New Parallel and Streaming Algorithms for Directed Densest Subgraph

Slobodan Mitrović*
University of California, Davis
smitrovic@ucdavis.edu

Theodore Pan*
University of California, Davis
thjpan@ucdavis.edu

Mahdi Qaempanah*
Sharif University of Technology
mahdi.ghaempanah111@student.sharif.edu

Mohammad Amin Raeisi*
Yale University
amin.raeisi@yale.edu *

Abstract

Finding dense subgraphs is a fundamental problem with applications to community detection, clustering, and data mining. Our work focuses on finding approximate densest subgraphs in *directed graphs* in computational models for processing massive data. We consider two such models: Massively Parallel Computation (MPC) and semi-streaming. We show how to find a $(2+\varepsilon)$-approximation in $\tilde{O}(\sqrt{\log n})$ MPC rounds with sublinear memory per machine. This improves the state-of-the-art results by Bahmani et al. [BGM14, WAW 2014] and Mitrović & Pan [MP24, ICML 2024]. Moreover, we show how to find an $O(\log n)$-approximation in a *single pass* in semi-streaming. This is in stark contrast to prior work, which implies $\tilde{\Omega}(n^{1/6})$ approximation for a single pass; a better approximation is known only for randomized streams (Mitrović & Pan). This is the first *deterministic single-pass* semi-streaming algorithm for the densest subgraph problem, both for undirected and directed graphs. Our semi-streaming approach is also an insertion-only dynamic algorithm, attaining the first directed densest subgraph algorithm with $O(\log^2 n)$ worst-case update time while using sub-linear memory. We empirically evaluate our approaches in two ways. First, we illustrate that our single-pass semi-streaming algorithm performs much better than the theoretical guarantee. Specifically, its approximation on temporal datasets matches the $(2+\varepsilon)$-approximation of an $O(\log n)$-pass algorithm by Bahmani et al. [BKV12, VLDB 2012]. Second, we demonstrate that our MPC algorithm requires fewer rounds than prior work.

1 Introduction

Computing dense subgraphs in directed graphs is a classical optimization task where we are interested in subgraphs with a large edge-to-vertex ratio. In particular, given a directed graph $G=(V,E)$, the densest subgraph (DS) problem asks to find two vertex subsets $S,T\subseteq V$ such that the ratio $|E(S,T)|/\sqrt{|S|\cdot|T|}$ is maximized, where $E(S,T)$ is the set of directed edges from S to T. This problem has found a wide range of applications in graph mining, including the analysis of social networks [LRJA10, For10, CS10], bioinformatics [FNBB06, SHK+10], visualization [ZP12, ZT12], and finance [ZZY+17, FRM19, CT22, JZT+22].

With the constant increase in the size and prevalence of large datasets, it has become crucial to develop algorithms that solve fundamental optimization problems with limited constraints, such

*Equal contribution.

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

Massively Parallel Computation model				
	Approximation	Memory per machine	Round complexity	Reference
Undirected	$(1+\varepsilon)$	$O(n^\delta)$	$\tilde{O}(\sqrt{\log n})$	[GLM19]
Directed	$(1+\varepsilon)$	$O(n^\delta)$	$O(\log n)$	[BGM14]
	$(2+\varepsilon)$	$O(n^\delta)$	$\tilde{O}(\sqrt{\log n})$	Our work
	$(2+\varepsilon)$	$\tilde{O}(n)$	$O(\sqrt{\log n})$	[MP24]

Semi-streaming model				
	Approximation	# of passes	Deterministic or randomized	Reference
Undirected	$(1+\varepsilon)$	1	Randomized	[EHW15]
	$O(\log n)$	1	Deterministic	Our work
	$(2+\varepsilon)$	$O(\log n)$	Deterministic	[BKV12]
Directed	$\tilde{\Omega}(n^{1/6})$	1	Randomized	[EHW15]
	$(2+\varepsilon)$	1	Random order stream	[MP24]
	$O(\log n)$	1	Deterministic	Our work
	$(2+\varepsilon)$	$O(\log n)$	Deterministic	[BKV12]

Dynamic model				
	Approximation	Memory usage	Update time	Reference
Undirected	$(4+\varepsilon)$	$\tilde{O}(n)$	$\tilde{O}(1)$ amortized	[BHNT15]
	$O(\log n)$	$\tilde{O}(n)$	$\tilde{O}(1)$ worst-case	Our work (insertion-only)
	$(1+\varepsilon)$	$\tilde{O}(m)$	$\tilde{O}(1)$ worst-case	[SW20]
Directed	$O(\log n)$	$\tilde{O}(n)$	$\tilde{O}(1)$ worst-case	Our work (insertion-only)
	$(1+\varepsilon)$	$\tilde{O}(m)$	$\tilde{O}(1)$ worst-case	[SW20]

Table 1: A summary of state-of-the-art results on the DS problem in MPC, semi-streaming, and dynamic models for constants $\epsilon > 0$ and $\delta \in (0,1)$, and graphs with n vertices and m edges.

as memory per computing unit or data access. While it is known how to find a DS in polynomial time [Cha00], it is unclear how to efficiently implement this algorithm in the context of large-scale modern computation. It inspired several research groups to study DS computation in distributed, parallel, and streaming settings under various memory constraints and approximation guarantees.

In Massively Parallel Computation (MPC), which is a theoretical abstraction of popular large-scale frameworks such as MapReduce and Hadoop, [BGM14] proposed an $O(\log n/\varepsilon^2)$ MPC round algorithm for constructing $(1+\varepsilon)$-approximate DS for directed and undirected graphs, where $\varepsilon > 0$ is a precision parameter and $n = |V|$. For undirected DS, this complexity was improved by [GLM19] to $O(\sqrt{\log n} \cdot \log\log n)$ rounds. Recently, in the setting where each machine in MPC has $\tilde{\Theta}(n)$ memory, [MP24] designed an algorithm for $(2+\varepsilon)$-approximation of directed DS that takes $O(\sqrt{\log n})$ rounds.

In the context of semi-streaming, [BKV12] proposed an elegant peeling-based algorithm that computes a $(2+\varepsilon)$-approximation of directed DS in $O(\log n/\varepsilon)$ passes. If the goal is to compute undirected DS, [EHW15] provide a single-pass algorithm that guarantees a $(1+\varepsilon)$-approximation. Interestingly, the same technique for directed DS attains a $\tilde{\Omega}(n^{1/6})$-approximation. Better approximations are only known if the underlying stream is randomized [MP24] or if $\Omega(n^{1.5})$ memory can be used in the semi-streaming setting [EHW15].

Motivated by this disparity between the state-of-the-art results for directed and undirected DS, we ask: *What algorithmic techniques help narrow the gap in computing directed versus undirected DS?*

1.1 Our contributions

Table 1 provides a summary of previous state-of-the-art results in comparison to ours.

Result 1 (Theorem 4.2 rephrased)**.** *Given an n-vertex graph and $\varepsilon > 0$, there exists a sublinear MPC algorithm that outputs a $(2+\varepsilon)$-approximate directed DS in $\tilde{O}(\sqrt{\log n})$ rounds. The algorithm uses $O(n^\delta)$ memory per machine and $O(n^{1+\delta}+m)$ total memory for $\delta \in (0,1)$.*

This improves on [BGM14], which uses $O(\log n)$ rounds, and on [MP24], which uses $O(\sqrt{\log n})$ MPC rounds but requires near-linear memory per machine. Additionally, Result 1 matches the state-of-the-art round complexity of $\tilde{O}(\sqrt{\log n})$ for undirected graphs from [GLM19], bridging the gap between the directed and undirected DS problems in MPC.

Result 2 (Theorem 5.1 rephrased)**.** *Given an n-vertex graph and $\varepsilon > 0$, there exists a single-pass deterministic semi-streaming algorithm that outputs an $O(\log n)$-approximate directed DS.*

This is the first single-pass semi-streaming algorithm for the directed DS problem on arbitrary streams. [MP24] attains a $(2+\varepsilon)$-approximation but only for randomized streams. [EHW15] attains a $(1+\varepsilon)$-approximation when additional memory, i.e., $O(n^{1.5} \operatorname{poly}\log n)$, is allowed. If we were to extend the ideas of using uniform sampling from prior works, a generalization of the construction in [MP24] shows that it will result in at least a $\tilde{\Omega}(n^{1/6})$-approximation. We also note that this is a deterministic algorithm and can be easily adapted to undirected graphs, leading to the first deterministic single-pass semi-streaming algorithm for the DS problem in both undirected and directed cases.

Result 2 is also an insertion-only dynamic algorithm. Its worst-case update time is $O(\log n)$ for undirected and $O(\log^2 n)$ for directed graphs. To our knowledge, no previous algorithm maintains an approximate *directed* DS in semi-streaming. [BHNT15] shows how to maintain a $(4+\varepsilon)$-approximate undirected DS with $\tilde{O}(1)$ amortized update time and $\tilde{O}(n)$ memory but may have $\Omega(n)$ worst-case update time. [SW20] shows how to maintain a $(1+\varepsilon)$-approximate directed DS with $O(\log^5 n)$ worst-case update time but requires linear memory.

Empirical evaluation suggests that, in practice, our semi-streaming algorithm yields an approximation much better than $\log n$. On temporal datasets specifically, it matches the approximation of [BKV12].

2 Preliminaries

Transformation: General directed to bipartite undirected graph. Given a directed graph $G_{\text{dir}} = (V_{\text{dir}}, E_{\text{dir}})$, we represent it as a bipartite graph $G = (V_1, V_2, E)$ where: (i) V_1 and V_2 are two copies of V_{dir}, and (ii) there is an edge in E between $u \in V_1$ and $v \in V_2$ iff there is a directed edge from u to v in E_{dir}. All directed graphs will be treated as bipartite with this representation.

Notation. For a bipartite graph $G = (S, T, E)$, we use n to refer to $|S| + |T|$, the total number of vertices. Given two vertex sets $A \subseteq S$ and $B \subseteq T$, we refer to the edges between them by $E_G(A, B) \stackrel{\text{def}}{=} \{e = (i, j) \in E : i \in A, j \in B\}$. We use $d_G(v)$ to denote the degree of vertex v in G.

Directed densest subgraph. Given bipartite graph $G = (V, V, E)$ and vertex sets $S, T \subseteq V$, the *density* $\rho(S, T)$ is defined as $\rho(S, T) \stackrel{\text{def}}{=} |E_G(S, T)|/\sqrt{|S| \cdot |T|}$. A *directed densest subgraph* is sets S^*, T^* such that $(S^*, T^*) \in \arg\max_{S,T \subseteq V} \rho(S, T)$.

Massively Parallel Computation (MPC). In MPC, synchronous rounds of computation are performed across N machines. Each machine has S words of memory and, initially, the input data is arbitrarily distributed across the machines. During a round, each machine computes its local data. Then, after the round, machines exchange messages synchronously. Each machine can send messages to any other machine, but each machine can send and receive at most S words of data. The primary objective is to perform computation in as few rounds as possible. With respect to S, three regimes are primarily studied: given $\delta \in (0, 1)$, *sub-linear* ($S = n^{\delta}$), *near-linear* ($S = n \operatorname{poly}\log n$), and *super-linear* ($S = n^{1+\delta}$). In this work, we focus on the most restrictive sub-linear memory regime. Even though the running time in definition of the MPC model is allowed to be arbitrarily large, the running time of our MPC algorithm is near-linear per round.

Semi-streaming. In the semi-streaming setting, an input graph $G = (V, E)$ is given as a stream of edges. That is, an algorithm receives one edge $e \in E$ at a time, and updates its internal memory based on e. This internal memory is constrained to be $O(n \cdot \operatorname{poly}\log n)$. After all edges are presented as a stream, we say the algorithm made a *pass* over the graph. An algorithm can make multiple passes over data. During a pass, the algorithm can perform arbitrarily large polynomial-time computations. We remark that our algorithm spends $O(\operatorname{poly}\log n)$ time to update its memory per edge.

3 The base algorithm

In this section, we describe the base of our approach. Then, in Sections 4 and 5, we develop new insights enabling us to extend this approach to the MPC and semi-streaming model.

As a reminder, we use S^* and T^* to denote the vertex subsets corresponding to the densest subgraph. Additionally, we view the input graph G as a bipartite undirected rather than a directed graph; details of this transformation are described in Section 2. We assume that our algorithm is given the density $D = \rho(S^*, T^*)$ and the ratio $z = \sqrt{|S^*|/|T^*|}$.[2] Then, leveraging the knowledge of D and z, the main idea of this base algorithm is to construct a pair of vertex subsets $(S, T) \subseteq V \times V$ such that it approximates (S^*, T^*) in the following sense:

- Each vertex in S has at least $D/(2z)$ neighbors in T.
- Each vertex in T has at least $Dz/2$ neighbors in S.

We show that such a pair (S, T) must exist and that it is a 2-approximation of the directed densest subgraph. The proofs of the following lemmas can be found in Appendix A and Appendix B.

Lemma 3.1 (Subgraph existence)**.** *Let (S^*, T^*) be a directed densest subgraph. Let $D = \rho(S^*, T^*)$ be its density and $z = \sqrt{|S^*|/|T^*|}$. There exists an induced subgraph H on vertex sets (S, T) for which it holds that $d_H(v) \geq D/(2z)$ for all $v \in S$ and $d_H(v) \geq Dz/2$ for all $v \in T$.*

Lemma 3.2 (Sufficient condition for 2-approximation)**.** *Let (S^*, T^*) be a directed densest subgraph. Let $D = \rho(S^*, T^*)$ be its density and $z = \sqrt{|S^*|/|T^*|}$. Then, any induced subgraph H on vertex sets (S, T) which satisfies $d_H(v) \geq D/(2z)$ for all $v \in S$ and $d_H(v) \geq Dz/2$ for all $v \in T$ has density at least $D/2$. In other words, it is a 2-approximation of the directed densest subgraph.*

Lemma 3.2 provides sufficient conditions under which a given subgraph is a 2-approximate densest one. These conditions inspire a simple peeling procedure for constructing such a subgraph, e.g., Algorithm 1. Let S and T denote the two bipartite sides of that maintained subgraph. Each S and T is initialized to V. Then, it iteratively removes vertices that do **not** satisfy the degree conditions stated by Lemma 3.2. More precisely, the algorithm iteratively removes vertices from S with degrees less than $D/(2z)$ and vertices from T with degrees less than $Dz/2$. If **only** these two steps were performed without any extra stopping rule, this algorithm *could execute too many peeling iterations before terminating*. To see that, consider the construction in Figure 1.

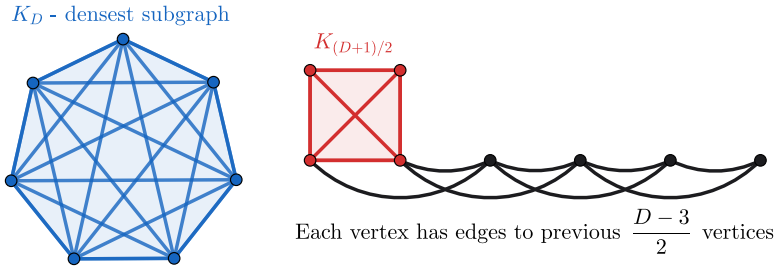

Figure 1: An illustration of the construction that requires $\Omega(n)$ iterations of peeling. We observe that the added vertices (black vertices) can extend as far as desired and will be removed one by one from right to left.

We construct an undirected graph; we obtain its directed version by considering each undirected edge as two directed ones. Let the densest subgraph be the complete graph K_D, which has density $D - 1$ and $z = 1$, so we will peel vertices with degree less than $(D - 1)/2$. Then, disjoint to this subgraph, we start with a $\frac{D+1}{2}$-complete subgraph with vertices labeled $1, \ldots, \frac{D+1}{2}$. We continually add vertex i with edges to vertices $i - \frac{D-3}{2} + 1, i - \frac{D-1}{2} + 2, \ldots, i - 1$ for $i > \frac{D+1}{2}$. This disjoint subgraph and all of its subgraphs have a smaller density than K_D, and the number of iterations of peeling is linear with the number of vertices we add, resulting in $\Omega(n)$ iterations of peeling.

We make a crucial observation that enables us to avoid this behavior – if a set S or T does not decrease in size by a factor of $(1+\varepsilon)$ at least, the current (S, T) subgraph already achieves the desired

[2] Our final algorithms do not require knowledge of D and z. Those details are discussed in Section 4.

approximation (see the proof of Theorem 3.4 for details). This stopping criterion – implemented by Lines 5 and 6 – enables us to ensure sufficient progress in each peeling iteration.

Algorithm 1 Computes a $2(1+\varepsilon)$-approximate directed densest subgraph assuming that $D = \rho(S^*, T^*)$ and $z = \sqrt{|S^*|/|T^*|}$

Input: bipartite graph $G = (S, T, E)$, parameters $\varepsilon > 0$, D and z

1: $k_S = D/(2z)$, $k_T = Dz/2$
2: **while** true **do**
3: $A \leftarrow$ all vertices $v \in S$ with $d_G(v) < k_S$
4: $B \leftarrow$ all vertices $v \in T$ with $d_G(v) < k_T$
5: **if** $|S| \geq z^2|T|$ and $|A| \leq \frac{\varepsilon}{1+\varepsilon}|S|$ **then return** (S, T)
6: **if** $|S| \leq z^2|T|$ and $|B| \leq \frac{\varepsilon}{1+\varepsilon}|T|$ **then return** (S, T)
7: Remove A from S and B from T

3.1 Comparison to prior work

The idea of gradually peeling vertices based on their degrees has appeared in prior works on densest subgraphs, e.g., see [BKV12, GLM19, MP24]. Nevertheless, we are unaware of a peeling-based algorithm that uses stopping rules akin to those on Lines 5 and 6 of Algorithm 1. On the one hand, a typical peeling iteration for computing densest subgraphs removes vertices whose degree is less than $1+\varepsilon$ times the average degree. This immediately implies $O(\log_{1+\varepsilon} n)$ peeling steps and has no need for stopping rules. However, the average degree changes as the graph changes; so, the degree threshold above which to peel vertices evolves. On the other hand, Algorithm 1 uses *the same* degree thresholds, i.e., k_S and k_T, throughout the entire execution. Then, the advantage of fixed degree thresholds is that they *do not need to be recomputed* in distributed and streaming settings. This is one of the key properties enabling us to design our algorithms in the coming sections.

3.2 Analysis of Algorithm 1

We use *peeling iteration* to refer to a single while-loop iteration of Algorithm 1. During a peeling iteration, all vertices in S or T with degree below the respective threshold are removed, implying:

Lemma 3.3. *Algorithm 1 executes $O(\log_{1+\varepsilon} n)$ peeling iterations.*

Proof. After a peeling iteration, either S or T decreases in size by a factor of at least $(1+\varepsilon)$, or the algorithm outputs the current vertex sets and terminates (Lines 5 and 6). Once S and T are empty, Algorithm 1 finishes. Hence, there are $O(\log_{1+\varepsilon} n)$ iterations of peeling. □

Theorem 3.4. *Let (S^*, T^*) be a directed densest subgraph of G. For $D = \rho(S^*, T^*)$ and $z = \sqrt{|S^*|/|T^*|}$, Algorithm 1 outputs a $2(1+\varepsilon)$-approximate directed densest subgraph.*

Proof. Let H be the subgraph from Lemma 3.1. Since, $D = \rho(S^*, T^*)$ and $z = \sqrt{|S^*|/|T^*|}$, Algorithm 1 will not remove any vertex from H. Hence, S and T produced by Algorithm 1 are not empty sets. Therefore, there must exist an iteration of Algorithm 1 in which A (resp. B) is at most $\varepsilon/(1+\varepsilon)|S|$ (resp. $\varepsilon/(1+\varepsilon)|T|$). That is, in this iteration, S and T would decrease in size by a factor *less* than $1+\varepsilon$. Observe that when such an iteration occurs, the algorithm terminates. Therefore, we have the following two cases.

Case Algorithm 1 terminates at Line 5. We have that $|A| \leq \frac{\varepsilon}{1+\varepsilon}|S|$ meaning that at least $\frac{1}{1+\varepsilon}|S|$ vertices in S have degree at least k_S. This allows us to lower bound the density, giving us

$$\rho(S, T) \geq \frac{k_S \cdot \frac{|S|}{1+\varepsilon}}{\sqrt{|S||T|}} = \frac{D}{2(1+\varepsilon)} \cdot \frac{1}{z}\sqrt{\frac{|S|}{|T|}} \geq \frac{D}{2(1+\varepsilon)}$$

where we used $|S| \geq z^2|T|$. Therefore, the produced subgraph is a $2(1+\varepsilon)$-approximation.

Case Algorithm 1 terminates at Line 6. Similarly, we have that $|B| \leq \frac{\varepsilon}{1+\varepsilon}|T|$ meaning that at least $\frac{1}{1+\varepsilon}|T|$ vertices in T have degree at least k_T. This gives us

$$\rho(S,T) \geq \frac{k_T \cdot \frac{|T|}{1+\varepsilon}}{\sqrt{|S||T|}} = \frac{D}{2(1+\varepsilon)} \cdot z\sqrt{\frac{|T|}{|S|}} > \frac{D}{2(1+\varepsilon)}$$

using the $|S| \leq z^2|T|$ constraint. Therefore, this also produces a $2(1+\varepsilon)$-approximation. □

4 $\tilde{O}(\sqrt{\log n})$ MPC rounds in the sublinear memory regime

In this section, we extend Algorithm 1 to the MPC sublinear memory regime. Directly translating Algorithm 1 into an MPC is straightforward – standard MPC algorithmic primitives, e.g., see [GSZ11, ASS+18], enable us to perform one iteration of peeling in $O(1)$ MPC rounds. Since Algorithm 1 takes $O(\log n)$ iterations, this direct MPC implementation results in an $O(\log n)$ MPC round algorithm in the sublinear memory regime. However, we aim to obtain a quadratically faster algorithm.

4.1 Our improved approach

On a high level, we execute the $O(\log n)$ iterations of (a variant of) Algorithm 1 in $\tilde{O}(\sqrt{\log n})$ MPC rounds. We present the intuition behind this approach in two steps: (i) Recall a known framework for simulating $O(\log n)$ iterations in $\tilde{O}(\sqrt{\log n})$ MPC rounds; and (ii) Describe a modified variant of Algorithm 1 that fits into that simulation framework.

Known simulation techniques. Consider a T-iteration algorithm $\mathcal{A}_{\text{LOCAL}}$ such that: (1) $\mathcal{A}_{\text{LOCAL}}$ maintains a state of each vertex, e.g., a vertex v is removed or not, and (2) the state of v in iteration i depends only on the states of v and its neighbors in iteration $i-1$, e.g., $\mathcal{A}_{\text{LOCAL}}$ decides whether v should be removed in iteration i based on the number of v's non-removed neighbors in iteration $i-1$. The **output** of $\mathcal{A}_{\text{LOCAL}}$ is the state of each vertex in each of the T iterations.

Remark: Algorithm 1 does not have the properties of $\mathcal{A}_{\text{LOCAL}}$, e.g., evaluating the conditions on Lines 5 and 6 requires "global" computation. However, we note that these lines can be simulated through post-processing, constructing sets S and T for each iteration and comparing their sizes using the states of vertices. The remaining steps of Algorithm 1 can be phrased in the language of $\mathcal{A}_{\text{LOCAL}}$.

In how many MPC rounds can we execute $\mathcal{A}_{\text{LOCAL}}$? Based on our description, observe that the state of a vertex v at any moment during the execution of $\mathcal{A}_{\text{LOCAL}}$ is a function of the v's T-hop neighborhood. Let $N_j(v)$ denote the j-hop neighborhood of v. This observation gives rise to the following idea:

(1) Gather $N_T(v)$ of each vertex $v \in V$.

(2) Place $N_T(v)$ on a single machine in MPC; $N_T(v)$ for different v are placed on different machines.

(3) In a single MPC round, execute $\mathcal{A}_{\text{LOCAL}}$ within $N_T(v)$ to learn the state of v.

Assuming that $N_T(v)$ fits in the memory of a single machine, a significant advantage of this approach is that it takes only $O(\log T)$ MPC rounds. Namely, Item 1 can be implemented using a well-known technique *graph exponentiation* [LW10, Gha17]. In this technique, $N_{2^i}(v)$ is gathered for each $v \in V$ and for each $i = 0 \cdots \log T$, using the following relation:

$$N_{2^{i+1}}(v) = \bigcup_{w \in N_{2^i}(v)} N_{2^i}(w).$$

Namely, each iteration of the graph exponentiation technique doubles the radius of the neighborhood collected around a vertex. To implement this technique in MPC, it is necessary to address: *Can $N_T(v)$ fit into the memory of a single machine in MPC?* This takes us to the second part of our approach.

Graph sparsification (Algorithm 2) – A modified variant of Algorithm 1. We can alter the recipe above to simulate $\mathcal{A}_{\text{LOCAL}}$ in MPC while still aiming for $o(T)$ rounds. Namely, (i) we can split those T iterations into T/k groups, each consisting of k consecutive iterations, (ii) execute group after group sequentially, such that (iii) each group is executed as described above by using graph exponentiation. Setting aside memory constraints, this method uses $O\left(\frac{T}{k} \cdot \log k\right)$ MPC rounds. The larger k is, the fewer MPC rounds are needed. However, a larger k implies a bigger $N_k(v)$ which might not fit in a

Algorithm 2 Finds partial $(2+\varepsilon)$-approximation of directed densest subgraph

Input: bipartite graph $G = (S, T, E)$, $\varepsilon \in (0, 1)$, $\delta \in (0, 1)$, D and z

1: $k_S = D/(2z)$, $k_T = Dz/2$, $\alpha = (1+\varepsilon)^{\sqrt{\log_{1+\varepsilon} n}}$
2: Freeze all vertices in S of degree greater than $k_S\alpha$
3: Freeze all vertices in T of degree greater than $k_T\alpha$
4: Mark as frozen each edge with both endpoints frozen
5: $f_1 \leftarrow$ number of frozen vertices in S
6: $f_2 \leftarrow$ number of frozen vertices in T
7: **if** $f_1 \geq \frac{z\sqrt{|S||T|}}{\alpha}$ or $f_2 \geq \frac{\sqrt{|S||T|}}{z\alpha}$ **then return** (S, T)
8: $p_1 \leftarrow \min\left(1, \frac{18\log n}{\varepsilon^2 k_S}\right)$
9: $p_2 \leftarrow \min\left(1, \frac{18\log n}{\varepsilon^2 k_T}\right)$
10: $t \leftarrow \frac{\sqrt{\delta \log_{1+\varepsilon} n}}{2}$
11: **for** t steps **do**
12: $\quad G_1 \leftarrow$ sample of each non-frozen edge of G with probability p_1
13: $\quad G_2 \leftarrow$ sample of each non-frozen edge of G with probability p_2
14: $\quad A \leftarrow$ all non-frozen vertices $v \in S$ with $d_{G_1}(v) < pk_S$
15: $\quad B \leftarrow$ all non-frozen vertices $v \in T$ with $d_{G_2}(v) < pk_T$
16: $\quad$ **if** $|S| \geq z^2|T|$ and $|A| \leq \frac{\varepsilon}{1+\varepsilon}|S| - f_1$ **then return** (S, T)
17: $\quad$ **if** $|S| \leq z^2|T|$ and $|B| \leq \frac{\varepsilon}{1+\varepsilon}|T| - f_2$ **then return** (S, T)
18: $\quad$ Remove A from S and B from T
19: **return** (S, T)

machine's memory. It turns out that $k = \Theta\left(\sqrt{\log n}\right)$ is the largest k our approach can tolerate after performing certain graph sparsification from [GU19] described next.

Fix $t = \Theta(\sqrt{\log n})$. Ignoring the memory-per-machine constraint, $N_t(v)$ can be collected in $O(\log\log n)$ MPC rounds. However, we cannot guarantee that these neighborhoods fit on a single machine with $O(n^\delta)$ memory, as some vertices can have large degrees. For example, vertices with a degree of $\omega(n^\delta)$ cannot even store their entire neighborhood on a single machine. Inspired by this, Algorithm 2 temporarily "ignores" these high-degree vertices when performing graph exponentiation. We call this process *freezing* high-degree vertices, seen on Line 2 and Line 3. Intuitively, freezing enables us to transform the current graph into one having sufficiently small degrees, and hence the graph exponentiation can be executed with $O\left(n^\delta\right)$ memory per machine. On Line 4, we also ignore the edges between frozen vertices since they do not affect the peeling of non-frozen vertices.

Algorithm 2 samples the graph on top of freezing high-degree vertices on Line 12 and Line 13. Combining both freezing and sampling, the graph becomes sparse enough so that $\Theta(t)$-hop neighborhoods fit within sublinear memory and peeling is simulated with high probability. Therefore, if $\Theta(t)$ iterations of peeling can be simulated in $O(\log\log n)$ rounds, then we can simulate the entirety of Algorithm 1 in $O(\sqrt{\log n} \cdot \log\log n)$ rounds. Between these phases of simulating $\Theta(\sqrt{\log n})$ iterations of peeling, frozen vertices are updated based on their new degrees. However, these frozen vertices are not peeled during these phases, and therefore our peeling does not quite match $\Theta(\sqrt{\log n})$ iterations of peeling in Algorithm 1. Nevertheless, our final algorithm uses $\tilde{O}(\sqrt{\log n})$ rounds. We show that the fraction of frozen vertices is too small to affect the entire peeling process.

4.2 Analysis of Algorithm 2

When trying to simulate Algorithm 1, it is important to note that our stopping rules are affected since we have limited information on the degrees of frozen vertices. So, we weaken our rules and this is reflected in the differences between Line 5 and Line 6 of Algorithm 1 and Line 16 and Line 17 of Algorithm 2. Algorithm 2 simulates $t = \Theta(\sqrt{\log n})$ iterations of peeling as described above, but due to frozen vertices, it is not obvious how much the sizes of vertex sets decrease by. Nevertheless, we establish the following claim about the number of frozen vertices. The proof of Lemma 4.1 is deferred to Appendix C.

Lemma 4.1. *Let $G = (S, T, E)$ be a bipartite graph and (S^*, T^*) be its directed densest subgraph. Let (S', T') be the output of Algorithm 2 ran on G given $\frac{\rho(S^*,T^*)}{(1+\varepsilon)^3} \leq D \leq \frac{\rho(S^*,T^*)}{(1+\varepsilon)^2}$ and $\frac{\sqrt{|S^*|/|T^*|}}{(1+\varepsilon)} \leq z \leq \sqrt{|S^*|/|T^*|}$. Then, with probability at least $1 - \frac{t}{n^2}$, it holds that:*

- ***Good approximation.*** *(S', T') is a $2(1+\varepsilon)^6$-approximate densest subgraph of G, or*
- ***Size reduction.*** *(S', T') contains a $2(1+\varepsilon)^6$-approximate densest subgraph of G and*

$$|S'| \leq \frac{|S|}{\gamma} \text{ or } |T'| \leq \frac{|T|}{\gamma}$$

where $\gamma = (1+\varepsilon)^{\frac{\sqrt{\delta \log_{1+\varepsilon} n}}{8}}$.

Following the ideas described before, we invoke Algorithm 2 $\Theta(\sqrt{\log n})$ times to simulate all iterations of peeling, resulting in the following theorem, whose proof is deferred to Appendix D.

Theorem 4.2. *There exists a sublinear MPC algorithm that runs in $\tilde{O}(\sqrt{\log n})$ rounds and attains a $2(1+\varepsilon)^6$-approximation of the directed densest subgraph with probability at least $1 - \frac{1}{n}$. The algorithm uses $O(n^\delta)$ memory per machine and $O(n^{1+\delta} + m)$ total memory for $\delta \in (0, 1)$.*

5 Single-pass semi-streaming algorithm

In this section, we extend Algorithm 1 to the single-pass semi-streaming setting. It is unclear how to adapt the sampling used for the MPC algorithm since the number of edges sampled could be $\omega(n \operatorname{poly} \log n)$, surpassing the memory limit of the semi-streaming model. Nonetheless, we show that an $O(\log n)$-approximate directed densest subgraph can be obtained by maintaining **only vertex degrees** throughout the stream. This directly results in a $\tilde{O}(n)$ memory requirement, as a vertex's degree can be maintained using a simple integer counter. To the best of our knowledge, this is the first semi-streaming single-pass algorithm for the directed densest subgraph problem that achieves better than $\operatorname{poly} n$ approximation.

5.1 Our approach

Algorithm 3 Finds $O(\log n)$-approximation of directed densest subgraph

Input: bipartite graph $G = (S, T, E), \varepsilon > 0$, D and z

$k_S = D/(2z)$, $k_T = Dz/2$
$l_S(v), l_T(v) \leftarrow 0$ for all $v \in V$ ▷ l stands for level
$d_S(v), d_T(v) \leftarrow 0$ for all $v \in V$ ▷ d stands for vertex degree in **its current** level
while stream not empty **do**
    $(u, v) \leftarrow$ next edge from stream
    if $l_S(u) \leq l_T(v)$ **then** $d_S(u) \leftarrow d_S(u) + 1$
    if $l_S(u) \geq l_T(v)$ **then** $d_T(v) \leftarrow d_T(v) + 1$
    if $d_S(u) \geq k_S$ **then**
        $l_S(u) \leftarrow l_S(u) + 1$
        $d_S(u) \leftarrow 0$
    if $d_T(v) \geq k_T$ **then**
        $l_T(v) \leftarrow l_T(v) + 1$
        $d_T(v) \leftarrow 0$
$S_i \leftarrow \{v : l_S(v) \geq i\}$ for all $0 \leq i \leq 2 \log_{1+\varepsilon} n$
$T_i \leftarrow \{v : l_T(v) \geq i\}$ for all $0 \leq i \leq 2 \log_{1+\varepsilon} n$
for $i = 1 \ldots 2 \log_{1+\varepsilon} n$ **do**
    if $|S_i| \geq z^2 |T_i|$ and $|S_i| \geq \frac{|S_{i-1}|}{1+\varepsilon}$ **then return** (S_i, T_i)
    if $|S_i| \leq z^2 |T_i|$ and $|T_i| \geq \frac{|T_{i-1}|}{1+\varepsilon}$ **then return** (S_i, T_i)

How can vertex degrees be leveraged to (approximately) simulate Algorithm 1? We observe that Algorithm 1 implicitly computes a vertex-vector l, where $l(v)$ is the peeling iteration after which

vertex v was removed from the graph. We refer to $l(v)$ as the *level* of v. Having l suffices to recover (S, T) in each iteration of the while-loop. Our approach approximates $l(v)$ for each vertex by using the evolution of vertex degrees as the stream is read. We now discuss details.

As a reminder, we think of an input graph as bipartite with the bipartitions S and T; details are described in Section 2. As our algorithm scans the edges in a stream, the maintained vertex degrees increase. Initially, each vertex level and vertex degree is 0. A vertex's level $l(v)$ increases when the algorithm is certain that v will not be deleted from the graph within the first $l(v)$ peeling iterations.

On a high level, once a vertex degree reaches $D/(2z)$ if it is in S or $Dz/2$ if it is in T, it will not be removed during the first peeling iteration. Therefore, we can confidently increase its level to 1. For a vertex in S to move to a level of at least 2, its degree must be at least $D/(2z)$ after the 1st peeling iteration. Unfortunately, before reading the entire stream, our single-pass algorithm cannot say with certainty that a vertex will be peeled in the 1st peeling iteration. However, if a vertex v is **not** peeled in the 1st iteration, an algorithm learns that information about v potentially before reading the entire stream. Inspired by this, we estimate the levels of vertices as follows.

For an edge (u, v) on the stream, we increase $d(u)$ only if $l(v) \geq l(u)$. This guarantees we only count edges where the other vertex has not been removed yet. Once a vertex moves to a higher level, we need to determine its degree at that level. However, it is not obvious how to calculate this degree without maintaining the edges in the subgraphs of each level. Therefore, in Algorithm 3, we assume the worst case and reset the degree of the vertex to 0 when it moves to a higher level. This poses challenges in analyzing whether these degree estimates are accurate enough to output an approximation of the directed densest subgraph, especially when the stream of edges is adversarial. Fortunately, we show that Algorithm 3 ran for specific input of D and z produces an $O(\log n)$-approximation. Precisely, we show the following result, whose proof is deferred to Appendix E.

Theorem 5.1. *There exists a single-pass semi-streaming algorithm that attains an $O(\log n)$-approximation of the directed densest subgraph.*

Remark. To adapt the algorithm to finding the *undirected* densest subgraph, we no longer need z. All other ideas remain the same, resulting in a single-pass semi-streaming algorithm for approximating the undirected densest subgraph as well. We also have a dynamic algorithm with $O(1)$ update time for each guess on D and z. We can maintain sets (S_i, T_i) throughout the algorithm without increasing the update time, resulting in $O(\log n)$ update time and $O(\log^2 n)$ update time for directed graphs. We can output our approximation of the densest subgraph anytime in $O(\text{poly} \log n)$ time.

6 Experiments

Baselines. We empirically evaluate the performance of our semi-streaming algorithm, comparing it to the $(2+\varepsilon)$-approximation $O(\log n)$ pass semi-streaming algorithm from [BKV12]. We also compare it to the single-pass semi-streaming algorithm from [MP24], but the density plots of [MP24] follow very similarly to [BKV12]. So, we only include [BKV12] in our approximation plots in Figure 2. As discussed in the introduction, [MP24] can have a quite high worst-case update time per edge on a stream, while the update time of our work is $O(\log^2 n)$. We illustrate this empirically in Figure 3.

To run these experiments, we use an M1 machine running macOS Sequoia 15.3.2 with 4 cores, 8 GB of RAM, 256 KB of L2 Cache, and 2.5 MB of L3 Cache (per core).

Data. We use 10 datasets from the Stanford Large Network Dataset Collection [LK14]. 5 of these datasets (Slashdot, Berkeley-Stanford Web, Google Web, Pokec, LiveJournal) are general directed graphs; most consist of edges sorted by endpoints. The other 5 datasets (Ask Ubuntu, Super User, Wikipedia, Twitter, Stack Overflow) are temporal directed graphs where edges are in sorted time order of when interactions happened. These temporal graphs reflect a stream of edge updates.

Results. We run the algorithms on temporal directed graphs with $\varepsilon = 0.2$. These datasets present events/edges in the order they occurred, making them an excellent benchmark for practical applications of dynamic algorithms. In Figure 2, we see that our algorithm matches [BKV12] for the largest density. We also observe that it is significantly less sensitive to error in z^2, making it better in practice than [BKV12] when our approximation of z^2 may not be as precise. Overall, we demonstrate that in practice our algorithm performs much better than the $\log n$ theoretical guarantee would imply.

Graph	Nodes	Edges
Slashdot	82,168	948,464
Google Web	875,713	5,105,039
Berk-Stan Web	685,230	7,600,595
Pokec	1,632,803	30,622,564
LiveJournal	4,847,571	68,993,773

(a) General directed graphs

Graph	Nodes	Edges
Ask Ubuntu	159,316	964,437
Super User	194,085	1,443,339
Wikipedia	1,140,149	7,833,140
Twitter	456,631	14,855,875
Stack Overflow	2,601,977	63,497,050

(b) Temporal directed graphs

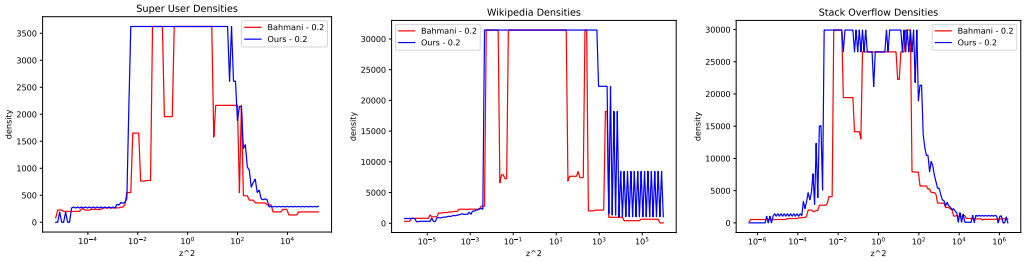

Figure 2: Density as a function of z^2 for various temporal datasets.

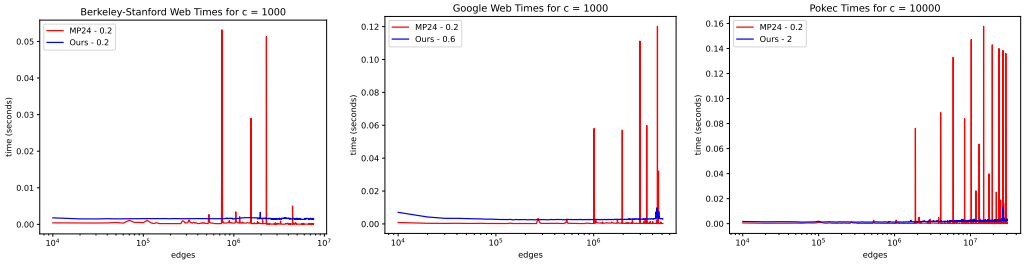

Figure 3: Update time between processing edges of the stream for various general datasets.

We also compare the update time of our algorithm to [MP24], using their threshold $f = 1/450$, in Figure 3. The plot contains update times of the algorithms for batches of 10,000 edges. The chosen c are where the largest densities are attained, and the chosen ε are based on maximizing densities. Note that ε does not significantly affect the running time of our algorithm. As we can see, the worst-case update time of our algorithm is significantly more stable and lower compared to [MP24].

Experimental results for the remaining datasets can be found in Appendix F. Additionally, we provide experiments for our MPC algorithm in Appendix G which show that our algorithm not only improves the theoretical upper bound on round complexity but also uses fewer rounds in practice.

7 Conclusion and future work

We study the directed densest subgraph problem in MPC and semi-streaming models. Our MPC algorithm bridges the gap between known algorithms for computing undirected and directed approximate DS. We also develop a simple deterministic single-pass semi-streaming algorithm. This is the first single-pass algorithm for the directed DS problem to achieve a sub-polynomial approximation.

Our work leaves a few intriguing questions. First, even though our MPC algorithm is able to match the round complexity of the state-of-the-art undirected DS algorithm, there is still a gap between their approximation factors, $1+\varepsilon$ compared to $2+\varepsilon$. Is it possible to improve the approximation guarantee for the directed DS problem while not increasing the round complexity? Second, can this upper bound of $\tilde{O}(\sqrt{\log n})$ MPC rounds in the sublinear memory regime be broken for either undirected or directed graphs? Third, our semi-streaming algorithm attains an $O(\log n)$-approximation. Can we develop a single-pass semi-streaming algorithm with an $\Theta(1)$-approximation for directed graphs?

Acknowledgments and Disclosure of Funding

S. Mitrović and T. Pan were supported by NSF CAREER award, No. 2340048. We are grateful to anonymous reviewers for their valuable feedback.

References

[ASS+18] Alexandr Andoni, Zhao Song, Clifford Stein, Zhengyu Wang, and Peilin Zhong. Parallel graph connectivity in log diameter rounds. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 674–685. IEEE, 2018.

[BGM14] Bahman Bahmani, Ashish Goel, and Kamesh Munagala. Efficient primal-dual graph algorithms for mapreduce. In *International Workshop on Algorithms and Models for the Web-Graph*, pages 59–78. Springer, 2014.

[BHNT15] Sayan Bhattacharya, Monika Henzinger, Danupon Nanongkai, and Charalampos Tsourakakis. Space-and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 173–182, 2015.

[BKV12] Bahman Bahmani, Ravi Kumar, and Sergei Vassilvitskii. Densest subgraph in streaming and mapreduce. *Proceedings of the VLDB Endowment*, 5(5), 2012.

[Cha00] Moses Charikar. Greedy approximation algorithms for finding dense components in a graph. In *International workshop on approximation algorithms for combinatorial optimization*, pages 84–95. Springer, 2000.

[CS10] Jie Chen and Yousef Saad. Dense subgraph extraction with application to community detection. *IEEE Transactions on knowledge and data engineering*, 24(7):1216–1230, 2010.

[CT22] Tianyi Chen and Charalampos Tsourakakis. Antibenford subgraphs: Unsupervised anomaly detection in financial networks. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 2762–2770, 2022.

[EHW15] Hossein Esfandiari, MohammadTaghi Hajiaghayi, and David P Woodruff. Applications of uniform sampling: Densest subgraph and beyond. *arXiv preprint arXiv:1506.04505*, 2015.

[FNBB06] Eugene Fratkin, Brian T Naughton, Douglas L Brutlag, and Serafim Batzoglou. Motifcut: regulatory motifs finding with maximum density subgraphs. *Bioinformatics*, 22(14):e150–e157, 2006.

[For10] Santo Fortunato. Community detection in graphs. *Physics reports*, 486(3-5):75–174, 2010.

[FRM19] András Faragó and Zohre R. Mojaveri. In search of the densest subgraph. *Algorithms*, 12(8):157, 2019.

[Gha17] Mohsen Ghaffari. Distributed mis via all-to-all communication. In *Proceedings of the ACM symposium on principles of distributed computing*, pages 141–149, 2017.

[GLM19] Mohsen Ghaffari, Silvio Lattanzi, and Slobodan Mitrović. Improved parallel algorithms for density-based network clustering. In *International Conference on Machine Learning*, pages 2201–2210. PMLR, 2019.

[GSZ11] Michael T Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, searching, and simulation in the mapreduce framework. In *International Symposium on Algorithms and Computation*, pages 374–383. Springer, 2011.

[GU19] Mohsen Ghaffari and Jara Uitto. Sparsifying distributed algorithms with ramifications in massively parallel computation and centralized local computation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1636–1653. SIAM, 2019.

[JZT+22] Yingsheng Ji, Zheng Zhang, Xinlei Tang, Jiachen Shen, Xi Zhang, and Guangwen Yang. Detecting cash-out users via dense subgraphs. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 687–697, 2022.

[LK14] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

[LRJA10] Victor E Lee, Ning Ruan, Ruoming Jin, and Charu Aggarwal. A survey of algorithms for dense subgraph discovery. *Managing and mining graph data*, pages 303–336, 2010.

[LW10] Christoph Lenzen and Roger Wattenhofer. Brief announcement: Exponential speed-up of local algorithms using non-local communication. In *Proceedings of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pages 295–296, 2010.

[MP24] Slobodan Mitrovic and Theodore Pan. Faster streaming and scalable algorithms for finding directed dense subgraphs in large graphs. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 35876–35891. PMLR, 2024.

[SHK+10] Barna Saha, Allison Hoch, Samir Khuller, Louiqa Raschid, and Xiao-Ning Zhang. Dense subgraphs with restrictions and applications to gene annotation graphs. In *Annual International Conference on Research in Computational Molecular Biology*, pages 456–472. Springer, 2010.

[SW20] Saurabh Sawlani and Junxing Wang. Near-optimal fully dynamic densest subgraph. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 181–193, 2020.

[ZP12] Yang Zhang and Srinivasan Parthasarathy. Extracting analyzing and visualizing triangle k-core motifs within networks. In *2012 IEEE 28th international conference on data engineering*, pages 1049–1060. IEEE, 2012.

[ZT12] Feng Zhao and Anthony KH Tung. Large scale cohesive subgraphs discovery for social network visual analysis. *Proceedings of the VLDB Endowment*, 6(2):85–96, 2012.

[ZZY+17] Si Zhang, Dawei Zhou, Mehmet Yigit Yildirim, Scott Alcorn, Jingrui He, Hasan Davulcu, and Hanghang Tong. Hidden: hierarchical dense subgraph detection with application to financial fraud detection. In *Proceedings of the 2017 SIAM international conference on data mining*, pages 570–578. SIAM, 2017.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly state our theoretical and experimental results as well as the assumptions of the algorithms.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The limitations are clearly stated in the statement of our results and are discussed in the conclusion.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: Our proofs of our theoretical results are written clearly and are all mathematically rigorous.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The parameters used for our algorithms are provided and the data ran for our experiments are cited. The implemented algorithms are the same as the pseudocode provided.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We provide access to the data and code in the supplemental material.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.

- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: All parameters used in our algorithms are provided. Also, the context and source of our data are provided.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Our implemented algorithms are deterministic and their results are completely determined by the input data, so there are no reportable errors in the results. We emphasize its statistical significance through using multiple data sets.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
   - If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: Our experiments compare approximation guarantees and time of execution of algorithms. We used a standard laptop whose characteristics are provided in the experimental section.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

Answer: [Yes]

Justification: There is no apparent harmful consequences of the results in this paper. Experiments are carried out in an ethical manner (no illegal data usage, etc.).

Guidelines:

- The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [No]

Justification: It is not obvious how the algorithms can be misused. Our results are more foundational research and we don't see any direct path to negative applications.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.

- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: There is no risk of misuse as data used was already publicly released and safe.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We credit the Stanford Large Network Dataset Collection where we got our data to run experiments.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: We are not releasing any new assets.

    Guidelines:

    - The answer NA means that the paper does not release new assets.

- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: Our research does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: Our research does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: No LLMs were used for the results in this paper.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (https://neurips.cc/Conferences/2025/LLM) for what should or should not be described.

A Proof of Lemma 3.1

Lemma 3.1 (Subgraph existence)**.** *Let (S^*, T^*) be a directed densest subgraph. Let $D = \rho(S^*, T^*)$ be its density and $z = \sqrt{|S^*|/|T^*|}$. There exists an induced subgraph H on vertex sets (S, T) for which it holds that $d_H(v) \geq D/(2z)$ for all $v \in S$ and $d_H(v) \geq Dz/2$ for all $v \in T$.*

Proof. We claim that the directed densest subgraph satisfies the constraints of such a subgraph H. Consider any vertex $v \in S^*$. Since H is the directed densest subgraph, removing v does not increase the density. Therefore, we have

$$\begin{aligned}
\frac{|E_G(S^*, T^*)|}{\sqrt{|S^*||T^*|}} &\geq \frac{|E_G(S^*, T^*)| - d_H(v)}{\sqrt{(|S^*| - 1)|T^*|}} \\
\implies d_H(v) &\geq \left(1 - \sqrt{\frac{|S^*| - 1}{|S^*|}}\right) |E_G(S^*, T^*)| \\
\implies d_H(v) &\geq \frac{|E_G(S^*, T^*)|}{2|S^*|} = \frac{D}{2z}
\end{aligned}$$

where for the last inequality, we used $(1 - 1/|S^*|)^{1/2} \leq 1 - 1/(2|S^*|)$. Similarly, we can show that for any vertex $v \in T^*$, it must satisfy $d_H(v) \geq Dz/2$.

Hence, the pair (S^*, T^*) satisfies the constraints of the claim. □

B Proof of Lemma 3.2

Lemma 3.2 (Sufficient condition for 2-approximation)**.** *Let (S^*, T^*) be a directed densest subgraph. Let $D = \rho(S^*, T^*)$ be its density and $z = \sqrt{|S^*|/|T^*|}$. Then, any induced subgraph H on vertex sets (S, T) which satisfies $d_H(v) \geq D/(2z)$ for all $v \in S$ and $d_H(v) \geq Dz/2$ for all $v \in T$ has density at least $D/2$. In other words, it is a 2-approximation of the directed densest subgraph.*

Proof. The amount of edges in H is lower bounded by $\max\left(|S| \cdot \frac{D^*}{2z}, |T| \cdot \frac{D^* z}{2}\right)$. Therefore, we have

$$\begin{aligned}
\rho(H) &\geq \frac{\max\left(|S| \cdot \frac{D^*}{2z}, |T| \cdot \frac{D^* z}{2}\right)}{\sqrt{|S||T|}} \\
&= \frac{D^*}{2} \cdot \max\left(\frac{1}{z}\sqrt{\frac{|S|}{|T|}}, z\sqrt{\frac{|T|}{|S|}}\right) \\
&\geq \frac{D^*}{2}
\end{aligned}$$

since the two parameters in $\max\left(\frac{1}{z}\sqrt{\frac{|S|}{|T|}}, z\sqrt{\frac{|T|}{|S|}}\right)$ are reciprocals of each other, implying one of them is at least 1. □

C Proof of Lemma 4.1

We will use the following lemma to connect degrees between the original graph and the sampled graph.

Lemma C.1. *Consider graph G, $k > 0$, $\varepsilon \in (0, 1)$. Let H be a subgraph of G obtained by sampling each edge of G independently with probability $p = \min\left(1, \frac{18 \log n}{\varepsilon^2 k}\right)$. Then with probability at least $1 - \frac{1}{n^2}$, for all $v \in G$, it holds:*

- *if $d_G(v) \leq \frac{k}{1+\varepsilon}$, then $d_H(v) < pk$;*

- *if* $d_G(v) \geq (1+\varepsilon)k$*, then* $d_H(v) > pk$*.*

Proof. If $p = 1$, then the lemma follows immediately. So, let $p = \frac{18 \log n}{\varepsilon^2 k}$. Consider any vertex $v \in G$. If $d_G(v) \leq k/(1+\varepsilon)$, then we consider two cases. When $d_G(v) \leq k/2$, we have

$$\begin{aligned}
\Pr[d_H(v) \geq pk] &\leq \exp\left(-\left(\frac{pk}{\mathbb{E}\left[d_H(v)\right]} - 1\right)\mathbb{E}\left[d_H(v)\right]/3\right) \\
&= \exp\left(-\frac{pk}{3} + \frac{\mathbb{E}\left[d_H(v)\right]}{3}\right) \\
&\leq \exp\left(-pk/6\right) \leq \frac{1}{n^3}
\end{aligned}$$

and when $k/2 < d_G(v) \leq k/(1+\varepsilon)$, we have

$$\begin{aligned}
\Pr[d_H(v) \geq pk] &\leq \exp\left(-\left(\frac{pk}{\mathbb{E}\left[d_H(v)\right]} - 1\right)^2\mathbb{E}\left[d_H(v)\right]/3\right) \\
&\leq \exp\left(-\varepsilon^2 pk/6\right) \leq \frac{1}{n^3}.
\end{aligned}$$

Therefore, using the union bound over all vertices, we have that $d_H(v) < pk$ for all $v \in G$, satisfying $d_G(v) \leq k/(1+\varepsilon)$, with probability at least $1 - \frac{1}{n^2}$. Now, if $d_G(v) \geq (1+\varepsilon)k$, then we have

$$\begin{aligned}
\Pr[d_H(v) \leq pk] &\leq \exp\left(-\left(1 - \frac{pk}{\mathbb{E}\left[d_H(v)\right]}\right)^2\mathbb{E}\left[d_H(v)\right]/2\right) \\
&\leq \exp\left(-\left(\frac{\varepsilon}{1+\varepsilon}\right)^2(1+\varepsilon)pk/2\right) \\
&\leq \exp\left(-\varepsilon^2 pk/6\right) \leq \frac{1}{n^3}.
\end{aligned}$$

Similarly, using the union bound over all vertices, we have that $d_H(v) > pk$ for all $v \in G$, satisfying $d_G(v) \geq (1+\varepsilon)k$, with probability at least $1 - \frac{1}{n^2}$. □

Lemma 4.1. *Let* $G = (S, T, E)$ *be a bipartite graph and* (S^*, T^*) *be its directed densest subgraph. Let* (S', T') *be the output of Algorithm 2 ran on* G *given* $\frac{\rho(S^*,T^*)}{(1+\varepsilon)^3} \leq D \leq \frac{\rho(S^*,T^*)}{(1+\varepsilon)^2}$ *and* $\frac{\sqrt{|S^*|/|T^*|}}{(1+\varepsilon)} \leq z \leq \sqrt{|S^*|/|T^*|}$*. Then, with probability at least* $1 - \frac{t}{n^2}$*, it holds that:*

- ***Good approximation.*** (S', T') *is a* $2(1+\varepsilon)^6$*-approximate densest subgraph of* G*, or*
- ***Size reduction.*** (S', T') *contains a* $2(1+\varepsilon)^6$*-approximate densest subgraph of* G *and*

$$|S'| \leq \frac{|S|}{\gamma} \text{ or } |T'| \leq \frac{|T|}{\gamma}$$

where $\gamma = (1+\varepsilon)^{\frac{\sqrt{\delta \log_{1+\varepsilon} n}}{8}}$.

Proof. Without loss of generality, assume the inputs of Algorithm 2 satisfy $\frac{|S|}{|T|} = cz^2$ for some $c > 1$. Now, if $f_1 \geq \frac{z\sqrt{|S||T|}}{\alpha}$, we use the high degrees of frozen vertices to see that

$$\rho(G) \geq \frac{f_1 k_S \alpha}{\sqrt{|S||T|}} \geq \frac{D}{2} \geq \frac{\rho(S^*, T^*)}{2(1+\varepsilon)^3}$$

meaning that G is a $2(1+\varepsilon)^3$-approximation of the densest subgraph. A similar argument can be made if $f_2 \geq \frac{\sqrt{|S||T|}}{z\alpha}$, so we assume that both of these conditions are not satisfied.

Now, if we look at the peeling process, it closely simulates Algorithm 1. Using Lemma C.1 and the constraints on D and z, we see that we're removing vertices in S with degree below $k_S/(1+\varepsilon)$ and keeping vertices in S with degree at least $(1+\varepsilon)k_S$ where

$$\frac{\rho(S^*,T^*)}{2(1+\varepsilon)^3\sqrt{|S^*||T^*|}} \leq k_S \leq \frac{\rho(S^*,T^*)}{2(1+\varepsilon)\sqrt{|S^*||T^*|}}$$

with probability at least $1-\frac{t}{n^2}$, using the union bound over all t iterations of peeling. Similarly, we see that we're removing vertices in T with degree below $k_T/(1+\varepsilon)$ and keeping vertices in T with degree at least $(1+\varepsilon)k_T$ where

$$\frac{\rho(S^*,T^*)}{2(1+\varepsilon)^4\sqrt{|S^*||T^*|}} \leq k_T \leq \frac{\rho(S^*,T^*)}{2(1+\varepsilon)^2\sqrt{|S^*||T^*|}}$$

also with probability at least $1-\frac{t}{n^2}$. Following the same argument behind Theorem 3.4 but using these bounds on k_S and k_T, we see that returning early will result in a $2(1+\varepsilon)^6$-approximation in total. Therefore, we will assume Algorithm 2 does not return early. Then, we consider two cases.

If $c \leq (1+\varepsilon)^{\sqrt{\delta \log_{1+\varepsilon} n}}$, then we see that

$$f_1 < \frac{z\sqrt{|S||T|}}{\alpha} \leq \frac{|S|}{(1+\varepsilon)^{\sqrt{\delta \log_{1+\varepsilon} n}}}$$

and

$$f_2 < \frac{\sqrt{|S||T|}}{z\alpha} \leq \frac{|T|}{(1+\varepsilon)^{\sqrt{\delta \log_{1+\varepsilon} n}/2}}.$$

Therefore, after t iterations of peeling, we have that the final returned vertex sets (S', T') satisfy

$$|S'| \leq \frac{|S|-(1+\varepsilon)f_1}{(1+\varepsilon)^{\lfloor t/2 \rfloor}} + (1+\varepsilon)f_1 \leq \frac{|S|}{(1+\varepsilon)^{\sqrt{\delta \log_{1+\varepsilon} n}/8}}$$

or

$$|T'| \leq \frac{|T|-(1+\varepsilon)f_2}{(1+\varepsilon)^{\lfloor t/2 \rfloor}} + (1+\varepsilon)f_2 \leq \frac{|T|}{(1+\varepsilon)^{\sqrt{\delta \log_{1+\varepsilon} n}/8}}$$

because of Line 16 and Line 17, using the upper bounds on f_1 and f_2, and sufficiently large n.

On the other hand, if $c > (1+\varepsilon)^{\sqrt{\delta \log_{1+\varepsilon} n}}$, then we still see that $f_1 \leq \frac{|S|}{(1+\varepsilon)^{\sqrt{\delta \log_{1+\varepsilon} n}}}$. So, if $|S|/|T| \geq z^2$ remains true throughout the whole algorithm, then due to Line 16 we have that the final returned vertex sets (S', T') satisfy

$$|S'| \leq \frac{|S|-(1+\varepsilon)f_1}{(1+\varepsilon)^{t}} + (1+\varepsilon)f_1 \leq \frac{|S|}{(1+\varepsilon)^{\sqrt{\delta \log_{1+\varepsilon} n}/8}}$$

following a similar argument as above. Otherwise, at some point we have that $|S|/|T| < z^2$, meaning that $|S|$ must have decreased by at least a factor of c. As a result, we also have that $|S'| \leq \frac{|S|}{(1+\varepsilon)^{\sqrt{\delta \log_{1+\varepsilon} n}/8}}$ in this case due to the lower bound on c. □

D Proof of Theorem 4.2

Theorem 4.2. *There exists a sublinear MPC algorithm that runs in $\tilde{O}(\sqrt{\log n})$ rounds and attains a $2(1+\varepsilon)^6$-approximation of the directed densest subgraph with probability at least $1-\frac{1}{n}$. The algorithm uses $O(n^\delta)$ memory per machine and $O(n^{1+\delta}+m)$ total memory for $\delta \in (0,1)$.*

Proof. Let $\mathcal{A}_{\text{MPC}}$ be our MPC algorithm. We first describe $\mathcal{A}_{\text{MPC}}$ and then provide its analysis.

Algorithm description. $\mathcal{A}_{\text{MPC}}$ considers all guesses on $1 \leq D \leq n$ and $\frac{1}{\sqrt{n}} \leq z \leq \sqrt{n}$ in parallel using powers of $1+\varepsilon$. For each of these parallel instances of D and z,

- The algorithm invokes Algorithm 2 $16\sqrt{\delta \log_{1+\varepsilon} n}/\delta$ times, constantly providing back the output of the previous invocation, with the edges of the induced subgraph of the new vertex sets, into the new invocation.
- If Algorithm 2 ever returns early in any of these invocations, then we call the pair of vertex sets that is returned early a pair of *potential vertex sets*.

$\mathcal{A}_{\text{MPC}}$ outputs the pair of potential vertex sets with the largest density over all instances of D and z.

Algorithm memory and round complexity. Using the same application of the graph exponentiation technique shown in [GLM19, Appendix C.2], we see that each machine will only use $O(n^\delta)$ memory since neighborhoods will have size bounded by

$$\left(\frac{36 \log n}{\varepsilon^2} \cdot \alpha\right)^t \in O(n^\delta)$$

with high probability and Algorithm 2 will take $O(\log \log n)$ rounds. Including the number of invocations to Algorithm 2, we have that $\mathcal{A}_{\text{MPC}}$ takes $O(\sqrt{\log n} \cdot \log \log n)$ rounds. Running the algorithm in parallel over all instances of D and z adds an $O(\log^2 n)$ factor to the memory per machine.

Algorithm approximation. Note that there will be guesses of D and z that satisfy

$$\frac{\rho(S^*,T^*)}{(1+\varepsilon)^3} \leq D \leq \frac{\rho(S^*,T^*)}{(1+\varepsilon)^2} \text{ and } \frac{\sqrt{|S^*|/|T^*|}}{(1+\varepsilon)} \leq z \leq \sqrt{|S^*|/|T^*|}.$$

Using these guesses and the approximation guarantees from Lemma 4.1, we guarantee these potential vertex sets to be a $2(1+\varepsilon)^6$-approximation with at least probability $1 - \frac{1}{n}$, taking the union bound over the invocations of Algorithm 2. Since $\mathcal{A}_{\text{MPC}}$ selects the vertex sets with the largest density, it attains at least a $2(1+\varepsilon)^6$-approximation of the directed densest subgraph. □

E Proof of Theorem 5.1

Lemma E.1. *Let the directed densest subgraph have vertex sets (S^*, T^*). Then, (S_i, T_i) in Algorithm 3 will be non-empty vertex sets for all $0 \leq i \leq 2\log_{1+\varepsilon} n$ given $D \leq \frac{\rho(S^*,T^*)}{8(1+\varepsilon)\log_{1+\varepsilon} n}$ and $\frac{\sqrt{|S^*|/|T^*|}}{(1+\varepsilon)} \leq z \leq \sqrt{|S^*|/|T^*|}$.*

Proof. Consider (S_i, T_i) after running Algorithm 3. Then, in the best-case scenario, all the edges in its induced subgraph, $E_G(S_i, T_i)$, were used to determine the peeling of its vertices for later vertex sets. However, because the degree of a vertex v is reset to 0 every time v moves to a higher level, information about edges incident to v up to that point is – informally speaking – erased; we call such edges *ignored*. At most $i \cdot k_S$ incident edges to a vertex in S_i are ignored, and, similarly, at most $i \cdot k_T$ incident edges to a vertex in T_i are ignored. Hence, throughout the entire algorithm, each vertex in $S_{2\log_{1+\varepsilon} n}$ ignores at most $2k_S \log_{1+\varepsilon} n$ edges incident to it and each vertex in $T_{2\log_{1+\varepsilon} n}$ ignores at most $2k_T \log_{1+\varepsilon} n$ edges incident to it.

Consider the induced subgraph on (S^*, T^*). Since (S^*, T^*) is the densest subgraph, we observe that all vertices in S^* have degree at least $8k_S \log_{1+\varepsilon} n$ and all vertices in T^* have degree at least $8k_T \log_{1+\varepsilon} n$ by following the proof of Lemma 3.1 and using the bounds on D and z. We claim that some non-empty subsets of these vertex sets will remain in $(S_{2\log_{1+\varepsilon} n}, T_{2\log_{1+\varepsilon} n})$, which is enough to prove the lemma. Rather than letting edges be ignored throughout the algorithm, we assume they're all ignored *at the beginning*: $2k_S \log_{1+\varepsilon} n$ edges incident to each vertex in S^* are ignored and $2k_T \log_{1+\varepsilon} n$ edges incident to each vertex in T^* are ignored. Then, we look at how vertices within the induced subgraph on (S^*, T^*) are peeled over time.

Notice that

$$|E_G(S^*, T^*)| \geq (8k_S \log_{1+\varepsilon} n)|S^*| = (8k_T \log_{1+\varepsilon} n)|T^*|$$

and so with the edges that are ignored, the number of remaining edges is lower bounded by

$$\begin{aligned}&|E_G(S^*, T^*)| - (2k_S \log_{1+\varepsilon} n)|S^*| - (2k_T \log_{1+\varepsilon} n)|T^*| \\ &\geq (4k_S \log_{1+\varepsilon} n)|S^*| = (4k_T \log_{1+\varepsilon} n)|T^*|.\end{aligned}$$

When peeling is performed on this subgraph, all the vertices that are peeled each cause at most k_S edges to be removed, if it is from S^*, and at most k_T edges to be removed, if it is from T^*. If we assume that all the vertices are peeled, at most $2k_S|S^*| = 2k_T|T^*|$ edges will be removed, and the graph must be empty. However, this is smaller than the lower bound on the number of edges above, so it is impossible to peel all the vertices. Therefore, we end up with a non-empty subgraph, and the vertices in this subgraph will not be removed throughout the entire algorithm. □

Theorem 5.1. *There exists a single-pass semi-streaming algorithm that attains an $O(\log n)$-approximation of the directed densest subgraph.*

Proof. Let $\mathcal{A}_{\text{STREAM}}$ be our semi-streaming algorithm. We now describe it and provide its analysis.

Algorithm description. $\mathcal{A}_{\text{STREAM}}$ runs Algorithm 3 on all $1 \leq D \leq n$ and $\frac{1}{\sqrt{n}} \leq z \leq \sqrt{n}$ in parallel using powers of $1+\varepsilon$. Then, out of all the outputs of Algorithm 3 where the vertex sets are both non-empty, we pick the one corresponding to the largest D as the final output of $\mathcal{A}_{\text{STREAM}}$.

Algorithm memory. Note that Algorithm 3 does not need to store (S_i, T_i) for all $0 \leq i \leq 2\log_{1+\varepsilon} n$ but only needs to store consecutive sets to compare their sizes. Therefore, Algorithm 3 uses $O(n)$ memory through this implementation detail. Additionally, we have $O(\log^2 n)$ total guesses on D and z. Each of these guesses is a copy of all the variables in Algorithm 3, resulting in $O(n \log^2 n)$ memory in total.

Algorithm approximation. Let the directed densest subgraph have vertex sets (S^*, T^*). Note that because we only update $d(u)$ for an edge (u, v) or (v, u) with $l(u) < l(v)$, this ensures that a vertex that is not removed must have degree at least $D/(2z)$, if it is in S, and degree at least $Dz/2$, if it is in T. Therefore, if Algorithm 3 outputs non-empty vertex sets for $D \geq \frac{\rho(S^*,T^*)}{8(1+\varepsilon)^2 \log_{1+\varepsilon} n}$, we follow a similar argument as the proof of Algorithm 1 to see that these vertex sets would be an $O(\log n)$-approximation of the directed densest subgraph.

Specifically, there exists a guess on D and z such that

$$\frac{\rho(S^*, T^*)}{8(1+\varepsilon)^2 \log_{1+\varepsilon} n} \leq D \leq \frac{\rho(S^*, T^*)}{8(1+\varepsilon) \log_{1+\varepsilon} n} \text{ and } \frac{\sqrt{|S^*|/|T^*|}}{(1+\varepsilon)} \leq z \leq \sqrt{|S^*|/|T^*|}.$$

Using Lemma E.1, we know that all vertex sets (S_i, T_i) in Algorithm 3 for this guess on D and z will be non-empty and so Algorithm 3 will output non-empty vertex sets. Since $\mathcal{A}_{\text{STREAM}}$ outputs the non-empty vertex sets corresponding to the largest D, they must satisfy $D \geq \frac{\rho(S^*,T^*)}{8(1+\varepsilon)^2 \log_{1+\varepsilon} n}$ and results in an $O(\log n)$-approximation. □

F Additional streaming experiments

We run the algorithms on the general directed graphs. We plot the results of our algorithm run on various values of ε ranging from 0.2 to 2 while we ran [BKV12] on $\varepsilon = 0.2$. Even though most of these datasets have edges sorted by their endpoints, which can be considered adversarial for our algorithm, we see from Figure 4 that their approximations are not far off from [BKV12]. Our maximum densities are within a factor of around 2 from [BKV12], which is much closer than our theoretical guarantees. We also consider a less adversarial order where we randomize the order of the edges. With a randomized order, we also see in Figure 4 that the computed densities increase significantly, almost matching those of [BKV12]. Figure 5 contains the update time plots for the remaining general datasets.

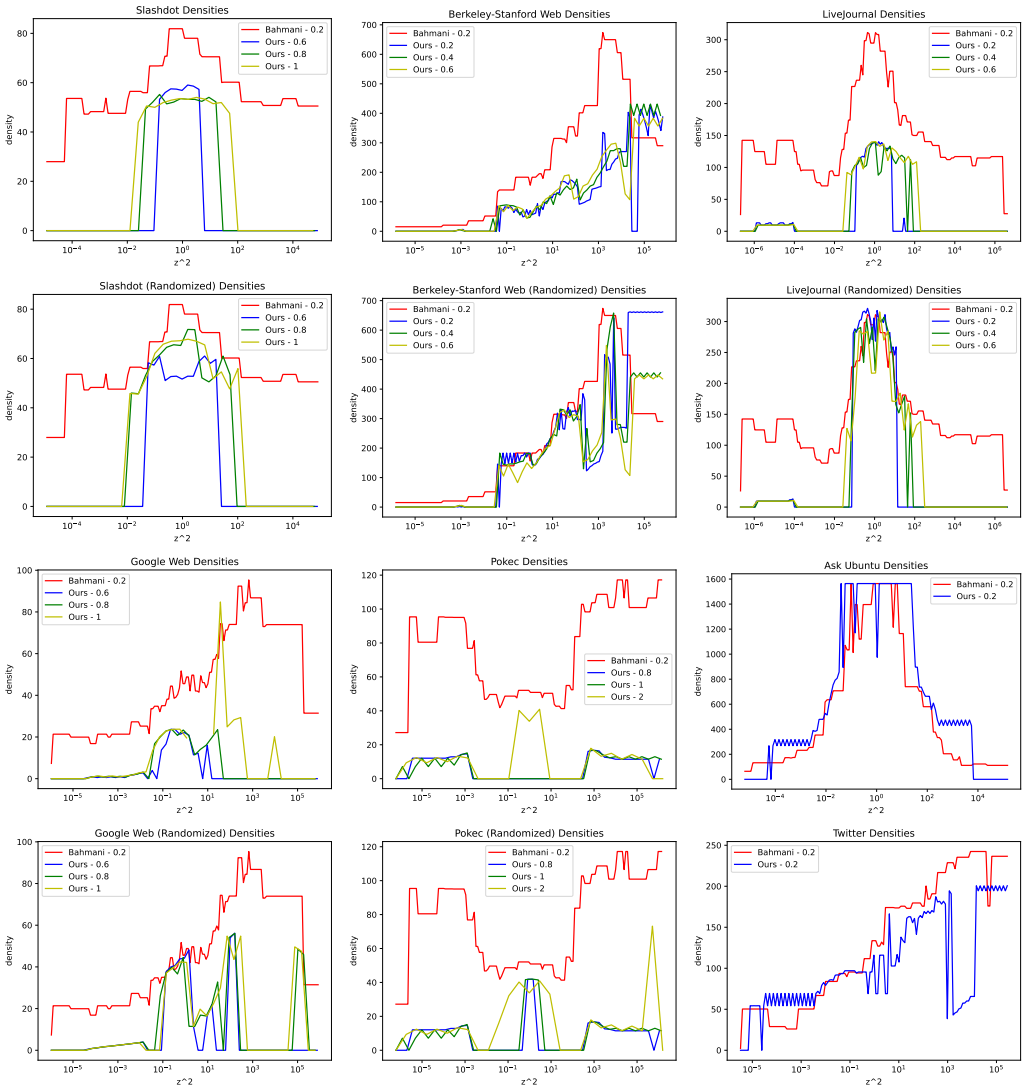

Figure 4: Density as a function of z^2 for general datasets and remaining temporal datasets.

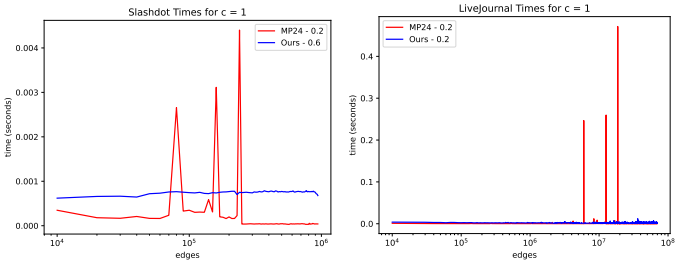

Figure 5: Update time between processing edges of the stream for remaining general datasets.

G MPC experiments

We compare our MPC algorithm to the MPC algorithm for directed graphs from [BKV12]. Specifically, we look at their density plots as well as how many sublinear MPC rounds the algorithms take. We use $\delta = 0.4, 0.6, 0.8$, where each machine has n^δ memory, and $\varepsilon = 0.6$, the approximation parameter for both algorithms. As we can see from Figure 6, our algorithm attains densities that

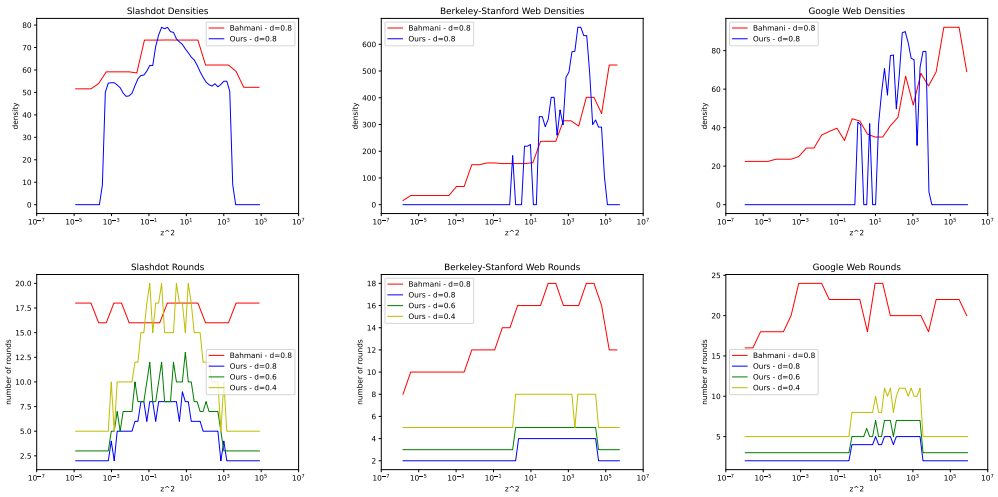

Figure 6: Density and number of MPC rounds as a function of z^2 for $\delta = 0.4, 0.6, 0.8$, $\varepsilon = 0.6$. The used datasets are described in Section 6.

match [BKV12]. Additionally, it does so using less than half the number of rounds when having the same amount of memory. It continues to use significantly less rounds for smaller amounts of memory and only matches the number of rounds for the Slashdot dataset where our algorithm uses a quadratically smaller amount of memory per machine compared to what is used by [BKV12]. This is a significant improvement in the number of rounds in practice and reflects our improvement in the algorithm's theoretical upper bound.

Remark. These experiments were not executed on actual large-scale frameworks, but simulated in a centralized setting. The rounds in plots correspond to the number of rounds taken in those MPC simulations.