Parallel and Streaming Algorithms for K-Core Decomposition

Hossein Esfandiari [1] **Silvio Lattanzi** [1] **Vahab Mirrokni** [1]

Abstract

The k-core decomposition is a fundamental primitive in many machine learning and data mining applications. We present the first distributed and the first streaming algorithms to compute and maintain an approximate k-core decomposition with provable guarantees. Our algorithms achieve rigorous bounds on space complexity while bounding the number of passes or number of rounds of computation. We do so by presenting a new powerful sketching technique for k-core decomposition, and then by showing it can be computed efficiently in both streaming and MapReduce models. Finally, we confirm the effectiveness of our sketching technique empirically on a number of publicly available graphs.

1. Introduction

A wide range of data mining, machine learning and social network analysis problems can be modeled as graph mining tasks on large graphs. The ability to analyze layers of connectivity is useful to understand the hierarchical structure of the input data and the role of nodes in different networks. A commonly used technique for this task is the k-core decomposition: a k-core of a graph is a maximal subgraph where every node has induced degree at least k. k-core decomposition has many real world applications from understanding dynamics in social networks (Bhawalkar et al., 2012) to graph visualization (Alvarez-Hamelin et al., 2005), from describing protein functions based on protein-protein networks (Altaf-Ul-Amin et al., 2006) to computing network centrality measures (Healy et al., 2006). k-core is also widely used as a sub-routine for community detection algorithms (Chester et al., 2012; Mitzenmacher et al., 2015) or for finding dense clusters in graphs (Lee et al., 2010; Mitzenmacher et al., 2015). As a graph theoretic tool k-core decomposition has been used to solve the densest subgraph problem (Lee et al., 2010; Bahmani et al., 2012; Epasto et al., 2015; Esfandiari et al., 2015).

k-core is often use a feature in machine learning systems with applications in network analysis, spam detection and biology. Furthermore, in comparison with other density-based measure as the densest subgraph, it has the advantage to assign a score to every node in the network. Finally, the k-core decomposition induce a hierarchical clustering on the entire network. For many applications in machine learning and in data mining, it is important to be able to compute it efficiently on large graphs.

In the past decade, with increasing size of data sets available in various applications, the need for developing scalable algorithms has become more important. By definition, the process of computing k-core decomposition is sequential: in order to find the k-core, one can keep removing all nodes of degree less than k from the remaining graph until there is no such a node. As a result, computing k-cores for big graphs in distributed systems is a challenging task. In fact, while k-core decomposition has been studied extensively in the literature and many efficient decentralized and streaming heuristics have been developed for this problem (Montresor et al., 2013; Sarayuce et al., 2015), nevertheless developing a distributed or a streaming algorithm with provable guarantees for k-core decomposition problem remains an unsolved problem. One difficulty in tackling the problem is that simple non-adaptive sampling techniques used for similar problems as densest subgraph (Lee et al., 2010; Esfandiari et al., 2015; Bahmani et al., 2012; Epasto et al., 2015; Bhattacharya et al., 2016) do not work here (See Related Work for details). In this paper, we tackle this problem and present the first parallel and streaming algorithm for this problem with provable approximation guarantee. We do so by defining an approximate notion of k-core, and providing an adaptive space-efficient sketching technique that can be used to compute an approximate k-core decomposition efficiently. Roughly speaking, a $1-\epsilon$-approximate k-core is an induced subgraph that includes the k-core, and such that the induced degree of every node is at least $(1-\epsilon)k$.

Our Contributions. As a foundation to all our results, we provide a powerful sketching technique to compute a $1-\epsilon$-approximate k-core for all k simultaneously. Our sketch is adaptive in nature and it is based on a novel iterative

[1]Google Research. Correspondence to: Hossein Esfandiari <esfandiari@googol.com>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

edge sampling strategy. In particular, we design a sketch of size $\tilde{O}(n)$ that can be constructed in $O(\log n)$ rounds of sampling[1].

We then show the first application of our sketching technique in designing a parallel algorithm for computing the k-core decomposition. More precisely, we present a MapReduce-based algorithm to compute a $1-\epsilon$ approximate k-core decomposition of a graph in $O(\log n)$ rounds of computations, where the load of each machine is $\tilde{O}(n)$, for any $\epsilon \in (0, 1]$.

Moreover, we show that one can implement our sketch for k-core decomposition in a streaming setting in *one pass* using only $\tilde{O}(n)$ space. In particular, we present a one-pass streaming algorithm for $1-\epsilon$-approximate k-core decomposition of graphs with $\tilde{O}(n)$ space.

Finally, we show experimentally the efficiency and accuracy of our sketching algorithm on few real world networks.

Related Work. The k-core decomposition problem is related to the densest subgraph problem. Streaming and turnstile algorithms for the densest subgrpah problem have been studied extensively in the past (Lee et al., 2010; Esfandiari et al., 2015; Bahmani et al., 2012; Epasto et al., 2015; Bhattacharya et al., 2016). While these problems are related, the theoretical results known for the densest subgraph problem are not directly applicable to the k-core decomposition problem.

There are two types of algorithms for the densest subgraph problem in the streaming. First type of algorithms simulates the process of iteratively removing vertices with small degrees (Bahmani et al., 2012; Epasto et al., 2015; Bhattacharya et al., 2016). All of these results are based on the fact that we only need logarithmic rounds of probing to find a $1/2$ approximation of the densest subgraph (Bahmani et al., 2012). However, this can not be used to $1-\epsilon$ approximate the k-coreness numbers.

The second type of algorithms do a (non-adaptive) single pass and use uniform samplings of edges (Esfandiari et al., 2015; Mitzenmacher et al., 2015; McGregor et al., 2015). These results are based on the fact that the density of the optimum solution is proportional to the sampling rate with high probability, where the probability of failure is *exponentially* small. There are two obstacles toward applying this approach to approximating a k-core decomposition. First, by using uniform sampling it is not possible to obtain a $(1-\epsilon)$ approximation of the coreness number for nodes of constant degree (unless we do not sample all edges with probability one). Second, in order to achieve $\tilde{O}(n)$ space, we can only sample $\tilde{O}(1)$ edges per vertex. Hence, the probability that the degree of a vertex in the sampled is not proportional to the sampling rate, is not exponentially small anymore. Therefore it is not possible to union bound over exponentially many feasible solutions. To overcome this issue, we analyze the combinatorial correlation between feasible solutions and wisely pick polynomially many feasible solutions that approximate all of the feasible solutions. To the best of our knowledge this is the first work that analyzes the combinatorial correlation of different feasible solution on a graph.

In recent years the k-core decomposition problem received a lot of attention (Bhawalkar et al., 2012; Montresor et al., 2013; Aksu et al., 2014; Sarayuce et al., 2015; Zhang et al., 2017), nevertheless we do not know of any previous distributed algorithms with small bounded memory and number of rounds. A recent related paper is (Sarayuce et al., 2015) where the authors present a streaming algorithm for the k-core decomposition problem. While the authors report good empirical results for their algorithm, they do not provide a guarantee for this problem, e.g., they do not prove an upper bound on the memory complexity of this algorithm. Finally we note that Monteresor et al. (Montresor et al., 2013) provide a distributed algorithm for this problem in a vertex-centric model. Although their model is different from our, more classic, MapReduce setting and their bound on the number of rounds is linear instead we achieve a logarithmic bound.

2. Preliminaries

In this section, we introduce the main definitions and the computational models that we consider in the paper. We start by defining k-core and by introducing the concept of approximate k-core. Then we describe the MapReduce and streaming models.

Approximate k-core. Let $G = (V, E)$ be a graph with $|V| = n$ nodes and $|E| = m$ edges. Let H be a subgraph of G, for any node $v \in G$ we denote by $d(v)$ the degree of the node in G and for any node $v \in H$ we denote by $d_H(v)$ the degree of v in the subgraph induced by H. A k-core is a maximal subgraph $H \subseteq G$ such that $\forall v \in H$ we have $d_H(v) \geq k$. Note that for any k the k-core is unique and it may be possibly disconnected. We say that a vertex v has *coreness* number k if it belongs to the k-core but it does not belong to the $(k+1)$-core. We denote the coreness number of node i in the graph G with $C_G(i)$(we drop the subscript notation when the graph is clear from the context).

We define the *core labeling* for a graph G as the labeling where every vertex v is labeled with its *coreness* number. It is wroth noting that this labeling is unique and that it defines a hierarchical decomposition of G.

In this paper we are interested in computing a good approxi-

[1] In the paper, we use the notation $\tilde{O}(\cdot)$ to denote the fact that poly-logarithmic factors are ignored.

mation of the *core labeling* for a graph G efficiently in the MapReduce and in the streaming model. For this reason, we introduce the concept of $1-\epsilon$ approximate k-core. We define a $1-\epsilon$ approximation to the k-core of G to be a subgraph H of G that contains the k-core of G and such that $\forall v \in H$ we have $d_H(v) \geq (1-\epsilon)k$. In other words, a $1-\epsilon$ approximation to the k-core of G is a subgraph of the $(1-\epsilon)k$-core of G and supergraph of the k-core of G. In Figure 1 we present the 3-core for a small graph and a $\frac{2}{3}$-approximate 3-core.

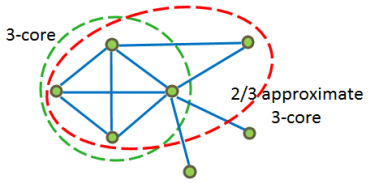


Figure 1. Example of 3-core and $\frac{2}{3}$-approximate 3-core.

Similarly, a $1-\epsilon$ approximate core-labeling of a graph G is a labeling of the vertices in G, where each vertex is labelled with a number between its coreness number and its coreness number multiplied by $\frac{1}{1-\epsilon}$.

In the paper we often refer to the classic greedy algorithm (Matula & Beck, 1983)(also known as peeling algorithm) to compute the coreness number. The algorithm works as follows: nodes are removed from the graph iteratively. In particular, in iteration i of the algorithm all nodes with degree smaller or equal to i are removed iteratively and they are assigned coreness number i. It is possible to show that the algorithm computes the correct coreness number of all nodes in the graph and it can be implemented in linear time.

MapReduce model. Here we briefly recall the main aspect of the model by Karloff et al. (Karloff et al., 2010) of the MapReduce framework (Dean & Ghemawat, 2010).

In the MapReduce model, the computation happens in parallel in several rounds. In each round, data is analyzed on each machine in parallel and then the output of the computations are shuffled between machines. The model has two main restrictions, one on the total number of machines and another on the memory available on each machine. More specifically, given an input of size N, and a small constant $\epsilon > 0$, in the model there are $N^{1-\epsilon}$ machines, each with $N^{1-\epsilon}$ memory available. Note that, the total amount of memory available to the entire system is $O(N^{2-2\epsilon})$.

The efficiency of an algorithm is measured by the number of the "rounds" needed by the algorithm to terminate. Classes of algorithms of particular interest are the ones that run in a constant or poly-logarithmic number of rounds.

Streaming We also analyze the approximate core labelling problem in the streaming model (Munro & Paterson, 1980). In this model the input consists of an undirected graph $G = (V, E)$ and the input is presented as a stream of edges. The goal of our algorithm is to obtain a good approximation of the core labelling at the end of the stream using only small memory ($\tilde{O}(n)$).

3. Sketching k-Cores

In this section we present a sketch to compute an approximate core labelling that uses only $O(n \operatorname{polylog}(n))$ space. Compared with previous sketching for similar problems (Lee et al., 2010; Esfandiari et al., 2015; Bahmani et al., 2012; Epasto et al., 2015; Bhattacharya et al., 2016) our sketching samples different area of the graphs with different, carefully selected, probabilities.

The main idea behind the sketch is to sample edges more aggressively in denser areas of the graph and less aggressively in sparser areas. More specifically, the algorithm works as follows: we start by sampling edges with some small probability, p, so that the resulting sampled graph, H, is sparse. We then compute the coreness numbers for the vertices in H. The key observation is that if a vertex has logarithmic coreness number in H we can precisely estimate its coreness number in the input graph G. Furthermore we can show that if a vertex has large enough coreness number in the input graph G it will have at least logarithmic coreness number in H. So using this technique we can detect efficiently all nodes with sufficiently high coreness number. To compute the coreness numbers of the rest of the node in the graph, we first remove from the graph the nodes for which we have a good estimation and then we iterate the same approach. In particular we double our sample probability p and sample edges again. Interestingly, we can show that by sampling edges adaptively, we can iteratively estimate the coreness of all nodes in the graph by analyzing only sparse subgraphs.

We are now ready to describe our sketching algorithm in details. We start by describing a basic subroutine that estimates a modified version of the coreness number. We dubbed the subroutine *ExclusiveCorenessLabeling*. The subroutine takes as input a subgraph, H, and a subset of the vertices $\Lambda \subseteq H$ and it runs a modified version of the classic peeling algorithm (Matula & Beck, 1983) to compute the coreness number. The main difference between *ExclusiveCorenessLabeling* and the peeling algorithm in (Matula & Beck, 1983) is that we do not compute labels for nodes in Λ and we do not remove them from the subgraph H. The pseudocode for *ExclusiveCorenessLabeling* is presented in Algorithm 1.

During the execution of the algorithm we use subroutine *ExclusiveCorenessLabeling* to compute a labelling for the

Algorithm 1 $ExclusiveCorenessLabeling(H, \Lambda)$

Input: A graph H with n vertices and a set $\Lambda \subseteq V_H$.
Initialize $\Gamma = V_H \setminus \Lambda$
Initialize $l \leftarrow 0$
while $\Gamma \neq \emptyset$ **do**
  while $\min_{v \in \Gamma}(d_H(v)) \leq l$ **do**
    Let $v \leftarrow argmin_{v \in \Gamma}(d_H(v))$
    Set $l_v \leftarrow l$
    Remove v from Γ
    Remove v from H
  end while$l \leftarrow l + 1$
end while

subset of the nodes in H for which we do not have already a good estimate of the coreness number.

Now we can formally present our algorithm, we start by sampling the graph G with $p \in O\left(\frac{\log n}{\epsilon^2 n}\right)$. In this way, we obtain a sparse graph H_0. Then we run *ExclusiveCoreness-Labeling* with $H = H_0$ and $\Lambda = \emptyset$ to obtain a labeling of the nodes in H_0. Let $l_0(i)$ be the label of vertex i in this labeling. If a vertex i has $l_0(i) \geq C \log n$, for a specific constant $C > 0$, we can estimate its coreness number in G precisely. Intuitively this is true because we are sampling the edges independently so we can use concentration results to bound its coreness number. Hence, in the first round of our algorithm we can compute a precise estimate of the coreness number for all nodes i with $l_0(i) \geq C \log n$.

In the rest of the execution of our algorithm we can recurse on the remaining nodes. To do so, we add the nodes with a good estimate to the set Λ and we remove from G the edges in the subgraph induced by the nodes in Λ. Then we increase the sampling probability p by 2 and sample G again. Similarly we obtain a new subgraph H_1 and we run *ExclusiveCorenessLabeling* with $H = H_1$ and Λ equal to the current Λ. So we obtain a labeling l_1 for the nodes in $H_1 \setminus \Lambda$. and also in this case if a vertex i has $l_1(i) \geq C \log n$, for a specific constant $C > 0$, we can estimate its coreness number in G precisely.

We iterate this algorithm for $\log n$ steps. In the remaining of the section we first present pseudocode of our sketching algorithm(Algorithm 2) then we show that at the end of the execution of the algorithm we have a good estimation of the coreness number for all nodes in G. Finally we argue that in every iteration the graphs H_i are sparse so the algorithm uses only small memory at any point in time.

We start by providing the pseudocode of the algorithm in Algorithm 2.

We are now ready to prove the main properties of our sketching technique. We start by stating two technical lemma whose proofs follow from application of concentration bounds and are deferred to the full version. The main goal of the lemma is to relate the degree of a vertex v in a

Algorithm 2 A sketch based algorithm to compute $1 - O(\epsilon)$ approximate core-labeling.

Input: A graph G with n vertices and parameter $\epsilon \in (0, 1]$.
Initialize $\Lambda \leftarrow \emptyset$
Initialize $p_0 \leftarrow \frac{12 \log n}{\epsilon^2 n}$
for $j = 0$ to $\log n$ **do**
  Let H_j be a subgraph of G with the edges sampled independently with probability p_j
  Run $Exclusive\_Core\_Labeling(H_j, \Lambda)$ and denote the label of vertex i on H_j by $l_j(i)$
  for $i \in H_j$ **do**
    if $l_j(i) \geq \frac{24 \log n}{\epsilon^2} \vee p_j = 1$ **then**
      // Node i has sufficiently high degree to estimate its coreness number.
      if $l_j(i) \leq \frac{48 \log n}{\epsilon^2}$ **then**
        Set the label of vertex i to $(1 - \epsilon)\frac{l_j(i)}{p_j}$
        Add i to Λ
      else
        Set the label of vertex i to $\frac{2(1-\epsilon)n}{2^{j-1}}$
        Add i to Λ
      end if
    end if
  end for
  Remove from G the edges of G induced by Λ
  $p_{j+1} \leftarrow 2p_j$
end for

subgraph of G and the sampled subgraph H.

Lemma 3.1. *Let G be a graph and let $\epsilon \in (0, 1]$ and $\delta \in (0, 1)$ be two arbitrary numbers. Let $f(n)$ be a function of n such that $f(n) \geq \frac{6 \log \frac{n}{\delta}}{\epsilon^2}$ and let H be a subgraph of G that contains each edge of G independently with probability $p \geq \frac{6 \log \frac{n}{\delta}}{\epsilon^2 f(n)}$. Then for all $v \in G$ the following statements holds, with probability $1 - \frac{\delta}{3n^2}$: (i) If $d_G(v) \geq f(n)$ we have $|d_H(v) - pd_G(v)| \leq \epsilon d_H(v)$, (ii) If $d_G(v) < f(n)$ we have $d_H(v) < 2pf(n)$.*

Lemma 3.2. *Let G be a graph and let $\epsilon \in (0, 1]$ and $\delta \in (0, 1)$ be two arbitrary numbers. Let $f(n)$ be a function of n such that $f(n) \geq \frac{6 \log \frac{n}{\delta}}{\epsilon^2}$ and let H be a subgraph of G that contains each edge of G independently with probability $p \geq \frac{6 \log \frac{n}{\delta}}{\epsilon^2 f(n)}$. For all $v \in G$ the following statements holds, with probability $1 - \frac{\delta}{3n^2}$: (i) If $d_H(v) \geq 2pf(n)$ we have $|d_H(v) - pd_G(v)| \leq \epsilon d_H(v)$, (ii) If $d_H(v) < 2pf(n)$ we have $d_G(v) \leq 2(1 + \epsilon)f(n)$. In addition, in the first case we have $d_G(v) \geq 2(1 - \epsilon)f(n)$. Furthermore, if the graph is directed the same claims hold for the in-degree($d^-(v)$) and the out-degree($d^+(v)$) of a node v.*

In the remaining of this section we assume that Lemma 3.1 and Lemma 3.2 hold and using this assumption we prove the main properties of our algorithm.

We start by comparing the labels computed by Algorithm 1 with the coreness number of its input graph. Recall that we denote the coreness number of node i in the graph G with $C_G(i)$.

Lemma 3.3. *Let $H = (V, E)$ be an arbitrary graph and let $\Lambda \subseteq V$ be an arbitrary set of vertices. Let $\hat{H} = (\Lambda, \hat{E})$ be an arbitrary graph on the set of vertices Λ, and let $H' = H \cup \hat{H}$. Let l_v be the label computed by $ExclusiveCorenessLabeling(H, \Lambda)$. Then for each vertex $v \in V \setminus \Lambda$ we have: (i) $l_v \geq C_{H'}(v)$, (ii) if $C_{H'}(v) \leq \min_{u \in \Lambda} \left(C_{H'}(u)\right)$, we have $l_v = C_{H'}(v)$.*

Proof. By definition of coreness number, if we iteratively remove all vertices with degree $C_{H'}(v) - 1$ from H', vertex v is not removed from the graph. Furthermore note that Algorithm 1 does not remove any vertex with degree more than $C_{H'}(v) - 1$ unless $C_{H'}(v) - 1 < l$. Thus, we have $l_v \geq C_{H'}(v)$ as desired.

Note that, if we set $\Lambda = \emptyset$, Algorithm 1 acts as the greedy algorithm that computes the coreness numbers. Moreover notice that if $C_{H'}(v) \leq \min_{u \in \Lambda} \left(C_{H'}(u)\right)$, the classic peeling algorithm does not removed any of the vertices in Λ until it does not consider nodes with degree smaller or equal than l. Therefore we have $l_v \geq C_{H'}(v)$ which proves the second statement of the theorem. □

We are now ready to state the two main Lemma proving the quality of the solution computed by our sketching technique.

Lemma 3.4. *For all $0 \leq j \leq \log n$ such that $p_j \leq 1$ and for any node v added to Λ in round j we have with probability $1 - \frac{1}{3n}$ that: $C(v) < 2(1+\epsilon)\frac{n}{2^{j-1}}$.*

Furthermore for all $0 \leq j \leq \log n$ such that $p_j < 1$ we have with probability $1 - \frac{1}{3n}$ that: $C(v) \geq 2(1-\epsilon)\frac{n}{2^j}$.

Lemma 3.5. *Algorithm 2 computes a $1 - 2\epsilon$ approximate core labeling, with probability $1 - \frac{2}{3n}$.*

The proofs of the Lemma is presented in the full version.

Now we give a lemma that bounds the total number of edges used in sketches $H_0, H_2, \ldots, H_\rho$.

Lemma 3.6. *The number of edges in $\cup_{i=0}^{\rho} H_i$ produced by Algorithm 2 is upper bounded by $O\left(\frac{n \log^2 n}{\epsilon^2}\right)$, with probability $1 - \frac{1}{n}$.*

Proof. In the proof, we assume that the statement of Lemma 3.4 holds, and the statements of Lemma 3.1 and 3.2 hold for $H_{j,k}$ and $H_{j,v}$ for all choices of j and k and v.

Consider an arbitrary $0 \leq j \leq \log n$. From Lemma 3.4, we have that for any $v \in H_j \setminus (\cup_{i=0}^{j-1} \Lambda_j)$ the coreness number of v is bounded by $2(1+\epsilon)\frac{n}{2^{j-1}}$. Now consider an orientation of the edges of H_j where every edge is oriented to its endpoint of smallest core number, breaking the ties in such a way that the in-degree of every node v, $d^-(v)$ is upperbounded by $C(v)^2$. Furthermore note that every edge in H_j is incident to a node of coreness number at most $2(1+\epsilon)\frac{n}{2^{j-1}}$, so using Lemma 3.1 we have that in-degree of every node in H_j is bounded by $2(1+\epsilon)^2 \frac{n}{2^{j-1}} p^j = 48\frac{(1+\epsilon)^2}{\epsilon^2} \log n$. So summing over all the in-degrees we get that the number of edges in H_j is bounded by $48\frac{(1+\epsilon)^2}{\epsilon^2} n \log n$. We conclude the proof by noticing that there are at most $\log n$ different H_j so the total memory used is $48\frac{(1+\epsilon)^2}{\epsilon^2} n \log^2 n$. □

Putting together Lemma 3.5 and Lemma 3.6 we get the main theorem of this section.

Theorem 3.7. *Algorithm 2 computes a $1 - 2\epsilon$ approximate core labeling and the total spaced used by the algorithm is $O\left(\frac{n \log^2 n}{\epsilon^2}\right)$, with probability $1 - \frac{2}{n}$.*

4. MapReduce and Streaming Algorithms

In this section we show how to compute our sketch efficiently using a MapReduce or a streaming algorithm.

4.1. MapReduce algorithm

Here, we show how to use implement the sketch introduced in Section 3 in the MapReduce model. In this way we obtain an efficient MapReduce algorithm for dense graphs[3].

Recall that the main limitation of the MapReduce model is on the number of machines and on the available memory on each machine. Our algorithm runs for $2 \log n$ rounds.[4] In the first round of MapReduce, the edges are sampled in parallel with probability $p_0 = \frac{12 \log n}{\epsilon^2 n}$. In this way, we obtain a graph H_0 that we analyze in the second round in a single machines(note that we can do it because from Lemma 3.6 we know that for all i the number of edges in H_i is bounded by $O\left(\frac{n \log^2 n}{\epsilon^2}\right)$. At the end of the second round, we obtain the labeling for the nodes with high coreness number and we add them to the set Λ_0. In the third round we send the set Λ_0 to all the machined and we sample in parallel the edges in $|E| \setminus \Lambda_0$ with probability $2p_0$ in a round of MapReduce. In this way, we obtain H_1 that in the fourth round is analyzed by a single machine to obtain the labelling of few additional nodes that are added to Λ_1. By iterating this process for $2 \log n$ rounds, we obtain an approximation of the coreness number for each node. The pseudo-code for the MapReduce algorithm is presented in Algorithm 3.

[2]Note that such an orientation exists, in fact it can be obtained by orienting every edges to its endpoint that is first removed by the classic peeling algorithm used to compute the coreness number.

[3]It is important to note that we only use polylogarithmic memory for each machine so our algorithm works also in more restrictive parallel models as the massively parallel model (Andoni et al., 2014; Im et al., 2017)

[4]The algorithm can be implemented using $\log n$ MapReduce rounds, but for simplicity, here we present a $2 \log n$ rounds version.

Algorithm 3 A MapReduce algorithm to compute $1 - O(\epsilon)$-approximate core-labeling.

Input: A graph G with n vertices and parameter $\epsilon \in (0, 1]$.
Initialize $\Lambda \leftarrow \emptyset$
Initialize $p_0 \leftarrow \frac{12 \log n}{\epsilon^2 n}$
for $j = 0$ to $\log n$ **do**
  // First round of MapReduce
  Send Λ to all machines
  Let E' be the set of edges of G that are not contained in the graph induced by Λ on G
  Sample with probability p_j in parallel using n machines the edges in E'
  // Second round of MapReduce
  Send all the sampled edge to a single machine
  Let H_j be the sampled subgraph of G
  Run $Exclusive\_Core\_Labeling(H_j, \Lambda)$ and denote the label of vertex i on H_j by $l_j(i)$
  for $i \in H_j$ **do**
    if $l_j(i) \geq \frac{24 \log n}{\epsilon^2} \vee p_j = 1$ **then**
      // Node i has sufficiently high degree to estimate its coreness number.
      if $l_j(i) \leq \frac{48 \log n}{\epsilon^2}$ **then**
        Set the label of vertex i to $(1 - \epsilon)\frac{l_j(i)}{p_j}$
        Add i to Λ
      else
        Set the label of vertex i to $\frac{2(1-\epsilon)n}{2^{j-1}}$
        Add i to Λ
      end if
    end if
  end for
  $p_{j+1} \leftarrow 2p_j$
end for

Algorithm 4 A streaming algorithm to compute $1 - O(\epsilon)$ approximate core-labeling.

Input: Stream of edges and parameter $\epsilon \in (0, 1]$.
Initialize $\Lambda_i \leftarrow \emptyset, \forall i$
Initialize $p_0 \leftarrow \frac{12 \log n}{\epsilon^2 n}$
Insertion of (u, v)
$r \leftarrow$ random number from $[0, 1]$
for $j = 0$ to $\log n$ **do**
  if $v \notin \cup_{i<j}\Lambda_i$ or $u \notin \cup_{i<j}\Lambda_i$ **then**
    if $r \leq p_j$ **then**
      Add (u, v) to H_j
      Run $Exclusive\_Core\_Labeling(H_j, \Lambda_i)$ and denote the label of vertex i on H_j by $l_j(i)$
      for $i \in H_j$ **do**
        if $l_j(i) \geq \frac{24 \log n}{\epsilon^2} \vee p_j = 1$ **then**
          if $l_j(i) \leq \frac{48 \log n}{\epsilon^2}$ **then**
            Set the label of vertex i to $(1 - \epsilon)\frac{l_j(i)}{p_j}$
            Add i to Λ_j
          else
            Set the label of vertex i to $\frac{2(1-\epsilon)n}{2^{j-1}}$
            Add i to Λ_j
          end if
        end if
      end for
      for $j' = j + 1$ to $\log n$ **do**
        Remove from $H_{j'}$ any edge induced by Λ_j
      end for
    end if
  else
    Break
  end if
  $p_{j+1} \leftarrow 2p_j$
end for

By Theorem 3.7 presented in the previous section we obtain the following corollary.

Corollary 4.1. *Let $G = (V, E)$ be a graph such that $|E| \in \Omega(|V|^{1+\gamma})$, for some constant $\gamma > 0$. Then there is an algorithm that computes w.h.p. an approximate core-labeling of the graph in the MapReduce model using $O(\log n)$ rounds of MapReduce.*

4.2. Semi-streaming algorithms

Next we show an application of our sketch in the streaming setting. We consider the setting where edges are only added to the graph. The main idea behind our streaming algorithm is to maintain at any point in time the sketch presented in Section 3, which requires only $\tilde{O}(n)$ space. In the remaining of the section we describe how we can maintain the sketching in streaming.

When an edge is added to G, we check by sampling if it is H_0. In this case in H_0, we recompute the labeling of H_0 and if one of the endpoints of the edge is added to Λ_0, we update the rest of the sketch to reflect this change. Then, if both endpoints of the edge are not contained in Λ_0, we check by sampling if the edge is contained in H_1. Also in this case, if it is in H_1, we recompute the labeling of H_1 and modify the sketch accordingly. We continue this procedure until both endpoints of the edge are contained in Λ. Notice that, by inserting edges $\cup_{i \leq j}\Lambda_i$s may only grow. Hence if at some point both endpoints of an edge (u, v) are in $\cup_{i \leq j}\Lambda_i$, by inserting more edge u and v remain in $\cup_{i \leq j}\Lambda_i$.

The pseudo-code for the streaming algorithm is presented in Algorithm 4(Note that here for simplicity we recompute the core labels after the insertion of an edge (u, v). However, one might recurse over the neighborhood of u and v and update the core labels locally).

By Theorem 3.7 presented in the previous section we obtain the following corollary.

Corollary 4.2. *There exists a streaming algorithm that computes w.h.p. an $(1 - \epsilon)$-approximate core-labeling of the input graph using $O\left(\frac{n \log^2 n}{\epsilon^2}\right)$ space.*

5. Experiments

In this section, we analyze the performances of our sketch in practice. First we describe our datasets. Next we discuss the implementations of our sketch presented in Section 3 and

our MapReduce algorithm. Then we study the scalability and the accuracy of our sketch. In particular, we analyze the trade-off between quality of the approximation and space used by the sketch.

Datasets. We apply our sketch to eight real-world graphs available in the SNAP, Stanford Large Network Dataset Library (Leskovec & Sosič, 2016): Enron (Klimt & Yang, 2004), Epinions (Richardson et al., 2003), Slashdot (Leskovec et al., 2009), Twitter (McAuley & Leskovec, 2012), Amazon (Yang & Leskovec, 2015), Youtube (Yang & Leskovec, 2015), LiveJournal (Yang & Leskovec, 2015), Orkut (Yang & Leskovec, 2015) with respectively 36692, 75879, 82168, 81306, 334863, 1134890, 3997962 and 3072441 nodes and 183831, 508837, 948464, 1768149, 925872, 2987624, 34681189 and 117185083 edges.

Implementation details. In order to have an efficient implementation of our sketch, we modify Algorithm 2 slightly. More specifically, we change line 14 to "if $l_j(i) \geq T \vee p_j = 1$" where T is a parameter of our implementation. Furthermore, we also modify line 22 to "$p_{j+1} \leftarrow M \cdot p_j$, where M is modifiable multiplicative factor (that in Algorithm 2 is fixed to 2). We also slightly modify our MapReduce algorithm to remove iteratively in parallel all nodes with degree smaller than 3 before sending the remaining graph to a single machine.

Metrics. To study the scalability of the algorithm we implement our MapReduce algorithm in distributed setting and we analyze the running time on different graphs by using a fixed number of machine. To evaluate the quality of our sketch, we consider the quality of the approximation and the space used.

For the quality of the approximation, we report the median error and the error at the 60, 70, 80 and 90 percentile of our algorithm. In interest of space, we report the errors only on nodes with coreness number at least 5, because high coreness number are harder to approximate and for almost all the nodes of coreness smaller than 5 we have errors close to 0.

For space we consider the maximum size of any sample graph H_i and the sum of their sizes. Note that the first quantity bounds the memory used by our distributed algorithm or a multi-pass streaming algorithm, and the second one bounds the memory used by a single pass streaming algorithm.

Scalability Results. In Figure 2 we present the results of our scalability experiments(In the experiment we fix $T = 4$ and $M = 2$). On the x axis we order the graphs based on their number of edges, in the y axis we show the relative running time on different graphs. Note that in the Figure the x axis is in logscale and the y axis is in linear scale so the running time of our algorithm grows sublinearly in the number of edges in the graph proving that our algorithm is able to leverage parallelization to obtain good performance.

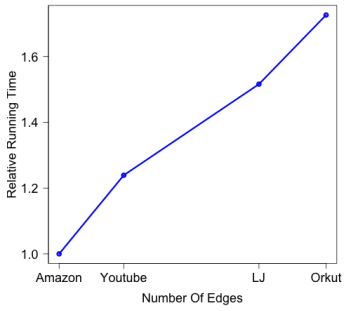


Figure 2. Running time of the distributed algorithm on graphs of increasing size.

For comparison we also run a simple iterative algorithm to estimate the k-core number that resembles a simple adaptation of the algorithm presented in (Lee et al., 2010; Esfandiari et al., 2015; Bahmani et al., 2012; Epasto et al., 2015; Bhattacharya et al., 2016) for densest subgraph. The adapted algorithm works as follows: it removes all nodes below a threshold T(initially equal to 4) from the graphs in parallel and estimate their coreness number as T. Then when no node with degree smaller than T is left, it iteratively increases T by a multiplicative factor $M = 2$ and recurse on the remaining graph. Interestingly we observe that this adapted algorithm is an order of magnitude slower than our distributed algorithm and so we could run it only on relatively small graphs like Amazon.[5]

Accuracy Results. All the reported number are the average over 3 runs of our algorithm. In all our experiment we either fix $T = 3$ and vary M or fix M to 2 and vary T. In Table 1 we present the space used in our algorithm when we vary the value of T.

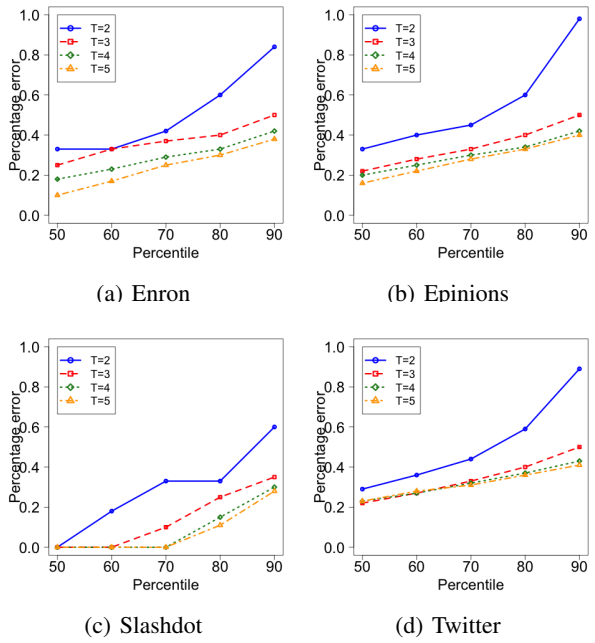
Table 1. Number of edges stored by the sketch as a function of T

Graph	T=2 (max)	T=3 (max)	T=4 (max)	T=5 (max)
Enron	59300	93482	116110	142557
Epinions	80791	120193	147513	169037
Slashdot	132308	203789	258083	302763
Twitter	58967	107164	148610	197321
	T=2 ($\sum$)	T=3 ($\sum$)	T=4 ($\sum$)	T=5 ($\sum$)
Enron	229549	337574	413380	470765
Epinions	322622	436049	515461	575731
Slashdot	537006	799586	975408	1118110
Twitter	299734	501805	682251	848405

[5]Note that the parallel version of the simple iterative algorithm is particularly slow in practice because it needs several parallel rounds to complete.

There are few interesting things to note. First, the size of the maximum sampled graph is always significantly smaller than the size of input graph and in some cases it is more than one order of magnitude smaller (for example in the Twitter case). Interestingly, note that the relative size of the maximum sampled graph decrease with the size of the input graph. This suggests that the sketch would be even more effective when applied to a larger graph. Also the total size of the sketch is also smaller than the size of the graph in many cases (for instance, the sketch for Twitter is always smaller than half of the size of the input graph). This implies that we can compute an approximation of the coreness number without processing most of the edges in the input graph.

In Figure 3, we report the approximation error of our algorithm. First we note that as T increases the approximation error decreases as predicted by our theorems. It is also interesting to note that the median error is always below 50% and with $T \geq 3$ is below 25%. Observe for $T \geq 3$, the error at the 90 percentile is below 50%. Overall our sketch provides a good approximation of the coreness numbers.



(a) Enron (b) Epinions

(c) Slashdot (d) Twitter

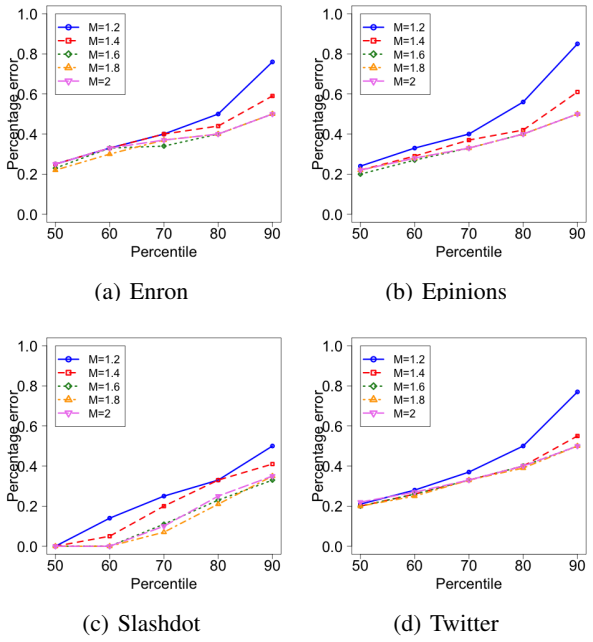
Figure 3. The approximation error of our sketch. We show the error at median 60,70,80 and 90 percentile as a function of T when we restrict our attention to node with coreness number at least 5.

Now we focus on the effect of M on our sketch. In Table 2, we present the space used by our algorithm as a function of M. Note that the maximum size of a single sample graph decrease with M, but the total size of the sketch increases (this is due to the increased number of sampled graphs). This suggests that we should use small M in distributed settings where we have tighter space constraint and larger M when we want to design single pass streaming algorithms.

Table 2. Number of edges stored by the sketch as a function of M

Graph	M=1.2 (max)	M=1.4 (max)	M=1.6 (max)	M=2 (max)
Enron	52202	67611	79075	85013
Epinions	93059	101692	112514	122134
Slashdot	128649	154429	171347	193257
Twitter	51774	67314	81233	92842
	M=1.2 ($\sum$)	M=1.4 ($\sum$)	M=1.6 ($\sum$)	M=2 ($\sum$)
Enron	740240	485841	398151	355671
Epinions	1023529	644102	517660	455226
Slashdot	1759205	1146974	938933	837956
Twitter	903449	650040	561669	521174

Finally it is interesting to note that as shown in Figure 4, the quality of the approximation is not very much influenced by the scaling factor M.



(a) Enron (b) Epinions

(c) Slashdot (d) Twitter

Figure 4. The approximation error of our sketch. We show the error at median 60,70,80 and 90 percentile as a function of M when we restrict our attention to node with coreness number at least 5.

6. Conclusions and future works

In this paper we introduce a new sketching technique for computing the core-labeling of a graph. In particular, we design efficient MapReduce and streaming algorithms to approximate the coreness number of all the nodes in a graph efficiently. We also confirm the effectiveness of our sketch via an empirical study. The most interesting open problem in the area is to design a fully dynamic algorithm (Italiano et al., 1999) to maintain the core-labeling of a graph by using only $\mathrm{polylog}\, n$ operations per update(edge addition or deletion).

References

Aksu, H., Canim, M., Chang, Y., Korpeoglu, I., and Ulusoy, Ö. Distributed k -core view materializationand maintenance for large dynamic graphs. *IEEE Trans. Knowl. Data Eng.*, 26(10):2439–2452, 2014.

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*, 7:207, 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. *CoRR*, abs/cs/0504107, 2005. URL `http://arxiv.org/abs/cs/0504107`.

Andoni, A., Nikolov, A., Onak, K., and Yaroslavtsev, G. Parallel algorithms for geometric graph problems. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pp. 574–583, 2014.

Bahmani, B., Kumar, R., and Vassilvitskii, S. Densest subgraph in streaming and mapreduce. *PVLDB*, 5(5): 454–465, 2012.

Bhattacharya, S., Henzinger, M., and Nanongkai, D. New deterministic approximation algorithms for fully dynamic matching. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pp. 398–411, 2016.

Bhawalkar, K., Kleinberg, J. M., Lewi, K., Roughgarden, T., and Sharma, A. Preventing unraveling in social networks: The anchored k-core problem. In *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II*, pp. 440–451, 2012.

Chester, S., Gaertner, J., Stege, U., and Venkatesh, S. Anonymizing subsets of social networks with degree constrained subgraphs. In *International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2012, Istanbul, Turkey, 26-29 August 2012*, pp. 418–422, 2012.

Dean, J. and Ghemawat, S. Mapreduce: a flexible data processing tool. *Commun. ACM*, 53(1):72–77, 2010.

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.

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

Healy, J., Janssen, J. C. M., Milios, E. E., and Aiello, W. Characterization of graphs using degree cores. In *Algorithms and Models for the Web-Graph, Fourth International Workshop, WAW 2006, Banff, Canada, November 30 - December 1, 2006. Revised Papers*, pp. 137–148, 2006.

Im, S., Moseley, B., and Sun, X. Efficient massively parallel methods for dynamic programming. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pp. 798–811, 2017.

Italiano, G. F., Eppstein, D., and Galil, Z. Dynamic graph algorithms. *Algorithms and Theory of Computation Handbook*, 1999.

Karloff, H. J., Suri, S., and Vassilvitskii, S. A model of computation for mapreduce. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pp. 938–948, 2010.

Klimt, B. and Yang, Y. Introducing the enron corpus. In *CEAS*, 2004.

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. 303336, 2010.

Leskovec, J. and Sosič, R. Snap: A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(1):1, 2016.

Leskovec, J., Lang, K. J., Dasgupta, A., and Mahoney, M. W. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009.

Matula, D. W. and Beck, L. L. Smallest-last ordering and clustering and graph coloring algorithms. *J. ACM*, 30(3): 417–427, 1983.

McAuley, J. J. and Leskovec, J. Learning to discover social circles in ego networks. 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. 548–556, 2012.

McGregor, A., Tench, D., Vorotnikova, S., and Vu, H. T. Densest subgraph in dynamic graph streams. In *International Symposium on Mathematical Foundations of Computer Science*, 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*, KDD '15, pp. 815–824, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3664-2.

Montresor, A., Pellegrini, F. D., and Miorandi, D. Distributed k-core decomposition. *IEEE Trans. Parallel Distrib. Syst.*, 24(2):288–300, 2013.

Munro, J. I. and Paterson, M. Selection and sorting with limited storage. *Theor. Comput. Sci.*, 12:315–323, 1980.

Richardson, M., Agrawal, R., and Domingos, P. M. Trust management for the semantic web. In *The Semantic Web - ISWC 2003, Second International Semantic Web Conference, Sanibel Island, FL, USA, October 20-23, 2003, Proceedings*, pp. 351–368, 2003.

Sarayuce, A. E., Gedik, B., Jacques-Silva, G., Wu, K.-L., and Catalyurek, U. V. Streaming algorithms for k-core decomposition. *PVLDB*, pp. 433–444, 2015.

Yang, J. and Leskovec, J. Defining and evaluating network communities based on ground-truth. *Knowl. Inf. Syst.*, 42 (1):181–213, 2015.

Zhang, Y., Yu, J. X., Zhang, Y., and Qin, L. A fast order-based approach for core maintenance. In *33rd IEEE International Conference on Data Engineering, ICDE 2017, San Diego, CA, USA, April 19-22, 2017*, pp. 337–348, 2017.