Faster Streaming and Scalable Algorithms for Finding Directed Dense Subgraphs in Large Graphs

Slobodan Mitrović [1] **Theodore Pan** [1]

Abstract

Finding dense subgraphs is a fundamental algorithmic tool in data mining, community detection, and clustering. In this problem, the aim is to find an induced subgraph whose edge-to-vertex ratio is maximized. We show how to find a $(2+\epsilon)$ approximation of the *directed* densest subgraph on randomized streams in a single pass while using $O(n \cdot \operatorname{poly}\log n)$ memory on n-vertex graphs. In contrast, the approach by Bahmani et al. (VLDB 2012) uses $O(\log n)$ passes and by Esfandiari et al. (2015) makes one pass but uses $O(n^{3/2})$ memory; both algorithms also apply to arbitrary-ordered streams. Our techniques extend to Massively Parallel Computation (MPC), yielding quadratic improvement over state-of-the-art by Bahmani et al. (VLDB 2012 and WAW 2014). We empirically show that the quality of our output is essentially the same as that of Bahmani et al. (VLDB 2012) while being 2 times faster on large graphs, even on non-randomly ordered streams.

1. Introduction

Given a directed graph $G = (V, E)$, the directed *densest subgraph problem* asks to find two vertex subsets $S, T \subseteq V$, not necessarily disjoint, such that the number of edges from S to T scaled by $\sqrt{|S| \cdot |T|}$ is maximized. When $S = T$, this problem is equivalent to finding the densest subgraph in undirected graphs. Dense subgraph discovery is a fundamental algorithmic tool in data mining (Kriegel & Pfeifle, 2005; Wu et al., 2019; Fang et al., 2022), community detection (Chen & Saad, 2010; Harenberg et al., 2014), spam detection (Leon-Suematsu et al., 2011; Zhang et al., 2016), fraud discovery (Zhang et al., 2017; Ren et al., 2021), clustering (Kriegel et al., 2011; Bhattacharjee & Mitra, 2021), graph compression (Buehrer & Chellapilla, 2008), and many other applications. Specifically for unsupervised learning, we observe multiple practical algorithms that implement dense subgraph discovery in finance, web graphs, or computational neuroscience just to name a few (Chen & Tsourakakis, 2022; Mitzenmacher et al., 2015; Chen et al., 2022). Given its importance, this problem has been studied in various computational settings, including semi-streaming (Bahmani et al., 2012; McGregor et al., 2015; Bhattacharya et al., 2015), dynamic (Epasto et al., 2015; Sawlani & Wang, 2020), distributed (Su & Vu, 2019), and parallel (Bahmani et al., 2012; Ghaffari et al., 2019), with the first study of its undirected version dating back to the eighties (Goldberg, 1984).

A directed densest subgraph can be found in polynomial time (Charikar, 2003). However, the corresponding algorithm solves a family of $O(|V|^2)$ linear programs, making it impractical for execution on large graphs. Nevertheless, much faster approximation algorithms have been developed. For instance, there exists a simple greedy algorithm running in near-linear time that yields a $2 + \varepsilon$ approximation for any arbitrary constant $\varepsilon > 0$ (Bahmani et al., 2012; Charikar, 2003). The prevalence of large graphs has motivated researchers to study the approximate densest subgraph problem in settings for processing big data, such as *streaming* and *massively parallel computation (MPC)*.

There has been a significant interest in designing semi-streaming algorithms for finding dense subgraphs, e.g., (Bahmani et al., 2012; Esfandiari et al., 2015; Bhattacharya et al., 2015; McGregor et al., 2015) and references therein. (Esfandiari et al., 2015) designed a single-pass semi-streaming algorithm for finding $(1 + \varepsilon)$-approximate densest subgraph in *undirected* graphs. For n-vertex graph G, this algorithm uniformly at random samples $O(n \cdot \log(n)/\varepsilon^2)$ edges from G in the streaming fashion. The authors show that the densest subgraph in such a sample is a $(1 + \varepsilon)$-approximate densest subgraph in G. This sampling idea fails in the context of directed graphs. To see that, consider the example in Figure 1. In that example, the directed densest subgraph is the star, having density $\Theta(\sqrt{n})$; each clique has density $\Theta(n^{0.499})$. How-

[1]Department of Computer Science, University of California, Davis, CA. Correspondence to: Slobodan Mitrović <smitrovic@ucdavis.edu>, Theodore Pan <thjpan@ucdavis.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

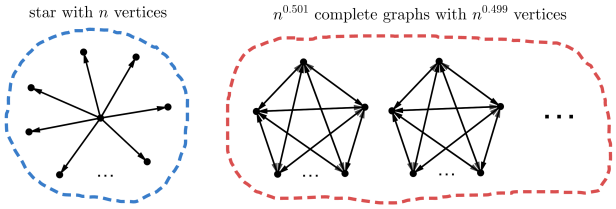

Figure 1. An illustration of why uniformly at random sampling applicable to undirected fails in the context of directed densest subgraph.

ever, sampling $O(n \cdot \log n)$ edges from that graph will, in expectation, contain only $O(n^{0.501})$ edges from the star, which is insufficient to recover the densest subgraph in Figure 1. To alleviate this, (Esfandiari et al., 2015) propose a single-pass semi-streaming algorithm that requires $O(n^{1.5} \operatorname{poly} \log n)$ memory. In another line of work, (Bahmani et al., 2012), the memory requirement is kept at $O(n \operatorname{poly} \log n)$ at the expense of making $O(\log n)$ passes and a $(2+\varepsilon)$-approximation. It is unknown whether these techniques can be extended to achieve the best of both worlds, i.e., a single pass and $O(n \operatorname{poly} \log n)$ memory.

Finding dense subgraphs in MPC has also received significant attention. For undirected graphs, the sampling strategy devised in (Esfandiari et al., 2015) for the streaming setting readily transfers to an $O(1)$ MPC round algorithm in the near-linear memory regime. For the sublinear memory regime, (Bahmani et al., 2014) developed an algorithm that in $O(\log(n)/\varepsilon^2)$ rounds outputs a $(1+\varepsilon)$ approximation. This was improved to $O(\sqrt{\log n} \cdot \log \log n)$ rounds by (Ghaffari et al., 2019). However, state-of-the-art algorithms for densest subgraphs have significantly higher round complexity for directed than for undirected graphs. The works (Bahmani et al., 2012) and (Bahmani et al., 2014) also design an algorithm for finding $(2+\varepsilon)$- and $(1+\varepsilon)$-approximate densest subgraphs in directed graphs in $O(\log(n)/\varepsilon)$ and $O(\log(n)/\varepsilon^2)$ MPC rounds, respectively. To the best of our knowledge, this is the most efficient MPC approach, even for the super-linear memory regime, leaving a considerable gap between the case of directed and undirected graphs: $O(\log n)$ vs. $O(1)$ round complexity. Given that many modern graphs are directed – examples include Twitter and Instagram "following" relationships, links on the Web, etc. – *it is natural to wonder whether faster algorithms for finding directed densest subgraphs can be developed.*

1.1. Our contributions

As our first result, we make progress in the semi-streaming setting.

Result 1 (Theorem 4.8 rephrased)**.** *Given an n-vertex graph, there exists a single-pass semi-streaming algorithm that, with high probability, over randomized streams outputs a $(2+\varepsilon)$-approximate directed densest subgraph while using $O(n \operatorname{poly}(\log(n)/\varepsilon))$ memory.*

We build on ideas we develop for Result 1 and improve the state-of-the-art in the context of MPC as well.

Result 2 (Theorems 5.1 and 5.2 summarized)**.** *Given an n-vertex graph, there exists an MPC algorithm that, with high probability, outputs a $(2+\varepsilon)$-approximate directed densest and*

- *when the memory per machine is $n^{1+\delta}$, for any constant $\delta > 0$, the algorithm runs in $O(1)$ rounds;*
- *when the memory per machine is $n \operatorname{poly}(\log(n)/\varepsilon)$, the algorithm runs in $O(\sqrt{\log n})$ rounds.*

The fastest previously known MPC algorithms require $O(\log n)$ rounds even when the memory per machine is $n^{1+\delta}$ (Bahmani et al., 2012; 2014). So, our approach makes significant progress toward closing the gap between the undirected and directed cases.

Moreover, our empirical evaluations show that our semi-streaming algorithm is at least 2 times faster than (Bahmani et al., 2012) while achieving essentially the same accuracy. It is interesting to note that our algorithm is by several percent more accurate than (Bahmani et al., 2012) on large graphs. These hold even when the underlying stream is non-randomized. Also, for our MPC algorithm with $n \operatorname{poly}(\log(n)/\varepsilon)$ memory per machine, our evaluations show that it uses at least half the number of phases compared to (Bahmani et al., 2012).

2. Preliminaries

Notation. Given a directed graph $G = (V, E)$, we use E_G to denote E, the set of edges specific to graph G. We use n to refer to $|V|$. Given two vertex sets $S, T \subseteq V$, we refer to the edges between them by $E_G(S,T) \stackrel{\text{def}}{=} \{e = (i,j) \in E_G : i \in S, j \in T\}$. For the sake of brevity, for $i, j \in V$, we also write $E_G(i,T) \stackrel{\text{def}}{=} E_G(\{i\},T)$ and $E_G(S,j) \stackrel{\text{def}}{=} E_G(S,\{j\})$. Specific to streams, we use $E_{\text{stream}}(S,T)$ to denote the subset of $E_G(S,T)$ remaining in the stream. For vertex $v \in V$, we use $d_G^-(v)$ to denote the in-degree and $d_G^+(v)$ to denote the out-degree of v in G.

When we say that an event A happens *with high probability*, or whp for short, we imply that $\Pr[A] \geq 1 - n^{-c}$, for a constant $c > 0$. In our algorithms, the constant c can be made arbitrarily large by paying constant factors in the pass/round or memory complexity.

Directed densest subgraph. Given directed graph $G = (V, E)$ 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 *densest subgraph* is sets S^*, T^* such that $(S^*, T^*) \in \arg\max_{S,T \subseteq V} \rho(S, T)$.

Semi-streaming. In the semi-streaming model, an algorithm scans input data, e.g., scans edge by edge of an input graph. After all the edges are scanned, we say that the algorithm made a *pass*. When the edges of the graph are presented to the algorithm in a random permutation, we say that the stream is *randomized*. Throughout the process, for n-vertex graph, the algorithm is typically allowed to use $O(n \operatorname{poly} \log n)$ memory. The complexity measure in this setup is the number of passes the algorithm makes and the memory it requires; as discussed for (Esfandiari et al., 2015), some approaches require polynomially more than memory n. After all the passes are performed, the algorithm outputs a solution.

Massively Parallel Computation (MPC). The Massively Parallel Computation (MPC) model has become a standard in the theoretical study of large-scale frameworks (Karloff et al., 2010; Goodrich et al., 2011; Beame et al., 2017) such as MapReduce, Hadoop, Spark, and Flume. In MPC, the computation proceeds in synchronous rounds across N machines. Each machine has S words of memory. Initially, input data is partitioned arbitrarily across the machines. During a round, each machine performs computation on its data locally. At the end of a round, machines exchange messages. The communication topology among the machines is a clique. During a round, a machine can, in total, send and receive at most S bits of data. There are three important regimes with respect to S relative to data size. Let the input be an n-vertex graph and $\delta \in (0, 1)$ be an arbitrary constant: *sub-linear* corresponds to $S = n^\delta$; *near-linear* corresponds to $S = n \operatorname{poly} \log n$; and, $S = n^{1+\delta}$ corresponds to *super-linear*.

Probability tools. In our analysis, we extensively apply the following well-known tool from probability.

Theorem 2.1 (Chernoff bound)**.** *Let $X_1, \ldots, X_k$ be independent random variables taking values in $[0, 1]$. Let $X \stackrel{\text{def}}{=} \sum_{i=1}^{k} X_i$ and $\mu \stackrel{\text{def}}{=} \mathbb{E}[X]$. Then,*

(A) For any $\delta \in [0, 1]$ it holds $\Pr[|X - \mu| \geq \delta\mu] \leq 2 \exp\left(-\delta^2 \mu/3\right)$.

(B) For any $\delta \in [0, 1]$ it holds $\Pr[X \geq (1 + \delta)\mu] \leq \exp\left(-\delta^2 \mu/3\right)$.

3. The Base Algorithm

Let c be a value such that there exists a densest subgraph (S, T) in G and $c = |S|/|T|$. The starting point of our approach is a classical peeling algorithm for the directed densest subgraph problem. This method receives c as input and iteratively "peels", i.e., removes, vertices that are below a certain degree threshold. Intuitively, the peeling is performed so that the removed vertices do not significantly affect the densest subgraph. We make this statement formal by Theorem 4.3. One iteration of this peeling primitive is given by VSETS-UPDATE (Algorithm 1).

Algorithm 1 (VSETS-UPDATE)

Input: $G = (V, E)$, $c > 0$, $\epsilon \in (0, 1)$, and vertex sets S, T

if $|S|/|T| \geq c$ **then**
  $A(S) \leftarrow \{i \in S : |E(i, T)| \leq (1 + \epsilon)\frac{|E(S,T)|}{|S|}\}$
  $S \leftarrow S \backslash A(S)$
else
  $B(T) \leftarrow \{j \in T : |E(S, j)| \leq (1 + \epsilon)\frac{|E(S,T)|}{|T|}\}$
  $T \leftarrow T \backslash B(T)$
end if
return (S, T)

By iteratively peeling and keeping track of the vertex sets that produce the densest subgraph, we are able to find an approximation of the directed densest subgraph. VSETS-UPDATE has two handy features. The first one is that, given two sets S and T, it reduces the size of one of them by $1+\varepsilon$.

Lemma 3.1. *Let $G = (V, E)$ be a directed graph. For $i \geq 0$, let (S_i, T_i) be a sequence of set-pairs such that $(S_0, T_0) = (V, V)$ and $(S_{i+1}, T_{i+1}) =$ VSETS-UPDATE$(G, c, \epsilon, S_i, T_i)$. Then $S_t = \emptyset$ or $T_t = \emptyset$ for some $t \in O(\log(n)/\epsilon)$.*

Proof. Consider VSETS-UPDATE invoked with $|S_i|/|T_i| \geq c$, so vertices are removed from S_i. Then, all vertices $u \in S_i \backslash A(S_i)$ satisfy $|E(u, T)| > (1+\epsilon)\frac{|E(S_i,T_i)|}{|S_i|}$. Since $S_{i+1} = S_i \backslash A(S_i)$, we have that $|S_{i+1}| < |S_i|/(1+\epsilon)$. Similarly, if $|S_i|/|T_i| < c$, we have that $|T_{i+1}| < |T_i|/(1 + \epsilon)$. So, the size of one of the sets decreases by at least $1 + \epsilon$ times each time VSETS-UPDATE is invoked. Then, one of the sets becomes empty after $O(\log_{1+\epsilon} n) = O(\log(n)/\varepsilon)$ invocations of VSETS-UPDATE. □

The second useful feature of VSETS-UPDATE is that one of the pairs (S_i, T_i), as defined in the statement of Lemma 3.1, is a $(2 + \epsilon)$-approximate densest subgraph. We prove such a statement in Theorem 4.3.

4. $(2+\epsilon)$ Approximation in a Single Pass

4.1. A Multi-pass Sampling-based Algorithm

We modify the base algorithm to sample edges to determine which vertices to peel, instead of using the entire graph for that. By sampling a sufficient number of edges, the degrees of vertices in the sample are used to estimate the degrees of vertices in the original graph. For each pass, Algorithm 2 creates a new sample of our graph to apply VSETS-UPDATE to. In our analysis, we extensively use the following threshold

$$\xi \stackrel{\text{def}}{=} \frac{60 \log n}{\epsilon^2}. \tag{1}$$

Algorithm 2 A semi-streaming directed densest subgraph algorithm

Input: $G = (V, E)$, $c > 0$, and $\epsilon \in (0, 1)$
Output: A $(2 + \varepsilon)$-approximate directed densest subgraph

1: Initialize each of S, T, S^*, T^* to V
2: **while** $S \neq \emptyset$ and $T \neq \emptyset$ **do**
3: Let H be a sample of $E_G(S,T)$ with each edge sampled independently with probability $\min\left\{\frac{n\xi}{(1-\epsilon)|E_G(S,T)|}, 1\right\}$
4: $(S, T) \leftarrow$ VSETS-UPDATE(H, c, ϵ, S, T)
5: **if** $\rho(S,T) > \rho(S^*, T^*)$ **then**
6: $S^* \leftarrow S, T^* \leftarrow T$
7: **end if**
8: **end while**
9: **return** S^*, T^*

Before we begin analyzing Algorithm 2, we prove Lemma 4.1 as a tool that connects the degrees of vertices of graph G and sampled subgraph H.

Lemma 4.1. *Let $G = (V, E)$ be a graph, $\epsilon \in (0, 1)$, and $p \in (0, 1]$. Let H be a subgraph of G that contains each edge of G independently with probability p. Then, for all $v \in G$, the following hold with probability $1 - \frac{1}{n^3}$: (i) if $d^-_G(v) \geq \xi/p$ then $pd^-_G(v) \leq (1+\epsilon)d^-_H(v)$, (ii) if $d^-_G(v) < \xi/p$ then $d^-_H(v) < 2\xi$ (same claims hold for out-degrees $d^+_G(v)$ and $d^+_H(v)$).*

The proof of Lemma 4.1 is given in Appendix A. Now, we begin analyzing Algorithm 2.

Lemma 4.2. *Algorithm 2 uses $O\left(\frac{n \log^2(n)}{\epsilon^3}\right)$ memory with probability $1 - \frac{\log_{1+\epsilon} n}{n^4}$.*

Proof. By the Chernoff bound, Theorem 2.1 (B), every pass of Algorithm 2 samples $O\left(\frac{(1+\epsilon)n\xi}{(1-\epsilon)}\right) = O\left(\frac{n \log n}{\epsilon^2}\right)$ edges with probability $1 - \frac{1}{n^4}$. From Lemma 3.1, the algorithm makes $O(\log_{1+\epsilon} n)$ passes. Therefore, by the union bound over all the passes, the total memory used by Algorithm 2 is $O\left(n\xi \cdot \log_{1+\epsilon} n\right) = O\left(\frac{n \log^2 n}{\epsilon^3}\right)$ with probability $1 - \frac{\log_{1+\epsilon} n}{n^4}$. □

Theorem 4.3. *Algorithm 2 produces a $2(1+\epsilon)$-approximation of the densest directed subgraph with probability $1 - \frac{\log_{1+\epsilon} n}{n^2}$.*

The proof of Theorem 4.3 is given in Appendix B and mainly follows a proof from (Charikar, 2003) with additional applications of the Chernoff bound and Lemma 4.1.

4.2. Improved Single-pass Algorithm

Building on Algorithm 2, we develop a single-pass semi-streaming algorithm for $(2+\varepsilon)$-approximate directed densest subgraph when the stream of edges is randomized. The main idea behind our algorithm is to leverage the additional randomization of the stream and avoid performing a new pass to estimate vertex degrees when S or T gets updated; these degrees are needed to execute VSETS-UPDATE.

Let $(S', T') =$ VSETS-UPDATE$(H, c, \varepsilon, S, T)$. Observe that $|E(u, T')|$ and $|E(S', v)|$, for $u \in S'$ and $v \in T'$, can be estimated by uniformly at random sampling $O(n\xi)$ edges from $E(S', T')$. *How can those edges be sampled without performing a new pass?* If our algorithm has not already seen any edge in $E(S', T')$ from the stream, this task would be easy – collect edges in $E(S', T')$ from the stream as long as fewer than the desired many are seen. However, our algorithm already has a subset E' of $E(S', T')$ in its memory, which should be considered. Assuming that the sampling probability is known – which is not known and we discuss it in a moment – then SET-SAMPLE (Algorithm 3) outputs the desired sample while considering E'.

Algorithm 3 (SET-SAMPLE)

Input: A set of edges $E' \subseteq E_G(S,T)$, vertex sets S and T, probability p, an estimate s of $|E_G(S,T)|$
Output: Approximately uniform random sample of edges from the union of the stream and E'

1: Let H_1 be a subset of E', where each in E' is included in H_1 with probability p.
2: Sample x from the binomial distr. $B(s - |E'|, p)$.
3: Set H_2 to next x edges in $E_G(S,T)$ from the stream.
4: **return** $H_1 \cup H_2$

SET-SAMPLE takes value p as input where, ideally, $p = n\xi/|E(S,T)|$. Unfortunately, $|E(S,T)|$ is not known; SET-SAMPLE has access to $E' \subseteq E(S,T)$ but $E_{\text{stream}}(S,T) = E(S,T) \setminus E'$ is in the rest of the stream still unseen by the algorithm. To alleviate this, our algo-

rithm *approximates* $|E(S,T)|$. By using SET-SAMPLE, this enables generating samples of the $E(S,T)$ sequence without going through the entire stream each time. This yields Algorithm 4, which uses only a single pass.

Algorithm 4 A single-pass randomized-stream directed densest subgraph algorithm

Input: $c > 0$, and $\epsilon \in (0,1)$
Output: A $(2+\varepsilon)$-approximate directed densest subgraph

Initialize each of S, T, S^*, T^* to V; $E' \leftarrow \emptyset$
while $E_{\text{stream}} \neq \emptyset$ **do**
  $E_A \leftarrow$ next $n\xi$ edges from E_{stream}
  if $|E_A(S,T)| < 2\xi$ or $E_{\text{stream}} = \emptyset$ **then**
    $E' \leftarrow E' \cup E_A(S,T) \cup E_{\text{stream}}(S,T)$
    If the densest subgraph in E' is denser than (S^*, T^*), update (S^*, T^*) to be that subgraph.
  else
    $s \leftarrow (1-\epsilon) \cdot \frac{|E_A(S,T)|}{|E_A|} \cdot (|E_{\text{stream}}| + n\xi) + |E'|$
    $E' \leftarrow E' \cup E_A(S,T)$
    $p \leftarrow \frac{n\xi}{(1-\epsilon)s}$
    $H \leftarrow$ SET-SAMPLE (E', S, T, p, s)
    if $p > 1$ **then** $H \leftarrow E' \cup E_{\text{stream}}(S,T)$
    $(S,T) \leftarrow$ VSETS-UPDATE(H, c, ϵ, S, T)
    if $\rho(S,T) > \rho(S^*, T^*)$ **then**
      $S^* \leftarrow S, T^* \leftarrow T$
    end if
    $E' \leftarrow (E' \cup E_H) \cap E_G(S,T)$
  end if
end while
return S^*, T^*

Notice that E_A in Algorithm 4 is a sample of $n\xi$ edges taken uniformly at random from *the rest* of the stream. Additionally, graph H uses SET-SAMPLE to create an approximately uniform sample of edges from graph $E(S,T)$. In both cases, the edges of the samples are not chosen independently, so the standard Chernoff bound cannot be applied. Therefore, to still establish some connection between the degrees of vertices of the sampled graphs and the original graph, we consider Lemma 4.4, a generalized version of the Chernoff bound which holds for negatively correlated variables. Boolean random variables $X_1, X_2, \ldots, X_n$ are negatively correlated if any subset S of $\{X_1, X_2, \ldots, X_n\}$ and any element $a \in S$ follows $\Pr(a = 1 | \forall_{b \in S - \{a\}} b = 1) \leq \Pr(a = 1)$.

Lemma 4.4 (Folklore)**.** *Let $X = \sum_{i=1}^{n} X_i$ where $X_1, X_2, \ldots X_n$ are negatively correlated boolean random variables. Then, for $\epsilon \in (0,1)$, we have that*

$$\Pr(|X - \mathbb{E}[X]| > \epsilon \mathbb{E}[X]) \leq 3\exp(-\epsilon^2 \mathbb{E}[X]/3).$$

With Lemma 4.4, we prove Lemma 4.5, which considers sampling using SET-SAMPLE, and Lemma 4.6, which considers sampling a fixed amount of edges uniformly at random. These lemmas will be used to analyze Algorithm 4.

Lemma 4.5. *Let $G = (V,E)$ be a graph, $\epsilon \in (0,1)$, and $p \in (0,1]$. Let $H \leftarrow$ SET-SAMPLE (E', S, T, p, s) be obtained from Algorithm 4. Then, for all $v \in G$, the following hold with probability $1 - \frac{1}{n^3}$: (i) if $d_G^-(v) \geq \xi/p$ then $pd_G^-(v) \leq (1+\epsilon)d_H^-(v)$, (ii) if $d_G^-(v) < \xi/p$ then $d_H^-(v) < 2\xi$ (same claims hold for out-degrees $d_G^+(v)$ and $d_H^+(v)$).*

Proof. Consider a sample H of G obtained by SET-SAMPLE. Number the edges in E from 1 to $m = |E|$. For each $i = 1 \ldots m$, define a Boolean random variable X_i which equals 1 iff $i \in H$ and 0 otherwise.

Let S be an arbitrary subset of $\{X_1, X_2, \ldots, X_m\}$ and X_i be an arbitrary element of S. Recall that SET-SAMPLE uses two sets to output H: one is the input set E', and the other one is the rest of the stream E_{stream}. To sample from E_{stream}, SET-SAMPLE first samples x on Algorithm 3, and then fetches the next x edges from $E_{\text{stream}} \cap E_G(S,T)$; hence, that edge-sample is correlated as the number of edges is fixed. If $i \in E'$, then X_i is independent of all the other random variables, and therefore, $\Pr(X_i = 1 | \forall_{b \in S - \{X_i\}} b = 1) = \Pr(X_i = 1)$. If $i \in E_{\text{stream}}$ instead, we observe that if the sample already contains an edge j, then it "reduces" the probability of i appearing in H. Overall, the random variables are negatively correlated.

Now, under the conditions of our lemma, we have that $p \leq 1$. So, when we use SET-SAMPLE, we see that

$$\begin{aligned} \mathbb{E}\left[d_H^-(v)\right] &= \mathbb{E}\left[\frac{p \cdot s}{|E_G(S,T)|} \cdot d_G^-(v)\right] \\ &= \frac{n\xi}{(1-\epsilon)|E_G(S,T)|} d_G^-(v) \end{aligned}$$

since SET-SAMPLE, in expectation, samples ps edges out of all the edges in $E_G(S,T)$. Notice that the result is the same as $\mathbb{E}\left[d_H^-(v)\right]$ in Lemma 4.1. Therefore, following the proof of Lemma 4.1 but using Lemma 4.4 instead of the Chernoff bound, we prove Lemma 4.5. □

Lemma 4.6. *Let $G = (V,E)$ be a graph, $\epsilon \in (0,1)$. Assuming $|E| \geq n\xi$, let H be a sample of $n\xi$ edges from G chosen uniformly at random with probability $p = \frac{n\xi}{|E|}$. Then, for all $v \in G$, the following hold with probability $1 - \frac{1}{n^3}$: (i) if $d_H^-(v) \geq 2\xi$ then $|d_H^-(v) - pd_G^-(v)| \leq \epsilon d_H^-(v)$, (ii) if $d_H^-(v) < 2\xi$ then $d_G^-(v) \leq \frac{2(1+\epsilon)\xi}{p}$ (same claims hold for out-degrees $d_G^+(v)$ and $d_H^+(v)$).*

The proof of Lemma 4.6 is given in Appendix C. The following is our analysis of Algorithm 4.

Lemma 4.7. *Algorithm 4 uses* $O\left(\frac{n\log^2 n}{\epsilon^3}\right)$ *memory with probability* $1-\frac{\log_{1+\varepsilon} n}{n^4}$.

The proof of this memory upper bound, Lemma 4.7, can be found in Appendix D, using applications of the Chernoff bound and Lemma 4.6.

Theorem 4.8. *Algorithm 4 produces a* $2(1+\epsilon)$*-approximation of the densest directed subgraph with probability* $1-\frac{\log_{1+\epsilon} n}{n^2}$.

Proof. The pruning of vertices in our algorithm follows that of Algorithm 2, using SET-SAMPLE to create sample graph H and then applying VSETS-UPDATE. Therefore, following the proof of Theorem 4.3 but using Lemma 4.5 instead of Lemma 4.1, we attain the same approximation of the directed densest subgraph. □

4.3. Removing the assumption on c_{OPT} being known

Executing Algorithm 4 requires knowing c_{OPT}, the ratio between the two optimal sets. We remove this assumption by using standard techniques. Namely, it is unclear how to learn c without finding the densest subgraph. To alleviate this, our algorithm guesses the value of c and runs the algorithm described above for each guess. Since there are $O(n^2)$ candidates for c, i.e., values a/b for $a,b\in\{1,\ldots,n\}$, our algorithm guesses the value of c in the multiplicative increments of $\delta>1$. More precisely, we run our algorithm for values of c ranging from $1/n$ to n and of the form δ^i/n. As Lemma 4.9 shows, this approach incurs an additional factor of $\sqrt{\delta}$ in our approximation.

Lemma 4.9. *The best density of Algorithm 4 ran on values of* $c=1/n$ *through* n*, multiplying by a constant factor of* δ*, is a* $2(1+\epsilon)\sqrt{\delta}$*-approximation of the densest directed subgraph with probability* $1-\frac{\log_{1+\epsilon} n}{n^2}$.

Proof. Consider c where $c_{\text{OPT}}/\delta\le c\le \delta c_{\text{OPT}}$. Drawing from the proof of Theorem 4.3, consider $\tilde{S},\tilde{T}$ with $|\tilde{S}|/|\tilde{T}|=c_{\text{OPT}}$ that maximizes $\rho(\tilde{S},\tilde{T})=\rho^*(G)$. Then,

$$\begin{aligned}\rho^*(G) &= \frac{|E(\tilde{S},\tilde{T})|}{\sqrt{|\tilde{S}||\tilde{T}|}} \le \frac{|\tilde{S}|\cdot d^*_{out}+|\tilde{T}|\cdot d^*_{in}}{\sqrt{|\tilde{S}||\tilde{T}|}}\\ &= \sqrt{c_{\text{OPT}}}\cdot d^*_{out}+\frac{1}{\sqrt{c_{\text{OPT}}}}\cdot d^*_{in}\\ &\le \sqrt{\delta c}\cdot d^*_{out}+\sqrt{\delta/c}\cdot d^*_{in}\\ &\le 2(1+\epsilon)^3\sqrt{\delta}\cdot\rho(S^*,T^*)\end{aligned}$$

Therefore, this gives a $2(1+\epsilon)\sqrt{\delta}$-approximation of the densest directed subgraph. □

5. MPC Algorithms

In this section, we extend our approach developed in Section 4 to two memory regimes of the MPC model. We present the ideas gradually. First, we show that slightly adjusting Algorithm 4 yields an $O(1)$ MPC round algorithm in the super-linear memory regime. To achieve this round complexity, we prove a certain edge-size progress that was not required for the semi-streaming analysis. Second, in Section 5.2 we show that, by further building on the MPC super-linear memory algorithm, it is possible to find an approximate directed densest subgraph in only $O(\sqrt{\log n})$ rounds in the near-linear memory regime.

5.1. Super-linear Memory Regime

Recall that in Algorithm 4, E_{stream} is the *remaining* edges in the stream. In developing a super-linear regime MPC algorithm, our main idea is to implement E_{stream} efficiently in MPC. More concretely, assume that a machine $\mathcal{M}$ aims to simulate Algorithm 4. Then, $\mathcal{M}$ first samples $n^{1+\delta}$ edges from the graph – let that set be X_1 – and presents X_1 to Algorithm 4 as a prefix of E_{stream}. Once Algorithm 4 makes a pass over X_1, $\mathcal{M}$ needs to fetch another set X_2 of $n^{1+\delta}$ edges from $E\setminus X_1$ and continue Algorithm 4 with X_2. This process continues until there are no edges to feed to Algorithm 4; the samples $X_1,X_2,\ldots$ represent E_{stream}. Clearly, this simulation corresponds to Algorithm 4 executed on a stream. Unfortunately, fetching sets X_i can happen $m/n^{1+\delta}$ times, which for $m\gg n^{1+\delta}$ is highly inefficient.

Our main idea is that after processing X_1, it suffices if $\mathcal{M}$ takes a random sample from $E_G(S,T)\cap(E_{\text{stream}}\setminus X_1)$ as opposed from $E_{\text{stream}}\setminus X_1$. The main question here is: "*How does this translate to round complexity?*" As the primary measure of progress, we show the following. Let (S',T') be the vertex set pair in Algorithm 4 just before our simulation of Algorithm 4 exhausted X_1. Then, we show that, up to low-order terms, $|E_G(S',T')|<m/n^{\delta}$. More generally, we show that whenever our simulation needs to fetch fresh X_i from the remaining unused edges, the number of *relevant* remaining edges reduces by a factor of n^{δ}. Hence, after $O(1/\delta)$ many such steps, the number of remaining relevant edges fits in the memory of a single machine, at which point the rest of the problem can be solved locally. We dive into the details of this argument in our proof, given in Appendix E, of the following claim.

Theorem 5.1. *There exists an algorithm that runs in* $O(1/\delta)$ *rounds with probability* $1-\frac{\log_{1+\epsilon} n}{n^4}$.

5.2. Near-linear Memory Regime

In this section, we improve the efficiency of our MPC algorithm for the super-linear memory regime, enabling us to

obtain a quadratically faster algorithm for the near-linear memory regime than currently known (Bahmani et al., 2012; 2014). Our approach stems from two ideas. The first idea is an observation: instead of sampling $n\xi$ edges, our algorithm requires sampling only $(|S| + |T|) \cdot \xi$ edges. Hence, once $|S| + |T|$ become sufficiently small, then the memory per machine becomes significantly larger than our sample sizes, and we can use ideas similar to those from the proof of Theorem 5.1 to improve the $O(\log n)$ round complexity. Unfortunately, this alone is not sufficient. To see that, consider an example in which $|S|/|T| = \Theta(n)$, e.g., the densest subgraph has a star-like structure. In that case, it is potentially needed to peel T for $O(\log n)$ times. Since $|S| + |T| = \Theta(n)$ during those peeling steps, our observation does not yield any improvement.

The second idea is that as long as our algorithm peels only T or only S, no fresh sampling is required. To elaborate, assume that the peeling process produces sets $(S, T_1), (S, T_2), \ldots, (S, T_j)$. Since S remains the same, the estimated degree between S and a vertex $v \in T_j$ is the same in all the sets $T_1, T_2, \ldots, T_j$. Hence, once $|E_G(S, v)|$ is estimated, no new sample is needed until S changes. In other words, the star-like example we pointed to above can be handled with a single sample. *Nevertheless, what does happen when S changes?* In that case, the algorithm might need a new round to obtain a fresh sample. Fortunately, the size of S reduces by at least a factor of $(1 + \varepsilon)$, increasing the gap between the memory per machine and $|S| + |T|$, which our algorithm utilizes in a way similar to described in Section 5.1. We formalize these ideas in the proof of Theorem 5.2 given in Appendix F.

Theorem 5.2. *There exists an algorithm that runs in $O(\sqrt{\log_{1+\epsilon} n})$ rounds with probability $1 - \frac{\log_{1+\epsilon} n}{n^4}$.*

Note that our approach uses samples of size $O((|S|+|T|) \cdot \mathrm{poly}(\log(n)/\varepsilon))$ in order to estimate the degrees between S and T whp and enable multiple iterations of peeling in a single round on a single machine. Therefore, when $|S| + |T| \gg n^\delta$, it is difficult to generalize our algorithm to the sub-linear memory regime since there is not enough memory per machine to store the entire sample needed to accurately estimate degrees of the vertices in S and T. Additionally, if we rely on recalculating the degrees of vertices directly using $O(1)$ rounds, our optimization can no longer be done until the graph is able to fit on a single machine. This could potentially require $\Theta(\log n)$ peeling steps, each of which requires $O(1)$ rounds.

6. Experiments

We now demonstrate the practical performance of Algorithm 4, comparing it to Algorithm 3 from (Bahmani et al., 2012) in the streaming setting. Additionally, we implement our MPC algorithm in the near-linear memory regime, described in Theorem 5.2, analyzing the amount of phases it takes in practice.

6.1. Data

We use 4 data sets from the Stanford Large Network Dataset Collection (Leskovec & Krevl, 2014): Slashdot, Berkeley-Stanford Web Graph, LiveJournal, and Twitter. In addition, we generate two synthetic graphs, PrefAttach 1 and 2, using the preferential attachment scheme (Simon, 1955). This scheme is popular in the scientific analysis of graphs as it can generate the power-law distribution (Barabási & Albert, 1999; Jacob & Mörters, 2015; Wan et al., 2017) and, as such, it models the vertex-degree distribution of many important graphs, including World Wide Web (Kunegis et al., 2013) and Wikipedia (Capocci et al., 2006; Gandica et al., 2015).

Table 1. Datasets we use in the evaluation.

Graph	Nodes	Edges
Slashdot	82,168	948,464
Berk-Stan Web	685,230	7,600,595
LiveJournal	4,847,571	68,993,773
PrefAttach 1	100,000	100,051,302
PrefAttach 2	1,000,000	999,999,375
Twitter	41,652,230	1,468,364,884

6.2. Experimental Setup

As a baseline, we use (Bahmani et al., 2012). Both algorithms, (Bahmani et al., 2012) and ours, are implemented in C++ on a M1 machine running macOS Monterey 12.5.1 with 4 cores, 8 GB of RAM, 256 KB of L2 Cache, and 2.5 MB of L3 Cache (per core). We run the algorithms for values of c ranging from $1/n$ to n with constant $\delta = 2$ and $\epsilon = 0.2$. Then, we study the algorithms in terms of how close their approximation of the densest subgraph is and their runtime. Plots of density and runtime are created with respect to the constant c.

An important note about Algorithm 4 is that it cannot calculate $\rho(S, T)$ for each iteration without seeing the whole graph. However, when implementing the algorithm, we approximate that density through our sampled graph by multiplying the density in the sampled graph, $\rho_H(S, T)$, by $1/p$.

Algorithm 4 creates samples of size $n\xi = \frac{f \cdot n \log n}{\epsilon^2}$. In the theoretical analysis, we set $f = 60$. In our evaluations, we also investigate the effect of f on the performance of our approach, expecting that f significantly smaller than 60 would also result in accurate and fast execution of Algorithm 4. To that end, in our implementation of Algorithm 4, we consider f ranging from $1/1200$ to $1/30$.

Although our guarantees for Algorithm 4 require randomized streams, to perform a faithful comparison to (Bahmani

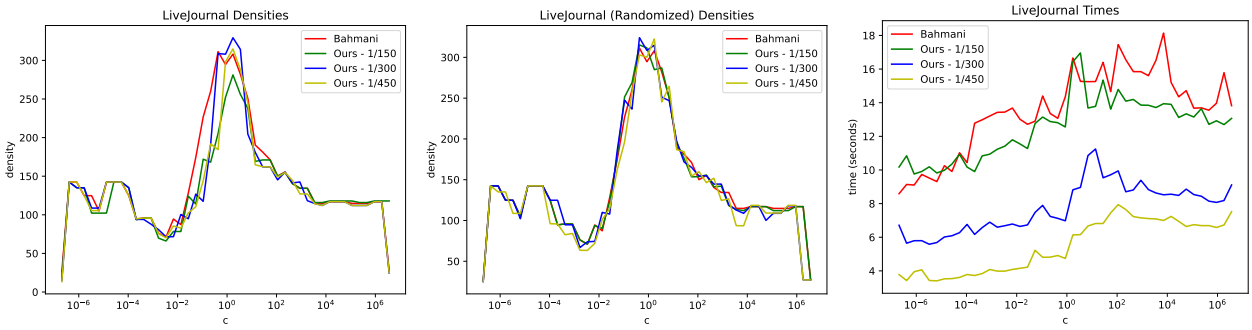

Figure 2. Density and running-time as a function of c for LiveJournal with $f = 1/150, 1/300, 1/450$.

Figure 3. Density and running-time as a function of c for Twitter with $f = 1/100, 1/200$.

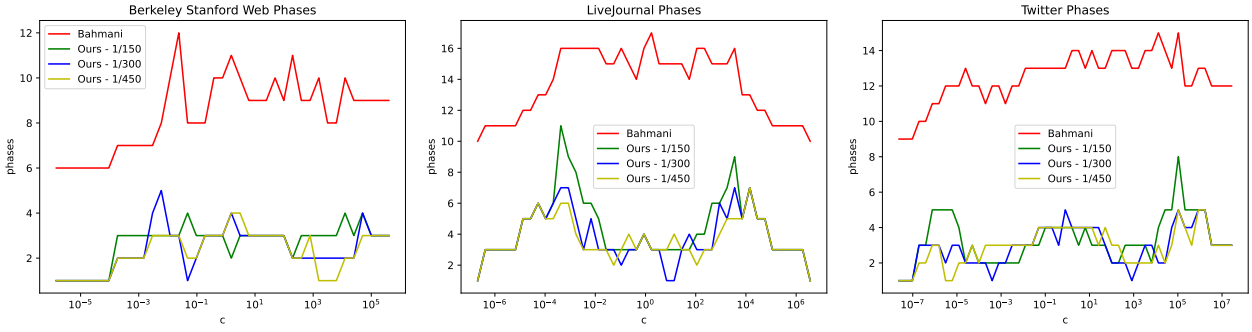

Figure 4. Number of phases as a function of c for various data sets with $f = 1/150, 1/300, 1/450$.

et al., 2012), we also execute Algorithm 4 **on arbitrarily ordered streams**. In many cases, these arbitrarily ordered streams are edges sorted by their endpoints, i.e., these streams are far from being randomized.

6.3. Streaming Results

Our algorithm finds subgraphs with densities very close to those output by our baseline (Bahmani et al., 2012). For each graph, the densities with the best corresponding f are within 3% of each other. Interestingly, this is the case even for relatively small values of f, such as $f = 1/300$, suggesting that relatively sparse samples can be leveraged in extracting dense subgraphs. Moreover, our algorithm attains higher accuracy for large graphs than (Bahmani et al., 2012). For example, our algorithm's density is better by 5.5% for LiveJournal (Figure 2) and by 1.5% for Twitter (Figure 3).

In addition to the maximum density of our algorithm being similar to (Bahmani et al., 2012), the shapes of the density plots with respect to c are incredibly similar. This further solidifies that, in terms of accuracy, Algorithm 4 matches (Bahmani et al., 2012) even on non-randomized streams.

In the cases of denser graphs, our algorithm is around 2 times faster than the baseline. For LiveJournal (Figure 2) and Twitter (Figure 3), they both reach around 2.5 times

faster. There is a balance between using smaller constants for our sample size, leading to the algorithm becoming faster, and the accuracy of the algorithm's density approximation.

Note that we can see improvements in the density attained when the stream becomes randomized. For example, when the stream is randomized, all the maximum densities attained by our algorithm for LiveJournal (Figure 2) become better than the maximum density attained by (Bahmani et al., 2012) and the plots match significantly more closely. However, as mentioned before, even with non-randomized streams our algorithm still performs well. In Appendix G, we provide results of additional evaluations, including results for $\epsilon = 0.1$ which are similar to those described here.

6.4. MPC Results

We also test the number of phases our MPC algorithm, which simulates Algorithm 4, takes in the near-linear memory regime. From Theorem 5.2, we already have a bound of $O(\sqrt{\log_{1+\epsilon} n})$ phases. Additionally, we make a comparison to (Bahmani et al., 2012) and their $O(\log_{1+\epsilon} n)$ parallel algorithm, defining a phase of their algorithm to be calculating the degrees of all vertices and peeling vertices based on these degrees. We let each machine have enough memory to hold $\frac{2}{1-\epsilon} \cdot n\xi = \frac{2}{1-\epsilon} \cdot \frac{f \cdot n \log n}{\epsilon^2}$ edges in order for there to be enough edges per machine for Algorithm 4 to run for at least one iteration during each phase.

As we can see from Figure 4, the amount of phases our MPC algorithm takes is at least half of that of (Bahmani et al., 2012). Also, there doesn't seem to be a correlation between the number of phases and the number of vertices of the graph n, showing that in practice these algorithms perform much better than their theoretical upper bounds. Appendix G contains plots of phases for other data sets.

7. Conclusion and Future Work

We studied the approximate directed densest subgraph problem. We developed sampling strategies that enabled us to design efficient algorithms for finding dense subgraphs in semi-streaming and MPC. Moreover, our approach appears to be highly practical – we showed empirically that these sampling strategies yield faster algorithms even on non-randomized streams.

Our work was largely inspired by a discrepancy between state-of-the-art efficiency of finding undirected and directed dense subgraphs. Even though we made significant progress in bridging this gap, many interesting questions remain. We mention three that we find the most exciting. (1) Is it possible to find a $\Theta(1)$-approximate directed densest subgraph in a non-randomized semi-streaming setting in $o(\log n)$ passes? As an intermediate question, studying the trade-off between the number of passes and the memory requirement would be interesting, e.g., interpolating between 1 pass and $n^{3/2}$ memory, and $O(\log n)$ passes and $O(n \operatorname{poly} \log n)$ memory. (2) For the same problem, does a $o(\log n)$ MPC round algorithm exist in the sublinear memory regime? (3) Our MPC algorithm for the near-linear memory regime outputs a $(2+\varepsilon)$-approximation. Is it possible to achieve a $(1+\varepsilon)$-approximation with the same round complexity?

Acknowledgements

We are grateful to Rishabh Bhaskaran for many insightful discussions on the empirical portion of our work. We are also grateful to anonymous reviewers for their valuable feedback. S. Mitrović was supported by the Google Research Scholar and NSF Faculty Early Career Development Program #2340048.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

References

Bahmani, B., Kumar, R., and Vassilvitskii, S. Densest subgraph in streaming and mapreduce. *Proceedings of the VLDB Endowment*, 5(5), 2012.

Bahmani, B., Goel, A., and Munagala, K. Efficient primal-dual graph algorithms for mapreduce. In *International Workshop on Algorithms and Models for the Web-Graph*, pp. 59–78. Springer, 2014.

Barabási, A.-L. and Albert, R. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

Beame, P., Koutris, P., and Suciu, D. Communication steps for parallel query processing. *Journal of the ACM (JACM)*, 64(6):1–58, 2017.

Bhattacharjee, P. and Mitra, P. A survey of density based clustering algorithms. *Frontiers of Computer Science*, 15:1–27, 2021.

Bhattacharya, S., Henzinger, M., Nanongkai, D., and Tsourakakis, C. 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*, pp. 173–182, 2015.

Buehrer, G. and Chellapilla, K. A scalable pattern mining approach to web graph compression with communities.

In *Proceedings of the 2008 international conference on web search and data mining*, pp. 95–106, 2008.

Capocci, A., Servedio, V. D., Colaiori, F., Buriol, L. S., Donato, D., Leonardi, S., and Caldarelli, G. Preferential attachment in the growth of social networks: The internet encyclopedia wikipedia. *Physical review E*, 74(3): 036116, 2006.

Charikar, M. Greedy approximation algorithms for finding dense components in a graph. In *Approximation Algorithms for Combinatorial Optimization: Third International Workshop, APPROX 2000 Saarbrücken, Germany, September 5–8, 2000 Proceedings*, pp. 84–95. Springer, 2003.

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

Chen, T. and Tsourakakis, C. Antibenford subgraphs: Unsupervised anomaly detection in financial networks. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 2762–2770, 2022.

Chen, T., Bonchi, F., Garcia-Soriano, D., Miyauchi, A., and Tsourakakis, C. E. Dense and well-connected subgraph detection in dual networks. In *Proceedings of the 2022 SIAM International Conference on Data Mining (SDM)*, pp. 361–369. SIAM, 2022.

Epasto, A., Lattanzi, S., and Sozio, M. Efficient densest subgraph computation in evolving graphs. In *Proceedings of the 24th international conference on world wide web*, pp. 300–310, 2015.

Esfandiari, H., Hajiaghayi, M., and Woodruff, D. P. Applications of uniform sampling: Densest subgraph and beyond. *arXiv preprint arXiv:1506.04505*, 2015.

Fang, Y., Luo, W., and Ma, C. Densest subgraph discovery on large graphs: Applications, challenges, and techniques. *Proceedings of the VLDB Endowment*, 15(12): 3766–3769, 2022.

Gandica, Y., Carvalho, J., and Dos Aidos, F. S. Wikipedia editing dynamics. *Physical Review E*, 91(1):012824, 2015.

Ghaffari, M., Lattanzi, S., and Mitrović, S. Improved parallel algorithms for density-based network clustering. In *International Conference on Machine Learning*, pp. 2201–2210. PMLR, 2019.

Goldberg, A. V. Finding a maximum density subgraph. 1984.

Goodrich, M. T., Sitchinava, N., and Zhang, Q. Sorting, searching, and simulation in the mapreduce framework. In *International Symposium on Algorithms and Computation*, pp. 374–383. Springer, 2011.

Harenberg, S., Bello, G., Gjeltema, L., Ranshous, S., Harlalka, J., Seay, R., Padmanabhan, K., and Samatova, N. Community detection in large-scale networks: a survey and empirical evaluation. *Wiley Interdisciplinary Reviews: Computational Statistics*, 6(6):426–439, 2014.

Jacob, E. and Mörters, P. Spatial preferential attachment networks: Power laws and clustering coefficients. 2015.

Karloff, H., Suri, S., and Vassilvitskii, S. A model of computation for mapreduce. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pp. 938–948. SIAM, 2010.

Kriegel, H.-P. and Pfeifle, M. Density-based clustering of uncertain data. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pp. 672–677, 2005.

Kriegel, H.-P., Kröger, P., Sander, J., and Zimek, A. Density-based clustering. *Wiley interdisciplinary reviews: data mining and knowledge discovery*, 1(3):231–240, 2011.

Kunegis, J., Blattner, M., and Moser, C. Preferential attachment in online networks: Measurement and explanations. In *Proceedings of the 5th annual ACM web science conference*, pp. 205–214, 2013.

Leon-Suematsu, Y. I., Inui, K., Kurohashi, S., and Kidawara, Y. Web spam detection by exploring densely connected subgraphs. In *2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*, volume 1, pp. 124–129. IEEE, 2011.

Leskovec, J. and Krevl, A. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

McGregor, A., Tench, D., Vorotnikova, S., and Vu, H. T. 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*, pp. 472–482. Springer, 2015.

Mitzenmacher, M., Pachocki, J., Peng, R., Tsourakakis, C., and Xu, S. C. 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*, pp. 815–824, 2015.

Ren, Y., Zhu, H., Zhang, J., Dai, P., and Bo, L. Ensemfdet: An ensemble approach to fraud detection based on bipartite graph. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pp. 2039–2044. IEEE, 2021.

Sawlani, S. and Wang, J. Near-optimal fully dynamic densest subgraph. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pp. 181–193, 2020.

Simon, H. A. On a class of skew distribution functions. *Biometrika*, 42(3/4):425–440, 1955.

Su, H.-H. and Vu, H. T. Distributed dense subgraph detection and low outdegree orientation. *arXiv preprint arXiv:1907.12443*, 2019.

Wan, P., Wang, T., Davis, R. A., and Resnick, S. I. Fitting the linear preferential attachment model. 2017.

Wu, J. M.-T., Lin, C. W., Fournier-Viger, P., Djenouri, Y., Chen, C.-H., and Li, Z. The density-based clustering method for privacy-preserving data mining. 2019.

Zhang, S., Zhou, D., Yildirim, M. Y., Alcorn, S., He, J., Davulcu, H., and Tong, H. Hidden: hierarchical dense subgraph detection with application to financial fraud detection. In *Proceedings of the 2017 SIAM International Conference on Data Mining*, pp. 570–578. SIAM, 2017.

Zhang, X., Li, Z., Zhu, S., and Liang, W. Detecting spam and promoting campaigns in twitter. *ACM Transactions on the Web (TWEB)*, 10(1):1–28, 2016.

A. Proof of Lemma 4.1

Proof. Consider a vertex $v \in V$. Let X be the random variable equal to $d_H^-(v)$. Observe that $\mathbb{E}[X] = pd_G^-(v)$. Then, if $d_G^-(v) \geq \xi/p$, we have

$$\begin{aligned} & \Pr(\mathbb{E}[X] \geq (1+\epsilon)X) \\ \leq\ & \Pr\left(\mathbb{E}[X] - X \geq \frac{\epsilon}{2}\mathbb{E}[X]\right) \\ \leq\ & 3\exp\left(-\epsilon^2\mathbb{E}[X]/12\right) \qquad (2) \\ =\ & 3\exp\left(-\epsilon^2 pd_G^-(v)/12\right) \\ \leq\ & 3\exp(-5\log(n)) \\ =\ & \frac{3}{n^5} \leq \frac{1}{n^4} \end{aligned}$$

assuming $n \geq 3$, where we used the Chernoff bound, Theorem 2.1 (A), to derive (2).

Now, if $d_G^-(v) < \xi/p$, we have

$$\begin{aligned} & \Pr(X \geq 2\xi) \\ \leq\ & \Pr(X - \mathbb{E}[X] \geq \xi) \\ \leq\ & \frac{3}{2}\exp(-\xi^2/(3\mathbb{E}[X])) \qquad (3) \\ \leq\ & \frac{3}{2}\exp(-\xi/3) \\ \leq\ & \frac{1}{n^4} \end{aligned}$$

where (3) follows by the Chernoff bound.

Therefore, using the union bound, the claim holds with probability $1 - \frac{1}{n^3}$. The same proof applies to out-degrees. □

B. Proof of Theorem 4.3

Proof. Consider a pass with $|S|/|T| \geq c$. Hence, vertices are removed from S in that pass. Let $p = n\xi/((1-\epsilon)|E_G(S,T)|)$; the same probability is used on Algorithm 2 of Algorithm 2. Then, for all vertices $i \in A(S)$, where $A(S)$ is defined in VSETS-UPDATE, we have

$$\begin{aligned} \sqrt{c}|E_G(i,T)| &\leq \sqrt{c}\cdot\frac{(1+\epsilon)^2|E_H(S,T)|}{p|S|} \qquad (4) \\ &\leq \sqrt{c}\cdot(1+\epsilon)^3\frac{|E_G(S,T)|}{|S|} \qquad (5) \\ &\leq \sqrt{c}\cdot(1+\epsilon)^3\frac{|E_G(S,T)|}{\sqrt{c|S||T|}} \\ &= (1+\epsilon)^3\frac{|E_G(S,T)|}{\sqrt{|S||T|}} \\ &= (1+\epsilon)^3\rho(S,T). \qquad (6) \end{aligned}$$

For (4), we either have that $|E_G(i,T)| \geq \xi/p$ or $|E_G(i,T)| < \xi/p$. If $|E_G(i,T)| \geq \xi/p$, we use Lemma 4.1 to get

$$\begin{aligned} \sqrt{c}|E_G(i,T)| &\leq \sqrt{c}\cdot\frac{(1+\epsilon)|E_H(i,T)|}{p} \\ &\leq \sqrt{c}\cdot\frac{(1+\epsilon)^2|E_H(S,T)|}{p|S|}. \end{aligned}$$

On the other hand, if $|E_G(i,T)| < \xi/p$, (4) follows directly. Additionally, since $\mathbb{E}[|E_H(S,T)|] = p|E_G(S,T)|$, a direct application of Chernoff bound, Theorem 2.1 (B), yields (5).

Similarly, for a pass with $|S|/|T| < c$, we have that for all vertices $j \in B(T)$,

$$\frac{1}{\sqrt{c}}|E_G(S,j)| \leq (1+\epsilon)^3\rho(S,T). \qquad (7)$$

We define $d_{\text{out}}^{\text{rem}}(i) \stackrel{\text{def}}{=} |E_G(i,T)|$ and $d_{\text{in}}^{\text{rem}}(j) \stackrel{\text{def}}{=} |E_G(S,j)|$ where T is the respective set during the pass i is removed from S and, similarly, S is the respective set during the pass j is removed from T. Let $d_{out}^* = \max_{i\in V}\{d_{\text{out}}^{\text{rem}}(i)\}$ and $d_{in}^* = \max_{j\in V}\{d_{\text{in}}^{\text{rem}}(j)\}$. Then, for $\tilde{S},\tilde{T} \subseteq V$ with $|\tilde{S}|/|\tilde{T}| = c$ that maximizes $\rho(\tilde{S},\tilde{T}) = \rho^*(G)$, we have that

$$\begin{aligned} \rho^*(G) &= \frac{|E(\tilde{S},\tilde{T})|}{\sqrt{|\tilde{S}||\tilde{T}|}} \leq \frac{|\tilde{S}|\cdot d_{out}^* + |\tilde{T}|\cdot d_{in}^*}{\sqrt{|\tilde{S}||\tilde{T}|}} \\ &= \sqrt{c}\cdot d_{out}^* + \frac{1}{\sqrt{c}}\cdot d_{in}^* \leq 2(1+\epsilon)^3\rho(S^*,T^*) \end{aligned}$$

where the last inequality comes from using (6) and (7). (The last chain of inequalities is also shown in (Charikar, 2003).)

Therefore, the algorithm produces a $2(1+\epsilon)^3$-approximation. Note that we use Lemma 4.1 on each vertex for every pass. Therefore, by the union bound, we have a success probability of $1 - n\cdot\log_{1+\epsilon}n\cdot\frac{1}{n^3} = 1 - \frac{\log_{1+\epsilon}n}{n^2}$. □

C. Proof of Lemma 4.6

Proof. Note that we can apply Lemma 4.4 since we perform uniform random sampling with a fixed number of edges. Therefore, when referencing Lemma 4.1 in this proof, it will be using Lemma 4.4 instead of the Chernoff bound in the context of uniform random sampling with a fixed number of edges.

Consider a vertex $v \in V$. Note that if $d_H^-(v) \geq 2\xi$, then $d_G^-(v) \geq \xi/p$ with probability $1 - \frac{1}{n^4}$ using the Chernoff bound. Therefore, with Lemma 4.1, we have that $|d_H^-(v) - pd_G^-(v)| \leq \epsilon d_H^-(v)$.

If $d_H^-(v) < 2\xi$, we split our analysis into two cases. If

$d^-_G(v) \geq \xi/p$, then

$$d^-_G(v) \leq \frac{(1+\epsilon)d^-_H(v)}{p} < \frac{2(1+\epsilon)\xi}{p}$$

using Lemma 4.1. Otherwise, if $d^-_G(v) < \xi/p$, then $d^-_G(v) < \frac{2(1+\epsilon)\xi}{p}$ holds as well. Therefore, the statements in Lemma 4.6 hold with probability $1 - \frac{1}{n^3}$ using the union bound. The same proof applies to out-degrees. □

D. Proof of Lemma 4.7

Proof. First, note that our algorithm takes $O(\log_{1+\epsilon} n)$ iterations.

For the graph H, we either have $p \leq 1$ or $p > 1$. If $p \leq 1$, we create a sample of edges with probability $\frac{n\xi}{(1-\epsilon)s}$ using SET-SAMPLE. Note that s is an approximation of $|E_G(S,T)|$ where, with probability $1 - \frac{1}{n^4}$, it holds

$$s \leq |E_G(S,T)| \leq \frac{1+\epsilon}{1-\epsilon} \cdot s \tag{8}$$

using the Chernoff bound, Theorem 2.1 (A). So, for each of the $O(\log_{1+\epsilon} n)$ iterations, graph H will have $O\left(\frac{(1+\epsilon)^2 n \log n}{\epsilon^2(1-\epsilon)^2}\right) = O\left(\frac{n \log n}{\epsilon^2}\right)$ edges with probability $1 - \frac{1}{n^4}$ and $\epsilon < 0.9$. On the other hand, if $p > 1$, we must have that

$$(1-\epsilon)s < n\xi \implies |E_G(S,T)| < \frac{(1+\epsilon)n\xi}{(1-\epsilon)^2}$$

with probability $1 - \frac{1}{n^4}$ using (8). Then, in this case, graph H will also have $O\left(\frac{n \log n}{\epsilon^2}\right)$ edges.

Note that E' only increases by at most $n\xi$ edges each iteration in addition to edges from graph H. Therefore, using the union bound, the number of edges in E' at any point in time is at most $O\left(\frac{n \log n \cdot \log_{1+\epsilon} n}{\epsilon^2}\right) = O\left(\frac{n \log^2 n}{\epsilon^3}\right)$ with probability $1 - \frac{\log_{1+\epsilon} n}{n^4}$.

Now, E_A is either a uniform random sample of $n\xi$ edges from the remaining edges in the stream, since the stream is randomized, or is the rest of the stream. If it is the rest of the stream, then our algorithm reduces to finding the densest subgraph in $E' \cup E_A(S,T)$. This uses at most $O\left(\frac{n \log^2 n}{\epsilon^3}\right)$ memory. However, if E_A is not the rest of the stream, then either $|E_A(S,T)| < 2\xi$ or $|E_A(S,T)| \geq 2\xi$.

If $|E_A(S,T)| < 2\xi$, then we add the edges from $E_A(S,T)$ and $E_{\text{stream}}(S,T)$ to E' and find the densest subgraph in E'. Using Lemma 4.6, at most $\frac{2(1+\epsilon)\xi}{p} \leq 2(1+\epsilon)n$ edges, where $p = \frac{n\xi}{|E_{\text{stream}}(S,T) \cup E_A(S,T)|}$, are added to E' from E_A and the stream. This uses $O\left(\frac{n \log^2 n}{\epsilon^3}\right)$ memory.

If $|E_A(S,T)| \geq 2\xi$, then we have that the number of edges from $E_G(S,T)$ that were in the stream is at least $\frac{s(1-\epsilon)|E_A(S,T)|}{|E_A|}$ using Lemma 4.6, allowing us to calculate s' for creating graph H.

Therefore, the total amount of memory used by the algorithm is $O\left(\frac{n \log^2 n}{\epsilon^3}\right)$. □

E. Proof of Theorem 5.1

Proof. We first describe the algorithm and then an analysis of its round complexity.

Algorithm description. Let $\mathcal{A}_{\text{MPC}}$ be our MPC algorithm.

- $\mathcal{A}_{\text{MPC}}$ maintains the set of *relevant* edges, denoted by E_{rel}. Initially, $E_{\text{rel}} = E_G$.
- This algorithm simulates an instance of Algorithm 4, and only that instance. Let the instance be simulated on a machine $\mathcal{M}$; the simulation proceeds in phases as follows:
  - Let (S_{i+1}, T_{i+1}) be the vertex sets of Algorithm 4 at the end of phase i of $\mathcal{A}_{\text{MPC}}$.
  - At the beginning of phase $i+1$, $\mathcal{A}_{\text{MPC}}$ updates $E_{\text{rel}} \leftarrow E_{\text{rel}} \cap E_G(S_{i+1}, T_{i+1})$.
  - Then, $\mathcal{A}_{\text{MPC}}$ places on $\mathcal{M}$ a set X_{i+1} of $n^{1+\delta}$ edges chosen uniformly at random from E_{rel} and updates $E_{\text{rel}} \leftarrow E_{\text{rel}} \setminus X_{i+1}$.
  - The rest of phase $i+1$ consists of continuing the Algorithm 4 instance with the set X_{i+1}.
- $\mathcal{A}_{\text{MPC}}$ stops when E_{rel} fits entirely on $\mathcal{M}$.

All the steps of $\mathcal{A}_{\text{MPC}}$, except updating E_{rel} and sampling X_{i+1}, are done on a single machine. Removing X_i from E_{rel} can be done by first sorting E_{rel} and X_i together and then removing that edge that has a duplicate in the sorted list. Sorting can be done in $O(1)$ rounds (Goodrich et al., 2011). Performing $E_{\text{rel}} \cap E_G(S_{i+1}, T_{i+1})$ can be done again by sorting E_{rel}, S_{i+1} and T_{i+1} altogether. Those edges incident to S_{i+1} or T_{i+1} are the only ones kept in E_{rel}. To sample X_{i+1}, first, each edge in E_{rel} samples a random integer in the range $[1, n^8]$. Whp, all the integers sampled by the edges are distinct. Second, all the edges are sorted with respect to the sampled integers; effectively, this creates a random permutation of E_{rel}. Third, the first $n^{1+\delta}$ edges in that sorted list are X_{i+1}.

Algorithm analysis. Consider a phase i of $\mathcal{A}_{\text{MPC}}$ and let (S_i, T_i) be our vertex sets before the pruning starts. Now

consider some pair of vertex sets (S', T') while running Algorithm 4 during phase i. We define E_{rem} as the union of E_{rel} and the remaining edges in X_i that the algorithm has not inspected so far. There are $|E_G(S', T')| - |E'|$ edges in $E_G(S', T')$ remaining in E_{rem}.

Algorithm 4 needs at most $\frac{2n\xi}{1-\epsilon}$ edges in $E_G(S', T')$ from the stream in order to start peeling. Let k be the number of edges taken from the stream to perform one iteration of peeling. Therefore, using the Chernoff bound, Theorem 2.1 (A), we see that

$$k \leq \frac{2n\xi|E_{\text{rem}}|}{(1-\epsilon)^2(|E_G(S', T')| - |E'|)}.$$

with probability $1-\frac{1}{n^4}$, assuming that $|E_G(S', T')|-|E'|$ is $\Omega\left(n \log^2 n/\epsilon^2\right)$. If $|E_G(S', T')|-|E'|$ is $O\left(n \log^2 n/\epsilon^2\right)$, then all the edges in E_{rel} for the next phase fit on $\mathcal{M}$.

Now, if $k \geq \frac{n^{1+\delta}}{2\log_{1+\epsilon} n}$, then we have that

$$\begin{aligned} \frac{n^{1+\delta}}{2\log_{1+\epsilon} n} &\leq \frac{2n\xi|E_{\text{rem}}|}{(1-\epsilon)^2(|E_G(S', T')| - |E'|)} \\ \implies |E_G(S', T')| &\leq \frac{4\xi \log_{1+\epsilon} n}{(1-\epsilon)^2 n^{\delta}} \cdot |E_{\text{rem}}| + |E'|. \end{aligned}$$

Recall from Lemma 4.7 that E' contains at most $O\left(\frac{n' \log^2 n}{\epsilon^3}\right)$ edges. So, if $|E_{\text{rem}}| \leq |E'|$, then in the next phase E_{rel} fits entirely on $\mathcal{M}$. On the other hand, if $|E_{\text{rem}}| > |E'|$, then whp $E_G(S', T')$ has $O\left(\frac{\log^2 n}{\epsilon^3 n^{\delta}} \cdot |E_{\text{rem}}|\right)$ edges for $\epsilon < 0.9$. Therefore, since E_{rem} is a subset of $E_G(S_i, T_i)$, this phase causes the number of edges between vertex sets S and T to reduce by a factor of $\Omega(n^{\delta})$. Then, $\mathcal{A}_{\text{MPC}}$ takes $O(1/\delta)$ rounds if every phase has at least one pair of vertex sets with $k \geq \frac{n^{1+\delta}}{2\log_{1+\epsilon} n}$.

Consider now that $k < \frac{n^{1+\delta}}{2\log_{1+\epsilon} n}$ for all pairs of vertex sets in a phase. Observe that Algorithm 4 takes at most $2\log_{1+\epsilon} n$ iterations, meaning that the entire algorithm will be finished this phase entirely on $\mathcal{M}$.

Combining these two cases with respect to k, $\mathcal{A}_{\text{MPC}}$ takes $O(1/\delta)$ rounds. □

F. Proof of Theorem 5.2

Proof. Let $\mathcal{A}_{\text{MPC}}$ be the algorithm described in the proof of Theorem 5.1. We extend $\mathcal{A}_{\text{MPC}}$ this algorithm to obtain the desired round complexity. In this proof, we only describe the extension.

Algorithm description. In a phase i, we calculate $E_G(u, T_i)$ and $E_G(S_i, v)$ for all $u \in S_i, v \in T_i$. Then, if $|S_i|/|T_i| \geq c$, we peel from set T using VSETS-UPDATE until the resulting sets (S_i', T_i') satisfy $|S_i'|/|T_i'| < c$. Similarly, if $|S_i|/|T_i| < c$, we peel from set S using VSETS-UPDATE until the resulting sets (S_i', T_i') satisfy $|S_i'|/|T_i'| \geq c$. Finally, we do what $\mathcal{A}_{\text{MPC}}$ does every phase except that the edge samples now have size $(|S| + |T|) \cdot \xi$.

Algorithm analysis. Consider a phase i of our algorithm and let (S_i, T_i) be our vertex sets before the peeling starts. Without loss of generality, assume that $|S_i|/|T_i| \geq c$. Note that calculating the degrees of vertices and peeling from set T can all be done in a constant number of rounds. Afterwards, we have that the resulting sets (S_i', T_i') satisfy $|S_i'|/|T_i'| < c$. This guarantees that both vertex sets are peeled from.

Now, let $n' = |S_i'| + |T_i'|$. We consider some pair of vertex sets (S', T') while running Algorithm 4 during phase i. With samples of $n'\xi$ edges, we follow a similar proof of Theorem 5.1 to see that if $k \geq \frac{n \operatorname{poly}(\log(n)/\varepsilon)}{2\log_{1+\epsilon} n}$, either E_{rel} fits entirely on $\mathcal{M}$ in the next phase or $E_G(S', T')$ has $O\left(\frac{n' \log^2 n}{\epsilon^3 n \operatorname{poly}(\log(n)/\varepsilon)} \cdot |E_{\text{rem}}|\right)$ edges. With the latter, the number of edges between vertex sets S and T reduces by a factor of $x = \Omega\left(\frac{\epsilon^3 n \operatorname{poly}\log n}{n' \log^2 n}\right)$. Since both vertex sets are peeled from, n' reduces by at least a factor of $1+\epsilon$ every phase. Then, the algorithm takes $O(\sqrt{\log_{1+\epsilon} n})$ rounds if every phase has at least one pair of vertex sets with $k \geq \frac{n \operatorname{poly}(\log(n)/\varepsilon)}{2\log_{1+\epsilon} n}$ because x reduces by at least a factor of $1+\epsilon$ every phase as well due to n'.

With $k < \frac{n \operatorname{poly}(\log(n)/\varepsilon)}{2\log_{1+\epsilon} n}$ for all pair of vertex sets in a phase, the entire algorithm can be finished in this phase entirely on $\mathcal{M}$. So, the algorithm takes $O(\sqrt{\log_{1+\epsilon} n})$ rounds. □

G. Additional empirical evaluation

In this section, we provide empirical evaluation in addition to those given in the main paper. We refer a reader to Section 6 for details on the baselines, datasets, and computational setup.

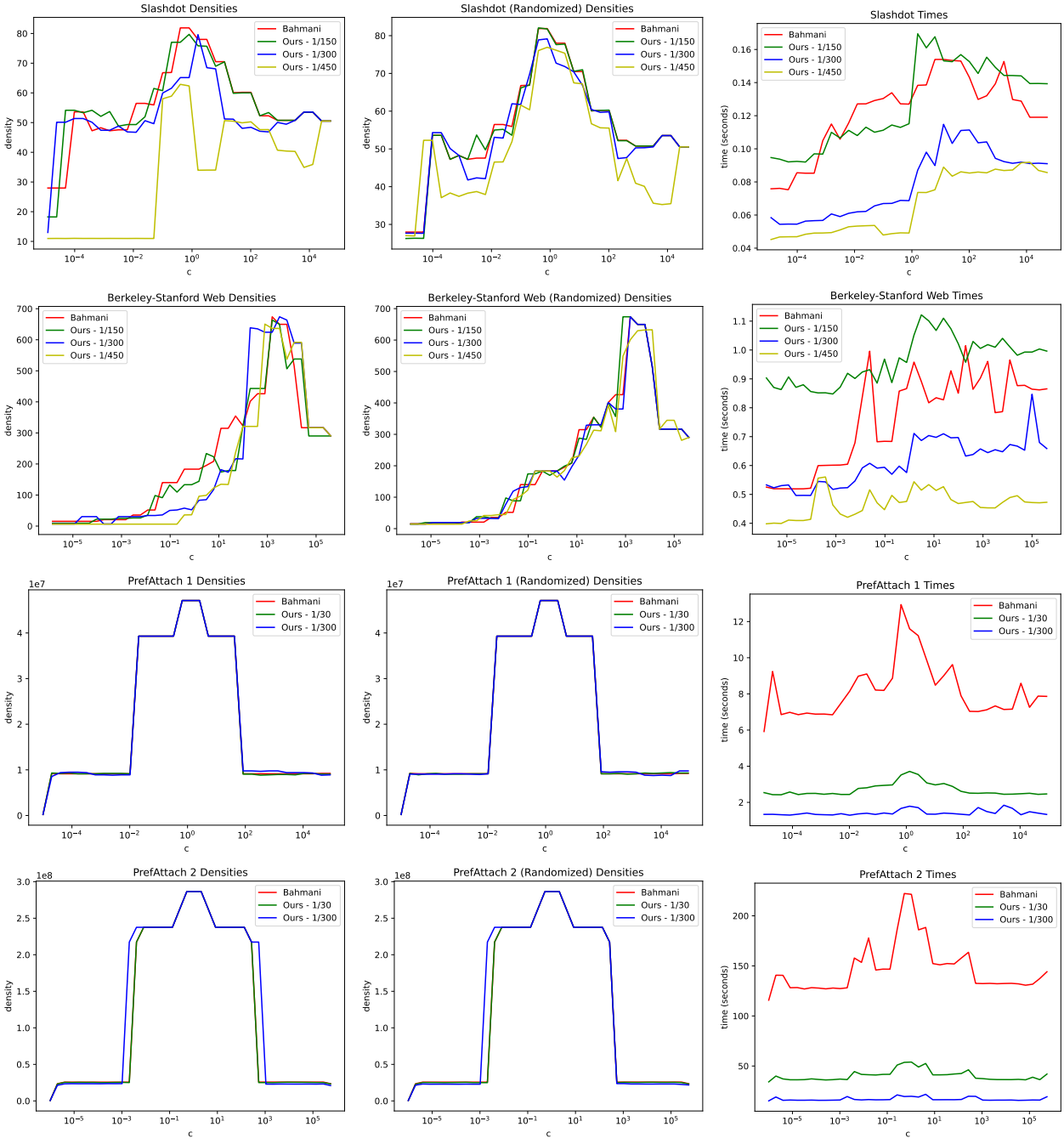

Figure 5. Density and running-time as a function of c for remaining data sets, with and without randomization

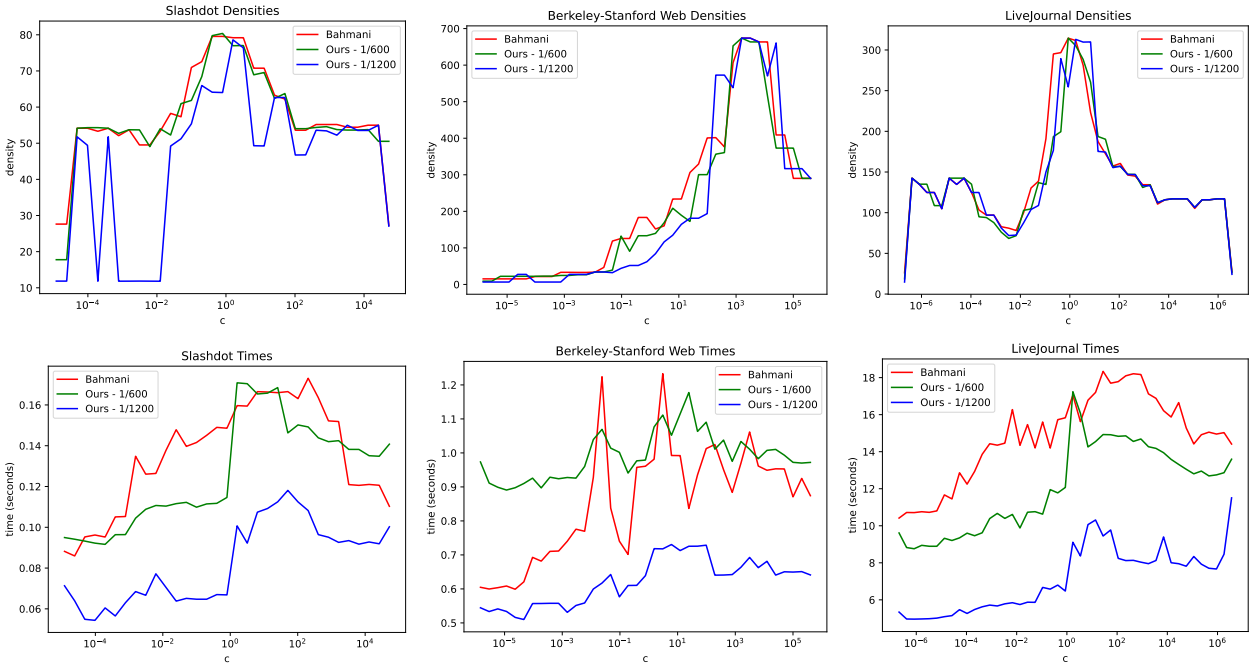

Figure 6. Density and running-time as a function of c for some data sets with $\epsilon = 0.1$ and $f = 1/600, 1/1200$

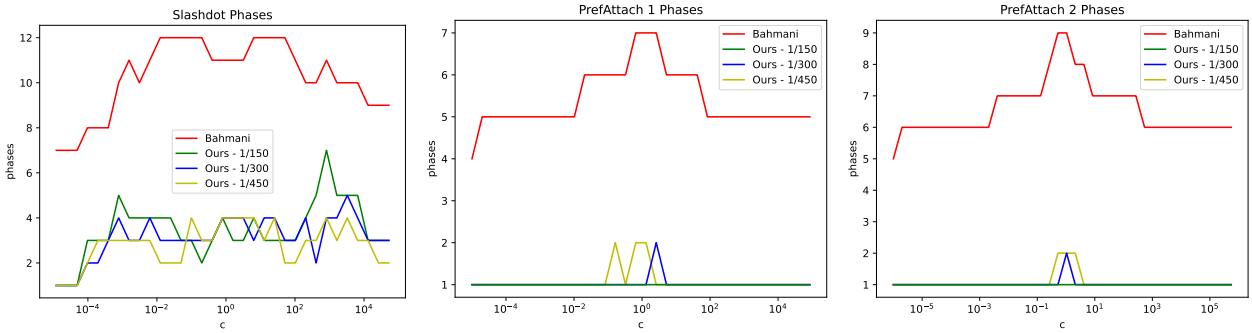

Figure 7. Number of phases as a function of c for remaining data sets with $f = 1/150, 1/300, 1/450$.