Improved Parallel Algorithms for Density-Based Network Clustering

Mohsen Ghaffari [*1] **Silvio Lattanzi** [*2] **Slobodan Mitrović** [*3]

Abstract

Clustering large-scale networks is a central topic in unsupervised learning with many applications in machine learning and data mining. A classic approach to cluster a network is to identify regions of high edge density, which in the literature is captured by two fundamental problems: the densest subgraph and the k-core decomposition problems. We design massively parallel computation (MPC) algorithms for these problems that are considerably faster than prior work. In the case of k-core decomposition, our work improves exponentially on the algorithm provided by Esfandiari et al. (ICML'18). Compared to the prior work on densest subgraph presented by Bahmani et al. (VLDB'12, '14), our result requires quadratically fewer MPC rounds. We complement our analysis with an experimental scalability analysis of our techniques.

1. Introduction

Density-based clustering is a classic technique in unsupervised learning. In this field, extracting dense subgraphs, i.e., subgraphs that have large edge-to-vertex ratio, is a basic primitive used in a wide range of machine learning and data analysis tasks. In fact, thanks to their nice structural properties (for example they often correspond to well-connected components that are also robust to outliers and noise), extracting dense subgraphs has proved to be useful in spam detection (Gibson et al., 2005), finding communities in social networks (Leskovec et al., 2008; Chen & Saad, 2012; Gionis & Tsourakakis, 2015), computational biology (Fratkin et al., 2006; Saha et al., 2010), and detecting common patterns (Liu & Yan, 2010; Chen et al., 2011).

In literature, there are two classic formalizations of density-based clustering in graphs: the densest subgraph problem and the k-core decomposition problem. In the densest subgraph problem, we are interested in finding the subgraph of highest density in a graph. This problem was originally introduced by Goldberg (Goldberg, 1984) and since then, it attracted extensive attention from the machine learning, data mining, and algorithmic community (Esfandiari et al., 2015; Bahmani et al., 2012; Epasto et al., 2015; Jethava et al., 2012; Miller et al., 2010; Papailiopoulos et al., 2014). In the k-core decomposition, a given network is decomposed into vertex-layers based on their local connectivity. Namely, for each node v this decomposition determines the maximum threshold k such that there exists a subgraph whose minimum degree is at least k and the subgraph contains v. Such a decomposition is very useful in practice and it induces a natural hierarchical clustering on any graph. For this reason, the k-core decomposition has found many applications in analyzing dynamic of social networks (Bhawalkar et al., 2015), in community detection (Mitzenmacher et al., 2015), in visualization of large-scale complex networks (Alvarez-Hamelin et al., 2005), in analyzing protein-protein interaction (Altaf-Ul-Amin et al., 2006) and, more broadly, is used for detecting dense components of graphs (Lee et al., 2010b).

In modern computational tasks, we often need to cope with very large networks. Thus, it becomes increasingly more important to design efficient parallel algorithms for unsupervised learning problems. Obtaining such algorithms is often challenging because a bulk of graph problems are inherently sequential, e.g., their structure is defined iteratively and in an adaptive manner. For instance, finding the k-core decomposition of a graph on n vertices is a simple task if the underlying graph is sufficiently small to fit on a single machine. To solve this problem sequentially, it suffices to iteratively select a vertex v of the smallest degree, set that degree to be the label of v, and remove v from the graph. Unfortunately, implementing this approach directly as a parallel algorithm requires n rounds of computation, which is prohibitive. Nevertheless, in many scenarios, if we allow approximate but relatively accurate solutions, then the problem becomes tractable. Motivated by this, we study the approximate k-core decomposition and the approximate densest subgraph problems, and provide efficient parallel algorithms for them. Next, we discuss the

[*]Equal contribution [1]ETH Zurich [2]Google Research Zurich [3]MIT. Correspondence to: Mohsen Ghaffari <ghaffari@inf.ethz.ch>, Silvio Lattanzi <silviol@google.com>, Slobodan Mitrović <slobo@mit.edu>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).

precise model of parallelism that we use in this work.

The MPC model. We provide algorithms for the massively parallel computation (MPC) model, which is a theoretical abstraction of practical settings such as MapReduce (Dean & Ghemawat, 2008), Hadoop (White, 2012), Spark (Zaharia et al., 2010) and Dryad (Isard et al., 2007). The MPC model (Karloff et al., 2010; Goodrich et al., 2011; Beame et al., 2013) is considered de-facto the standard theoretical model for large-scale parallel computing.

Computation in MPC proceeds in synchronous parallel *rounds* over multiple machines. Each machine has memory S. At the beginning of a computation, data is partitioned across the machines. During each round, machines process data locally. At the end of a round, machines exchange messages with a restriction that each machine is allowed to send messages and also receive messages of total size S. The efficiency of an algorithm in this model is measured by the number of rounds it takes for the algorithm to terminate and by the size of the memory of every machine. An interesting regime of MPC computation is when $S \ll N$, where N is the total size of the input, as otherwise all the data can be stored on a single machine. With respect to S, there are three main regimes that have been studied in the context of parallel algorithms for graphs on n vertices: the sublinear regime $S \in O(n^\delta)$, the almost linear regime $S \in \tilde{O}(n)$, and the superlinear regime $S \in O(n^{1+\delta})$, where δ is an arbitrary constant. We begin by presenting a simple algorithm for approximate k-core decomposition in the regime $S \in \tilde{O}(n)$. Then, we focus on the most restrictive regime $S \in O(n^\delta)$, in which we study both approximate k-core decomposition and approximate densest subgraph.

1.1. Our Contributions

k-core decomposition. As our first result we present a simple algorithm for computing $(1+\varepsilon)$-approximate k-core decomposition for all the vertices simultaneously when the memory per machine S is $\tilde{O}(n)$. In the same regime, (Esfandiari et al., 2018) show how to obtain such decomposition in $O(\log n)$ MPC rounds, while our algorithm requires only $O(\log\log n)$ many rounds.

Theorem 1. *There is an algorithm that for any constant $\varepsilon \in (0,1)$ in $O(\log\log n)$ rounds whp computes a $(1+\varepsilon)$-approximate coreness value for each vertex. This algorithm requires $\tilde{O}(n)$ memory per machine.*

We also consider a more restricted memory-per-machine regime in which each machine is allowed to have maximum load of $S = \tilde{O}(n^\delta)$ bits, for any fixed constant $\delta > 0$. To the best of our knowledge no algorithm was known for this regime prior to our work.

Theorem 2. *There is an algorithm that for any constant $\varepsilon \in (0,1)$ in $O(\sqrt{\log n} \cdot \log\log n)$ MPC rounds whp computes a $(2+\varepsilon)$-approximate coreness value for each vertex. The algorithm uses $\tilde{O}(n^\delta)$ memory per machine, for an arbitrary constant $\delta \in (0,1)$, and the total memory of $\tilde{O}(\max\{m, n^{1+\delta}\})$.*

By extending our ideas for the k-core decomposition, we also improve the state-of-the-art of parallel computation of graph orientation. Due to the space constraints, we present this result in Appendix D.

Densest subgraph. Given a graph G, (Esfandiari et al., 2015) show that a densest subgraph of $\tilde{O}(n)$ randomly and independently sampled edges of G is a $(1+\varepsilon)$-approximate densest subgraph of G. When $S \in \tilde{O}(n)$, it is easy to translate this idea into an algorithm for finding $(1+\varepsilon)$-approximate densest subgraph in $O(1)$ MPC rounds.

In the regime $S \in \tilde{O}(n^\delta)$ bits, for any fixed constant $\delta > 0$, it is implicit in the work of (Bahmani et al., 2014) that there exists an algorithm for computing $(1+\varepsilon)$-approximate densest subgraph in $O(\log n)$ rounds. This algorithm uses the Multiplicative Weights Update (MWU) framework, and crucially relies on executing each MWU iteration in a separate MPC round. Since MWU executes at least $O(\log n)$ iterations, translating this approach directly to MPC requires at least $O(\log n)$ rounds. Our contribution is that we show how to compress many MWU iterations into $O(1)$ MPC rounds. This enables us to find a $(1+\varepsilon)$-approximate densest subgraph in $\tilde{O}(\sqrt{\log n})$ MPC rounds, and thus quadratically improve on the prior work.

Theorem 3. *There is an algorithm that for any constant $\varepsilon \in (0,1)$ in $O(\sqrt{\log n} \cdot \log\log n)$ MPC rounds whp computes a $(1+\varepsilon)$-approximate densest subgraph. The algorithm uses $\tilde{O}(n^\delta)$ memory per machine, for an arbitrary constant $\delta \in (0,1)$, and the total memory of $\tilde{O}(\max\{m, n^{1+\delta}\})$.*

1.2. Related Work

Several unsupervised problems have been studied in the MPC model, including metric clustering (Bateni et al., 2014; Ene et al., 2011), anomaly detection (Akoglu et al., 2009), etc. In this paper, we focus on graph clustering and in particular on density-based clustering.

Both k-core decomposition and the densest subgraph problems have been extensively studied in literature. The most related line of work is about MPC algorithms for these problem (Esfandiari et al., 2018; Bahmani et al., 2012; 2014). As mentioned earlier we improve the state of the art for both problems.

Another related area of work is about streaming algorithms for density-based problems. There are several solutions for the densest subgraph (Lee et al., 2010a; Esfandiari et al., 2015; Bahmani et al., 2012; Epasto et al., 2015; Bhattacharya et al., 2015) and an algorithm for k-core decom-

position (Esfandiari et al., 2018). The main idea behind those papers is either to use peeling or sampling to obtain sparser graphs and to solve the problem there. By combining these ideas it is possible to obtain $O(\log n)$ MPC rounds algorithm for densest k-core decomposition and the densest subgraph, but to overcome the $O(\log n)$ parallel round barrier we need to introduce in this paper new ideas and techniques.

2. Preliminaries

Given a graph $G = (V, E)$ and $v \in V$, we use $N_G(v)$ to denote the set of neighbors of v in G. When it is clear from the context, we omit the subscript and write $N(v)$ instead. We use $d_G(v) \stackrel{\text{def}}{=} |N_G(v)|$ (and similarly $d(v) \stackrel{\text{def}}{=} |N(v)|$) to refer to the degree of v in G.

We use $\tilde{O}(f)$ to hide $\log^c f$ factors, for any constant $c \geq 0$. In particular, $\tilde{O}(\log n)$ hides $\log^c \log n$. By saying that an event $\mathcal{E}$ happens with high probability (whp), we refer that $\Pr[\mathcal{E}] \geq 1 - n^{-c}$ for some constant $c \geq 1$.

k-core decomposition. A k-core of a graph G is a maximal subgraph H of G such that the minimum vertex-degree of H is at least k. We say that v has *coreness number* k (or only "coreness k") if it belongs to k-core but not to $(k+1)$-core. Also, for $\alpha \geq 1$, k' is an α-approximate coreness value of a vertex of coreness k if $k' \in [k, \alpha k]$.

In our analysis, we will use the following fact.

Observation 4. *Let $S \subseteq V$ be the set of all vertices that have coreness at most k. Then, number of the edges incident to S is at most $k|S|$.*

Densest subgraph. Given an undirected graph $G = (V, E)$ and a set $S \subseteq V$, the *density* $d(S)$ is defined as $d(S) \stackrel{\text{def}}{=} |E(S)|/|S|$, where $E(S)$ is the set of the edges of the subgraph induced by S. A *densest subgraph* $S^\star$ is a set such that $S^\star \in \arg\max_{S \subseteq V} d(S)$. Then, $T \subseteq V$ is a α-approximate densest subgraph of G, for $\alpha \geq 1$, if $d(T) \geq d(S^\star)/\alpha$.

3. $(1+\varepsilon)$-approximate Coreness with $\tilde{O}(n)$ Memory per Machine

In this section we prove Theorem 1, that improves exponentially on the round complexity obtained by (Esfandiari et al., 2018). Compared to prior work, our algorithm samples dense regions less aggressively. Intuitively, this allows us to collect a relatively large induced subgraph on each machine and handle a wide range of coreness values in $O(1)$ MPC rounds.

Intuitive Discussions About the Algorithm. The starting point of our approach is an algorithmic primitive that for a given threshold k and a graph G labels each vertex by ABOVE if its coreness is at least k and by BELOW otherwise. To achieve that, this primitive (Algorithm 1) iteratively removes all the vertics whose current degree is less than k. The algorithm proceeds in this manner as long as there is at least one such vertex. All the vertices removed in this process are labeled by BELOW, while the remaining vertices are labeled by ABOVE.

Input : G : a graph
k : coreness threshold
Output : Label each vertex of G by ABOVE if its coreness is at least k and label by BELOW otherwise

while do
  Let S be the set of all the vertices $v \in G$ for which $d_G(v) < k$.
  if $S \neq \emptyset$ **then**
    Label all the vertices of S by BELOW.
    Remove S from G.
  else
    Break.
Label each unlabeled vertex of G by ABOVE.
return the obtained labels

Algorithm 1: A centralized algorithm for computing vertices with coreness above k

Even for a single k, Algorithm 1 might execute $\Theta(n)$ iterations, hence its direct implementation in MPC is inefficient. Nevertheless, by proper sparsification it is possible to transform this algorithm to compute $(1+\varepsilon)$-approximate coreness values in $O(\log n)$ rounds, as shown by (Esfandiari et al., 2018). Their algorithm considers coreness values from the largest to the smallest, grouping values $(2^{i-1}, 2^i]$ together. When a range $(2^{i-1}, 2^i]$ is processed, all the edges whose both endpoints have coreness value larger than 2^i are ignored. The remaining edge-set is sparsified by keeping each edge with probability $\Theta(2^i \log n/(\varepsilon^2 n))$. This step reduces the maximum coreness value to $O(\log n/\varepsilon^2)$, which by Observation 4 implies that the resulting graph can be stored on a single machine. Moreover, degrees of vertices with coreness $(2^{i-1}, 2^i]$ are concentrated around their expectation. This implies an algorithm that in $O(1)$ MPC rounds labels all the vertices with coreness in $(2^{i-1}, 2^i]$, assuming that all the vertices of coreness higher than 2^i have already been computed.

A natural idea to improve the work of (Esfandiari et al., 2018) is to process the coreness values in ranges wider than $(2^{i-1}, 2^i]$. This, however, implies that the sparsification described above has to be less aggressive, which in turn results in a graph consisting of more than $\tilde{O}(n)$ edges and hence not fitting on one machine. To overcome this barrier, we use a vertex-based sampling idea. A similar idea was introduced by (Czumaj et al., 2018) for the maximum

matching problem. This partitioning enables us to sample *multiple* subgraphs each of which captures a wide ranges of coreness values. Intuitively, this work well because by sample by vertices we do not need to send all the nodes to a single machines and furthermore we can sample more aggressively. In the experimental section we will confirm the impact this intuition also experimentally. Each subgraph is then processed on a different machine. We present this algorithm in Section 3.1, and build on it in Section 3.2 to compose our final MPC algorithm.

3.1. A Warm-up Section

We now describe Algorithm 2, that given a parameter k and a suitably chosen induced subgraph H of G marks vertices of H that have coreness at least k in G.

For a parameter $p \in [0,1]$, let H be an induced graph obtained by sampling each vertex of G with probability p and independently of other vertices. Algorithm 2 labels by BELOW each vertex of H whose degree in H is less than $(1-\varepsilon/2)pk$, and labels the vertex by ABOVE otherwise. Although the vertices are labeled with respect to their degrees in H, we show that each vertex of $V(G) \cap V(H)$ having coreness at least k and each vertex having coreness at most $(1-\varepsilon)k$ in G is labeled correctly. This statement is formalized by the following lemma, whose proof is deferred to Appendix B.1.

Input : H : a graph
ε : approximation parameter
k : coreness threshold
p : sampling probability
Output : Labeling each vertex of H by ABOVE if its coreness is at least k and labeling by BELOW if its coreness is at most $(1-\varepsilon)k$

1 **while do**
2 Let S be the set of all the vertices $v \in H$ for which $d_H(v) < (1-\varepsilon/2)pk$.
3 **if** $S \neq \emptyset$ **then**
4 Label all the vertices of S by BELOW.
5 Remove S from H.
6 **else**
7 Break.
8 Label each unlabeled vertex of H by ABOVE.
9 **return** the obtained labels

Algorithm 2: Computing a $(1+2\varepsilon)$-approximate coreness with respect to a threshold

Lemma 5. *Let $p \in [0,1]$ and $p \geq \min\{1, 50\frac{\log n}{k\varepsilon^2}\}$. Let V_H be a vertex subset of G obtained by sampling each vertex from G with probability p and independently of other vertices. Let H be the graph induced on V_H. Then, for given H, ε, k and p, Algorithm 2 whp labels the vertices of H satisfying the following:*

(A) The vertices of H that have coreness at least k in G are labeled by ABOVE.

(B) The vertices of H that have coreness at most $(1-\varepsilon)k$ in G are labeled by BELOW.

3.2. The Main MPC Algorithm

Given a single coreness threshold k, Algorithm 2 approximately labels the vertices of a subgraph of G. We build on Algorithm 2 to approximately label all the vertices of G for all the relevant thresholds. We achieve this in two steps. First, we assume that all the vertices with coreness more than k are already properly labeled. Then, we design a method (Algorithm 3) that finds $(1+2\varepsilon)$-approximate coreness of all the vertices of G whose coreness is between $k^{0.9}$ and k. Moreover, this algorithm can be implemented in only $O(1)$ MPC rounds. Second, we use Algorithm 3 to label the vertices in batches as described next.

Let $k_{\max} = k$ be the maximum coreness of the unlabeled vertices so far. Then, Algorithm 3 is invoked to label all the vertices having coreness between $k^{0.9}$ and k. After Algorithm 3 terminates, it is then invoked with $k_{\max} = k^{0.9}$. It proceeds in this manner until $k_{\max}$ becomes $\log^2 n$. Once $k_{\max} \leq \log^2 n$, all the vertices are gathered on one machine and the labeling is performed locally. This in total requires $O(\log\log n)$ MPC rounds. In Section 3.3 we comment how to implement this while using $\tilde{O}(n)$ memory per machine.

Next we describe Algorithm 3. That is, we show how to label all the vertices having coreness between $k_{\max}^{0.9}$ and $k_{\max}$, assuming that all the vertices having coreness more than $k_{\max}$ have been labeled correctly. Algorithm 3 in parallel considers all the coreness thresholds of the form $(1+\varepsilon)^i \in [k_{\max}^{0.9}, k_{\max}]$ (see Line 2). Then, for a fixed threshold the algorithm partitions G into $1/p$ subgraphs (see Line 4). For each subgraph in parallel it invokes Algorithm 2. In this way, for each vertex v Algorithm 3 gets a sequence of labels saying whether v has coreness above or below each of the considered thresholds. Finally, Algorithm 3 sets the coreness of v to be the highest threshold, if any, for which v got label ABOVE.

The value p defined on Line 1 of Algorithm 3 satisfies the conditions of Lemma 5. Hence, the correctness follows by Lemma 5.

3.3. MPC Implementation

Let p be as defined on Line 1 of Algorithm 3. By Observation 4, the number of edges incident to all the vertices of coreness at most k is $O(kn)$. Furthermore, the probability of one of those edges being in H_i is p^2, i.e., an edge appears in H_i only if both of its endpoints appear there. Hence, H_i

Input : G : a graph
ε : approximation parameter
$k_{\max}$: maximum unlabeled coreness
l : the current labels of vertices
Output : Updated l for vertices of coreness between $k_{\max}^{0.9}$ and $k_{\max}$

$p \leftarrow \min\left\{1, 50\frac{\log n}{k_{\max}^{0.5}\varepsilon^2}\right\}$
$\mathcal{K} \leftarrow \{(1+\varepsilon)^i : i \in \mathbb{N} \text{ and } (1+\varepsilon)^i \in [k_{\max}^{0.9}, k_{\max}]\}$.
foreach $k \in \mathcal{K}$ **do in parallel**
  Partition $V(G)$ across $1/p$ machines. Each vertex is assigned to one of the machines independently and uniformly at random.
  Let G_i be an induced graph on machine i.
  Let H_i be obtained by removing the edge of G_i whose both endpoints are labeled by l.
  Pass H_i, ε, k and p to Algorithm 2 and record the returned labels.
foreach $v \in V(G)$ **unlabeled by** l **do**
  Let $k \in \mathcal{K}$ be the largest k for which Algorithm 2 labeled v by ABOVE.
  If such k exists, set $l(v) \leftarrow k$.
return the updated labels l

Algorithm 3: Labeling vertices with coreness between $k_{\max}^{0.9}$ and $k_{\max}$

in expectation contains $O(p^2kn) \in O(n\log^2 n/\varepsilon^4)$ many edges. In Appendix B.2 we prove the following statement, that shows that this bound holds whp as well.

Lemma 6. *H_i whp contains $O(n\log^2 n/\varepsilon^4)$ many edges.*

This concludes the proof of Theorem 1.

4. $(2+\varepsilon)$-approximate Coreness with Sublinear Memory per Machine

This section is devoted to proving Theorem 2. Similarly as in Section 3, we reduce the task of computing $(2+\varepsilon)$-approximate coreness to one that for a given threshold k labels by BELOW all the vertices of coreness less than k and potentially some of those having coreness between k and $(2+\varepsilon)k$.

Intuitive explanation of our approach. We first present an $O(\log n)$ MPC round process for computing $(2+\varepsilon)$-approximate coreness, and then explain how to transform it to an algorithm that runs in $\tilde{O}(\sqrt{\log n})$ rounds. We first observe that if all the vertices that have degree at most $(2+\varepsilon)k$ are simultaneously removed, then the number of vertices of coreness at most k reduces by at least $\varepsilon/(2+\varepsilon)$ fraction. (This statement, that we make formal in our proofs, follows from Observation 4.) Hence, by repeatedly removing the vertices of degree at most $(2+\varepsilon)k$ for $O(\log n)$ rounds divides the vertices on those having coreness at most and those having coreness more than $(2+\varepsilon)k$. Running this method in parallel for all the coreness thresholds $(1+\varepsilon)^i$ leads to the desired coreness decomposition. To obtain $\tilde{O}(\sqrt{\log n})$ round complexity, we split these $O(\log n)$ iterations of vertex-removal into $\Theta(\sqrt{\log n})$ phases, each phase consisting of $T \in \Theta(\sqrt{\log n})$ iterations. Then, one phase for each vertex v is executed by gathering a carefully chosen part of T-hop neighborhood of v (that can be done in $O(\log T)$ MPC rounds), and locally executing T iterations for v on the gathered neighborhood. This approach is motivated by a recent work of (Ghaffari & Uitto, 2019). To maintain the neighborhood size within the memory limit of n^δ, our algorithm carefully selects vertices of "large" degree and ignores/freezes them in this process. As we show, the "large degree" is chosen in such a way that it affects the round complexity only by little. We next describe our method for simulating one phase, called Algorithm 4.

In the initialization, Algorithm 4 marks as *frozen* each vertex of degree more than $2k \cdot 2^{\sqrt{\delta\log n}}$. Then, all the edges whose *both* endpoints are frozen are marked as frozen. After that, the algorithm proceeds in T iterations. In each iteration is sampled a subset of non-frozen edges. Each edge is sampled with probability $p = \frac{C\log n}{\varepsilon^2 k}$, for some constant C, and independently of other edges. Then, all the non-frozen vertices having degree less than $(2+\varepsilon)kp$ are labeled by BELOW, indicating that their coreness is below k. The edges incident to all such labeled vertices are removed.

In Section 4.2 we show that invoking this algorithm $\Theta(\sqrt{\log n})$ times suffices to label all the vertices of coreness at most k by BELOW. It is easy to see that Algorithm 4 can be simulated in $\Theta(\sqrt{\log n})$ MPC rounds. In Appendix C.2, we show how to simulate this algorithm in only $O(\log\log n)$ many MPC rounds.

4.1. Analysis of Algorithm 4

The next lemma bounds the number of frozen vertices in the initialization step.

Lemma 7. *Let $V_{\le k}$ be the set of vertices of G with coreness at most k. Let F be the vertices of coreness at most k frozen by* INITIALIZE *of Algorithm 4. Then,*

$$|F| \le \frac{|V_{\le k}|}{2^{\sqrt{\delta\log n}}}.$$

Proof. From Observation 4, $V_{\le k}$ has at most $k\cdot|V_{\le k}|$ edges incident to it. Since each vertex of F has degree more than $2k\cdot 2^{\sqrt{\delta\log n}}$, we have $|F| \le \frac{2k|V_{\le k}|}{2k\cdot 2^{\sqrt{\delta\log n}}}$. □

Set ε' equals $\varepsilon/3$. We also state the following concetration result, that follows directly by applying a Chernoff bound.

Input : G : a graph
ε : approximation parameter
k : coreness threshold
Output : Label by BELOW some vertices of G of coreness at most $(2+3\varepsilon')k$

/* INITIALIZE: */
1 Freeze all vertices of degree greater than $2k \cdot 2^{\sqrt{\delta \log n}}$.
2 Mark as frozen each edge with both endpoints frozen.
3 $\varepsilon' \stackrel{\text{def}}{=} \varepsilon/3$

/* PEELINGBELOW-k: */
4 $p \leftarrow \min\left\{1, \frac{10 \log n}{(\varepsilon')^2 k}\right\}$
5 $T \leftarrow \frac{\sqrt{\delta \log n}}{5}$
6 **for** T *steps* **do**
7 Sample each of the non-frozen edges with probability p. Let G' be the sampled graph.
8 Let S be the set of all non-frozen vertices $v \in G'$ for which $d_{G'}(v) < (2+2\varepsilon')pk$.
9 Label all the vertices of S by BELOW.
10 Remove S from G.
11 **return** the obtained labels

Algorithm 4: Labeling vertices by BELOW

Lemma 8. *Let S be the set obtained at Line 8 of Algorithm 4. Then, whp it holds:*

- *If $d_G(v) \le (2+\varepsilon')k$, then v is added to S.*
- *If $d_G(v) \ge (2+3\varepsilon')k$, then v is not added to S.*

Each iteration of PEELINGBELOW-k detects some vertices of coreness at most k and labels them by BELOW. The main property of this process is that in the next iteration the number of unlabeled vertices of coreness at most k is already small, or that their number drops significantly.

Lemma 9. *Let $V^i_{\le k}$ be the set of vertices of G with coreness at most k (including frozen ones) that are not yet labeled by the i^{th} iteration of* PEELINGBELOW-k *of Algorithm 4. Then, it holds*

$$|V^T_{\le k}| \le \left(2^{\sqrt{\delta \log n}}\right)^{-\varepsilon'/40} |V^1_{\le k}|. \tag{1}$$

4.2. The Main MPC Algorithm

We now describe our main MPC algorithm. Lemma 9 shows that executing Algorithm 4 reduces the number of unlabeled vertices of coreness at most k by a factor of $\left(2^{\sqrt{\delta \log n}}\right)^{\varepsilon'/40}$. Hence, to properly label by BELOW all the vertices whose coreness is below k, we invoke Algorithm 4 for $40 \cdot \sqrt{\delta \log n}/\varepsilon'$ times. Note that a vertex of coreness more than $(2+\varepsilon)k$ has at least $(2+\varepsilon)k$ incident vertices of coreness more than $(2+\varepsilon)k$. Hence, by Lemma 8 (recall that $\varepsilon' = \varepsilon/3$), no vertex of coreness more than $(2+\varepsilon)k$ is labeled by BELOW.

These executions are done for each coreness threshold $(2+\varepsilon)^i$ in parallel. Then, the coreness of vertex v is the largest threshold for which it has not been labeled by BELOW.

MPC Implementation. To obtain the round complexity of $\tilde{O}(\sqrt{\log n})$, we implement Algorithm 4 in $O(\log \log n)$ rounds as follows. We simulate PEELINGBELOW-k block (that begins at Line 4) by collecting all the relevant information in T-hop neighborhood of each vertex. This can be done by graph exponentiation in $O(\log \log n)$ rounds. After that, each vertex independently and locally simulates PEELINGBELOW-k. In Appendix C.2 we provide details on how to efficiently gather these T-hop neighborhoods.

5. $(1+\varepsilon)$-approximate Densest Subgraph with Sublinear Memory per Machine

We now focus on proving Theorem 3. We reduce this theorem to the task that for a given D either finds a subgraph of density at least $(1-\varepsilon)D$, or reports that such subgraph does not exist. Then, to find a $(1+\varepsilon)$-approximate densest subgraph, it suffices to execute this task for all the values $(1+\varepsilon)^i$ in place of D, and output the densest subgraph found in this way. Moreover, it is known (e.g., by (Esfandiari et al., 2015)) that by proper sampling we can assume that it holds $D \in O(\log n/\varepsilon^2)$. So, in the rest, for a constant $\varepsilon \in (0,1)$ we assume that we are given a value $D \in O(\log n)$, and our goal is to answer whether the input graph has a subgraph of density $(1-\varepsilon)D$ or not.

Starting point of our approach is the work by (Bahmani et al., 2014). That is, we design an algorithm for approximately solving the dual of the LP relaxation of the densest subgraph problem. We review this LPs in Section 5.1. In this way, we obtain a fractional dual solution. It was shown by (Bahmani et al., 2014) (as we recall in Appendix E.3) how to round this fractional to a $(1+\varepsilon)$-approximate integral primal solution, i.e., to round to a $(1+\varepsilon)$-approximate densest subgraph.

To solve this LP, (Bahmani et al., 2014) employ the Multiplicative Weights Update (MWU) method (that we recall in Section 5.2). They massage the dual LP so that its width becomes a constant, and then solve each MWU iteration in $O(1)$ many MPC rounds. Since, even with the reduced width, MWU requires at least $O(\log n)$ iterations of computation, their approach requires at least $O(\log n)$ MPC rounds. We also use MWU, but employ this method in a different way. Let $T = \Theta(\sqrt{\delta \log n})$. We provide a way

to collect all the relevant information for each vertex in its T-hop neighborhood so that it has size $O(n^\delta)$. Then, we use this information to execute T MWU iterations in $O(1)$ MPC rounds. Note that, however, even though that $D \in O(\log n)$, a vertex can have degree $\Theta(n)$. Hence, even a 1-hop neighborhood can contain all the vertices, and so cannot be stored on one machine. As our main result, we show how to reduce the degree of each vertex to only $\tilde{O}(2^T)$ in a way that all the relevant information is preserved for executing T MWU iterations for each vertex.

5.1. LP View

We now state the LP formulation of the densest subgraph problem (Charikar, 2000)

$$\begin{aligned} \text{maximize} \quad & \sum_e y_e \\ \text{subject to} \quad & y_e \le x_e \qquad \forall e \in E, e \text{ incident on } v \\ & \sum_v x_v \le 1 \\ & x_v, y_e \ge 0 \qquad \forall v \in V, \forall e \in E \end{aligned}$$

The dual LP of the LP above is

$$\begin{aligned} \text{minimize} \quad & z \\ \text{subject to} \quad & \alpha_{eu} + \alpha_{ev} \ge 1 \qquad \forall e = \{u, v\} \in E \\ & \sum_{e \text{ incident on } v} \alpha_{ev} \le z \qquad \forall v \in V \\ & \alpha_{ev} \ge 0 \qquad \forall e, v \end{aligned}$$

In the sequel, we focus on solving this dual LP. We reduce this task to the one of solving the feasibility problem obtained from the dual by fixing the value of z to D. Let $\textsc{Dual}(D)$ denote this feasibility question.

5.2. Multiplicative Weights Update Method

As $\textsc{Dual}(D)$ is a covering LP, in this section we provide a brief overview of the MWU method in the context of deciding feasibility of covering LPs, and refer a reader to Appendix E.2 for details. The feasibility question of a covering LP can be stated as follows:

Feasibility of Covering LP:
Given a convex set $P \subset \mathbb{R}^d$, a matrix $A \in \mathbb{R}^{r \times d}$ such that $Ax \ge 0$ for all $x \in P$, does there exist $y \in P$ such that $Ay \ge 1$?

To solve this problem by MWU, it is needed to provide access to the following oracle, that gets invoked by MWU:

$\textsc{Oracle}(w)$:
Given a vector $w \in \mathbb{R}^r_{\ge 0}$: return a vector x such that $w^T Ax \ge \|w\|_1$; otherwise report "fail".

The vector w is updated by MWU after each iteration, where the updates are a function of the output of $\textsc{Oracle}(w)$. Let ρ be such that $(Ax)_i \le \rho$ for any $x \in P$. The value ρ is called *width*. It is well-known that:

Theorem 10 ((Arora et al., 2012)). *Consider a feasibility covering LP problem of width ρ. Then, for any constant $\varepsilon \in (0, 1)$, after $\Theta(\rho \log r)$ iterations the MWU method either correctly reports that the covering problem is infeasible, or outputs a vector $\bar{x}$ such that such that $A\bar{x} \ge (1 - \varepsilon)1$.*

5.2.1. Applying MWU to $\textsc{Dual}(D)$

To solve $\textsc{Dual}(D)$ by MWU, we let the convex set P be the set of points corresponding to all but the first constraint of $\textsc{Dual}(D)$:

$$P = \{\alpha \in \mathbb{R}^{E \times V} : \sum_{e \text{ incident on } v} \alpha_{ev} \le D, \text{ and } \alpha_{ev} \ge 0\}.$$

Then, we use MWU to decide whether there exists a point $\alpha \in P$ such that $\alpha_{eu} + \alpha_{ev} \ge 1$ for all $e = \{u, v\} \in E$. A natural oracle for this problem is as follows. Recall that w corresponds to the constraints that we are aiming to satisfy. Hence, w is indexed by E. For each vertex v, $\textsc{Oracle}(w)$ selects an edge $e^\star_v$ such that

$$e^\star_v = \arg \max_{e \text{ incident to } v} w_e.$$

Let $\alpha^\star$ be the output of the oracle. Then, $\alpha^\star$ is set so that $\alpha^\star_{e^\star_v v} = D$ and $\alpha^\star_{ev} = 0$ for each $e \ne e^\star_v$.

It is not hard to see that this approach has width D, requiring $O(\log^2 n)$ iterations of MWU. (Recall that w.l.o.g. we assumed that $D \in O(\log n.)$ Nevertheless, as shown by (Bahmani et al., 2014), by adding constraints $\alpha_{ev} \le 2$, for all e, v, the width of $\textsc{Dual}(D)$ becomes $O(1)$ while the optimal solution remains the same. We discuss this width-reduction in more details in Appendix E.2.3. Hence, the massaged version of $\textsc{Dual}(D)$ can be solved in $O(\log n)$ iterations of MWU. Moreover, each of the iterations can be executed in $O(1)$ MPC rounds. The main challenge here is to execute these $O(\log n)$ MWU iterations in only $\tilde{O}(\sqrt{\log n})$ MPC rounds.

5.2.2. Compressing MWU Iterations

As earlier, let $T = \Theta(\sqrt{\delta \log n})$. Observe that $\textsc{Oracle}$ requires only local information of each vertex (access to its incident edges). Motivated by this, we will for each vertex collect its T-hop neighborhood and execute MWU locally. However, there are two difficulties with this approach. First, although D is relatively small, a vertex can have degree as large as $\Theta(n)$, e.g., in a star graph, hence its neighborhood would quickly become larger than the memory per machine. Second, even if we manage to somehow

sparsify the neighborhood of each vertex v based on the current value of w, ORACLE is invoked with different values of w from iteration to iteration and it is not a priori clear what will be the largest in w edges incident to v after several iterations. We now show how to overcome both of these barriers.

Assume for a moment that the degree of each vertex is $O(D) = O(\log n)$. Then, T-hop neighborhood of a vertex has size less than n^{δ}, and hence fits on one machine. Unfortunately, as discussed, a small D does not imply small degrees. However, a small D does imply sufficiently small degree for *most* of the vertices. To leverage this observation, for each vertex we keep only $\tilde{O}(2^T)$ incident edges with their values largest in the current vector w. This result in neighborhoods that fit on one machine. A downside of this approach is that neighborhoods of large-degree vertices do not have enough information to simulate MWU for T iterations, as the largest in w edges are changing from iteration to iteration. Nevertheless, we prove that *majority* of vertices simulate MWU correctly, which suffices to prove that our oracle solves the corresponding LP. We defer this proof to Appendix E.4.1.

6. Experiments

In this section we provide results of empirical evaluation of our k-core algorithms, while focusing on their scalability. In particular, we have two main goals:
– Understanding the effect of the vertex partitioning strategy in the performances of our algorithm from Section 3.
– Comparing the speed between an algorithm using almost linear memory and an algorithm(adapted from the algorithm presented in Section Section 4) using sublinear memory per machine.

Algorithms. We compare three different algorithms:
– SKC is the algorithm introduced in (Esfandiari et al., 2018) that is the current state-of-the-art algorithm for the k-core problem.
– VKC is similar to SKC, with the difference that we use the vertex partitioning strategy to sample the edges as in the algorithm presented in Section 3. We note that this algorithm does not capture all the optimization of the algorithm presented in Section 3, but it captures its main intuition.
– FKC is a simple version of the algorithm presented in Section 4 that runs in $O(\log n)$ rounds but uses sublinear memory.

Datasets. We test the performances of our algorithms on public graphs of increasing size from the SNAP Large Networks Data Collection (Yang & Leskovec, 2015). The datasets are described in Table 1.

Results. For each datasets and each algorithms we run three trials and in Fig. 1 we report the relative running time for the different algorithm. We note that for smaller graphs

Graph	# Nodes	# Edges
Amazon	334,863	925,872
Youtube	1,134,890	2,987,624
LiveJournal	3,997,962	34,681,189
Orkut	3,072,441	117,185,083
Friendster	65,608,366	1,806,067,135

Table 1. Description of the datasets analyzed in the experiments.

SKC and VKC have similar performances while for larger graphs, as Orkut, VKC outperforms SKC by more than a factor of 2. This shows that the vertex partitioning is quite impactful on large graph.

Interestingly FKC is significantly faster than all the other methods and scales to substantially larger networks, showing the importance of designing sublinear space algorithms. In fact, in practice, we observe that every round of FKC is significantly faster than a round of SKC or VKC. This is a consequence of the small amount of data processed during every round of FKC. Finally, we also note that for very large graphs as Friendster we could not even run SKC and VKC because of their linear memory requirement. Regarding the accuracy of the approximation we note that the quality of the solutions computed by SKC and VKC are very similar while FKC has slightly worse accuracy.

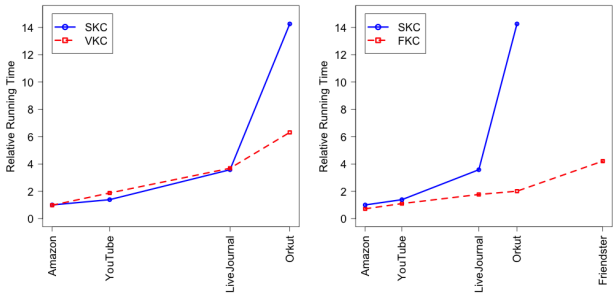

Figure 1. Comparison between running time of different algorithms. The x axes is in log-scale and it is index by the number of edges in the input graph. The y axes show the relative running time of the algorithms using as benchmark the running time of SKC on the Amazon graph.

7. Conclusions

We design new parallel algorithms for the densest subgraph and the k-core decomposition problems. We show that our algorithms outperform the state-of-the-art results both theoretically that empirically.

An interesting open question is to design an algorithm that computes a $(1+\epsilon)$-approximate k-core decomposition in $o(\log n)$ MPC rounds with sublinear memory per machine.

Acknowledgements

We thank the anonymous reviewers for their valuable feedback. S. Mitrović was supported by the Swiss NSF grant P2ELP2_181772 and MIT-IBM Watson AI Lab.

References

Akoglu, L., Mcglohon, M., and Faloutsos, C. Anomaly detection in large graphs. In *In CMU-CS-09-173 Technical Report*, 2009.

Altaf-Ul-Amin, M., Shinbo, Y., Mihara, K., Kurokawa, K., and Kanaya, S. Development and implementation of an algorithm for detection of protein complexes in large interaction networks. *BMC bioinformatics*, 7(1): 207, 2006.

Alvarez-Hamelin, J. I., Dall'Asta, L., Barrat, A., and Vespignani, A. k-core decomposition: A tool for the visualization of large scale networks. *arXiv preprint cs/0504107*, 2005.

Arora, S., Hazan, E., and Kale, S. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012.

Bahmani, B., Kumar, R., and Vassilvitskii, S. Densest subgraph in streaming and mapreduce. *Proceedings of the VLDB Endowment*, 5(5):454–465, 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.

Bateni, M., Bhaskara, A., Lattanzi, S., and Mirrokni, V. S. Distributed balanced clustering via mapping coresets. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pp. 2591–2599, 2014.

Beame, P., Koutris, P., and Suciu, D. Communication steps for parallel query processing. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI symposium on Principles of database systems*, pp. 273–284. ACM, 2013.

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. ACM, 2015.

Bhawalkar, K., Kleinberg, J., Lewi, K., Roughgarden, T., and Sharma, A. Preventing unraveling in social networks: the anchored k-core problem. *SIAM Journal on Discrete Mathematics*, 29(3):1452–1475, 2015.

Charikar, M. Greedy approximation algorithms for finding dense components in a graph. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*, pp. 84–95. Springer, 2000.

Chen, J. and Saad, Y. Dense subgraph extraction with application to community detection. *IEEE Transactions on Knowledge and Data Engineering*, 24(7):1216–1230, 2012.

Chen, T., Jiang, S., Chu, L., and Huang, Q. Detection and location of near-duplicate video sub-clips by finding dense subgraphs. In *Proceedings of the 19th ACM international conference on Multimedia*, pp. 1173–1176. ACM, 2011.

Czumaj, A., Łącki, J., Mądry, A., Mitrović, S., Onak, K., and Sankowski, P. Round compression for parallel matching algorithms. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pp. 471–484. ACM, 2018.

Dean, J. and Ghemawat, S. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

Ene, A., Im, S., and Moseley, B. Fast clustering using mapreduce. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011*, pp. 681–689, 2011. doi: 10.1145/2020408.2020515.

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, WWW 2015, Florence, Italy, May 18-22, 2015*, pp. 300–310, 2015. doi: 10.1145/2736277.2741638.

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

Esfandiari, H., Lattanzi, S., and Mirrokni, V. Parallel and streaming algorithms for k-core decomposition. In *International Conference on Machine Learning*, pp. 1396–1405, 2018.

Frank, A. and Gyárfás, A. How to orient the edges of a graph? *Combinatorics*, pp. 353–364, 1978.

Fratkin, E., Naughton, B. T., Brutlag, D. L., and Batzoglou, S. Motifcut: regulatory motifs finding with maximum density subgraphs. *Bioinformatics*, 22(14):e150–e157, 2006.

Ghaffari, M. Distributed mis via all-to-all communication. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pp. 141–149. ACM, 2017.

Ghaffari, M. and Uitto, J. 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*, pp. 1636–1653. SIAM, 2019.

Gibson, D., Kumar, R., and Tomkins, A. Discovering large dense subgraphs in massive graphs. In *Proceedings of the 31st international conference on Very large data bases*, pp. 721–732. VLDB Endowment, 2005.

Gionis, A. and Tsourakakis, C. E. Dense subgraph discovery: Kdd 2015 tutorial. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 2313–2314. ACM, 2015.

Goldberg, A. V. Finding a maximum density subgraph. Technical report, Berkeley, CA, USA, 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.

Isard, M., Budiu, M., Yu, Y., Birrell, A., and Fetterly, D. Dryad: distributed data-parallel programs from sequential building blocks. In *ACM SIGOPS operating systems review*, volume 41, pp. 59–72. ACM, 2007.

Jethava, V., Martinsson, A., Bhattacharyya, C., and Dubhashi, D. P. "the lovasz θ function, svms and finding large dense subgraphs". In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pp. 1169–1177, 2012.

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.

Lee, V. E., Ruan, N., Jin, R., and Aggarwal, C. A survey of algorithms for dense subgraph discovery. *In Managing and Mining Graph Data*, pp. 303–336, 2010a.

Lee, V. E., Ruan, N., Jin, R., and Aggarwal, C. A survey of algorithms for dense subgraph discovery. In *Managing and Mining Graph Data*, pp. 303–336. Springer, 2010b.

Lenzen, C. and Wattenhofer, R. 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*, pp. 295–296. ACM, 2010.

Leskovec, J., Lang, K. J., Dasgupta, A., and Mahoney, M. W. Statistical properties of community structure in large social and information networks. In *Proceedings of the 17th international conference on World Wide Web*, pp. 695–704. ACM, 2008.

Liu, H. and Yan, S. Robust graph mode seeking by graph shift. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 671–678. Citeseer, 2010.

Miller, B. A., Bliss, N. T., and Wolfe, P. J. Subgraph detection using eigenvector L1 norms. In *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010. Proceedings of a meeting held 6-9 December 2010, Vancouver, British Columbia, Canada.*, pp. 1633–1641, 2010.

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. ACM, 2015.

Papailiopoulos, D. S., Mitliagkas, I., Dimakis, A. G., and Caramanis, C. Finding dense subgraphs via low-rank bilinear optimization. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pp. 1890–1898, 2014.

Saha, B., Hoch, A., Khuller, S., Raschid, L., and Zhang, X.-N. Dense subgraphs with restrictions and applications to gene annotation graphs. In *Annual International Conference on Research in Computational Molecular Biology*, pp. 456–472. Springer, 2010.

White, T. *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.

Yang, J. and Leskovec, J. Defining and evaluating network communities based on ground-truth. *Knowl. Inf. Syst.*, 42(1):181–213, 2015. doi: 10.1007/s10115-013-0693-z. URL https://doi.org/10.1007/s10115-013-0693-z.

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.