$ space.*

Lower Bound. We adapt the proof of [6, Lemma 7] to show that for any $\lambda \geq 3/2$, a λ-approximation algorithm needs to make $\Omega(n/(\lambda^2 \operatorname{poly} \log(n)))$ oracle queries. Consider the following communication complexity problem "P1":

- 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, and the communcation complexity is the number of bits written to the board. Using a reduction from the multi-party set disjointness problem, the papers [6, 8] showed that this problem require $\tilde{\Omega}(\ell/k) = \tilde{\Omega}(n/k^2)$ communication bits.

Lemma 7.2. *If there is a sublinear-time algorithm with q oracle queries for the problem P1 defined above, then the problem P1 can also 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 tth 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. $\square$

Note that any $((k-1)/2 - \epsilon)$-approximation algorithm for the densest subgraph problem solves problem P1. Thus, the above lemma implies that any $((k-1)/2 - \epsilon)$-approximation algorithm requires $\tilde{\Omega}(n/k)$ queries. By considering any $k \geq 4$, we get the following theorem.

Theorem 7.3. *In the incidence-list model, for any $\lambda \geq 3/2$ and any $\epsilon > 0$, any $\lambda - \epsilon$-approximation algorithm for the densest subgraph problem needs to make $\tilde{\Omega}(n/\lambda^2)$ queries.*

8 Distributed Streams

In the distributed streaming model (see, e.g., [11]), there are 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 sample $\tilde{O}(n)$ edges (without replacement) as a sketch by using the sampling algorithm of Cormode et al. [11]: their algorithm can sample $\tilde{O}(n)$ edges using $\tilde{O}(k + n)$ bits of communication, whereas the coordinator needs $\tilde{O}(n)$ space and each site needs $\tilde{O}(1)$ space. The coordinator can then use this sketch to compute a $(2 + \epsilon)$-approximate solution.

Theorem 8.1. *In the distributed streaming setting with k sites [11], we can compute a $(2 + \epsilon)$-approximate solution to the densest subgraph problem using $\tilde{O}(k + n)$ bits of communication. The coordinator needs $\tilde{O}(n)$ space and each site needs $\tilde{O}(1)$ space.*

9 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. [14] 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 [23]. 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. 1, 3

[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. 1, 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. 1

[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. 1

[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. 1, 4, 5, 45

[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. 45

[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. 1, 4

[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. 4, 44

[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. 4, 45, 46

[12] Artur Czumaj and Christian Sohler. Sublinear-time algorithms. In *Property Testing*, pages 41–64, 2010. 44

[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. 1

[14] Alessandro Epasto, Silvio Lattanzi, and Mauro Sozio. Efficient densest subgraph computation in evolving graphs. In *Proceedings of the 24th International Conference on World Wide Web, WWW 2015, Florence, Italy, May 18-22, 2015*, pages 300–310, 2015. 4, 46

[15] Paul Erdös, Arthur H Stone, et al. On the structure of linear graphs. *Bull. Amer. Math. Soc*, 52:1087–1091, 1946. 5

[16] Hossein Esfandiari, MohammadTaghi Hajiaghayi, and David P. Woodruff. Applications of uniform sampling: Densest subgraph and beyond. *CoRR*, abs/1506.04505, 2015. 4

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

[18] David Gibson, Ravi Kumar, and Andrew Tomkins. Discovering large dense subgraphs in massive graphs. In *VLDB*, pages 721–732, 2005. 1

[19] 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. 4, 44

[20] A. V. Goldberg. Finding a maximum density subgraph. Technical report, Berkeley, CA, USA, 1984. 1

[21] Oded Goldreich. A brief introduction to property testing. In *Studies in Complexity and Cryptography*, pages 465–469. 2011. 44

[22] Oded Goldreich. Introduction to testing graph properties. In *Studies in Complexity and Cryptography*, pages 470–506. 2011. 44

[23] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 21–30, 2015.

[24] 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. 10, 11

[25] Ravi Kannan and V. Vinay. Analyzing the structure of large graphs. *Technical report*, 1999. 37

[26] 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. 1

[27] 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. 1

[28] 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. 1

[29] Samir Khuller and Barna Saha. On finding dense subgraphs. In *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I*, pages 597–608, 2009. 38

[30] Eugene Lawler. *Combinatorial Optimization: Networks and Matroids*. Saunders College Publishing, Fort Worth, 1976. 1

[31] 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. 1

[32] 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. 5

[33] Andrew McGregor, David Tench, Sofya Vorotnikova, and Hoa T. Vu. Densest subgraph in dynamic graph streams. In *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part II*, pages 472–482, 2015. 4

[34] Michael Mitzenmacher, Jakub Pachocki, Richard Peng, Charalampos E. Tsourakakis, and Shen Chen Xu. Scalable large near-clique detection in large-scale networks via sampling. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 815–824. ACM, 2015. 4

[35] J. Ian Munro and Mike Paterson. Selection and sorting with limited storage. *Theor. Comput. Sci.*, 12:315–323, 1980. Announced at FOCS'78. 1

[36] Krzysztof Onak. Sublinear graph approximation algorithms. In *Property Testing*, pages 158–166, 2010. 44

[37] Anna Pagh and Rasmus Pagh. Uniform hashing in constant time and optimal space. *SIAM J. Comput.*, 38(1):85–96, 2008. 10, 22

[38] Dana Ron. Algorithmic and analysis techniques in property testing. *Foundations and Trends in Theoretical Computer Science*, 5(2):73–205, 2009. 44

[39] Ronitt Rubinfeld and Asaf Shapira. Sublinear time algorithms. *SIAM J. Discrete Math.*, 25(4):1562–1588, 2011. 44

[40] 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. 5

[41] George Szekeres and Herbert S Wilf. An inequality for the chromatic number of a graph. *Journal of Combinatorial Theory*, 4(1):1–3, 1968. 5

[42] Lei Tang and Huan Liu. Graph mining applications to social network analysis. In *Managing and Mining Graph Data*, pages 487–513. 2010. 1

[43] Charalampos Tsourakakis. The k-clique densest subgraph problem. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1122–1132. International World Wide Web Conferences Steering Committee, 2015. 1