A. Relevant Concentration Bounds

Throughout the paper, we will use the following well-known variants of Chernoff bound.

Theorem 11 (Chernoff bound). *Let $X_1, \ldots, X_k$ be independent random variables taking values in $[0,1]$. Let $X \stackrel{\text{def}}{=} \sum_{i=1}^{k} X_i$ and $\mu \stackrel{\text{def}}{=} \mathbb{E}[X]$. Then,*

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

(B) For any $\delta \geq 1$ it holds $\Pr[X \geq (1+\delta)\mu] \leq \exp(-\delta\mu/3)$.

B. Proofs Missing from Section 3

B.1. Proof of Lemma 5

We prove the two properties separately. We assume that $p < 1$, and consequently $50\frac{\log n}{k\varepsilon^2} < 1$, as otherwise H equals G and the lemma follows directly. Observe that this assumption also implies $k > 50\frac{\log n}{\varepsilon^2}$, which will be important for showing that our bounds hold whp.

Proof of A. Let $V_{\geq k}$ be the set of vertices of G that have coreness at least k. Hence, by definition, each $v \in V_{\geq k}$ has at least k neighbors in $V_{\geq k}$. Let $u \in V_H \cap V_{\geq k}$. Therefore, $\mathbb{E}[d_H(v)] \geq pk \geq 50\frac{\log n}{\varepsilon^2}$. Then, by Chernoff bound (see Theorem 11 (A)), with probability at least $1-n^{-4}$ the vertex u has at least $(1-\varepsilon/2)pk$ neighbors in $V_H \cap V_{\geq k}$. Furthermore, by union bound, with probability at least $1-n^{-3}$ this property holds for all the vertices of $V_H \cap V_{\geq k}$ simultaneously. This implies that the vertices of $V_H \cap V_{\geq k}$ are whp labeled by ABOVE by Algorithm 2.

Proof of B. Consider the execution of Algorithm 1 with parameters G and $(1-\varepsilon)k+1$. Let L^i (L_H^i) be the set of vertices of G (of H) labeled by BELOW by the i^{th} iteration of Algorithm 1 (of Algorithm 2). By induction, we will show that $L^i \cap H \subseteq L_H^i$ for each i.

Before we provide the inductive proof, we note the following property. Let v be a vertex of coreness at most $(1-\varepsilon)k$. Let S_v be the set of vertices of G adjacent to v that were not yet labeled when v got labeled by BELOW by Algorithm 1. Observe that S_v is deterministically defined. Also, by definition, we have $|S_v| \leq (1-\varepsilon)k$. In expectation, H contains $p|S_v|$ vertices of S_v. Be Chernoff bound, whp H contains at most $p(|S_v| + \varepsilon k/6) < (1-\varepsilon/2)pk$ vertices of S_v. Furthermore, by union bound, whp this holds for all vertices simultaneously.

Base of induction. Before the first iteration, no vertex is labeled in either of the algorithms. On the other hand, all the vertices of $L^1 \cap H$ whp have less than $(1-\varepsilon/2)k$ neighbors in H. Hence, all such vertices are labeled by BELOW by Algorithm 2 and $L^1 \cap H \subseteq L_H^1$.

Inductive step. Consider iteration i. By our inductive hypothesis, it holds $L^{i-1} \cap H \subseteq L_H^{i-1}$. In other words, all the vertices of H labeled by BELOW by the i^{th} iteration of Algorithm 1 are also labeled by BELOW by the i^{th} iteration of Algorithm 2. (Algorithm 2 might also label by BELOW some additional vertices.) Let $v \in H$ be a vertex labeled by BELOW by Algorithm 1 in the i^{th} iteration. Since $L^{i-1} \cap H \subseteq L_H^{i-1}$, Algorithm 2 decides whether to label v based only on a subset of $S_v \cap H$. As discussed, whp probability it holds $|S_v \cap H| < (1-\varepsilon/2)k$ and hence Algorithm 2 labels v by BELOW as well.

B.2. Proof of Lemma 6

Let p be as defined on Line 1 of Algorithm 3. Fix a vertex v. Let x_v be the number of unlabeled vertices of G incident to v, and y_v the expected degree of v in H_i. We consider two cases: $x_v \geq \frac{10\log n}{p}$, and $x_v < \frac{10\log n}{p}$.

In the first case, $y_v \geq 10\log n$. Therefore, by Chernoff bound (Theorem 11 (A)), it holds $d_{H_i}(v) \leq 2y_v$ with probability at least $1-n^{-3}$.

In the second case, i.e., if $x_v < \frac{10\log n}{p}$, we have $y_v < 10\log n$. Then, $d_{H_i}(v) \leq y_v + 10\log n$ with probability at least $1-n^{-3}$ by Theorem 11 (B) for $\delta = \frac{10\log n}{y_v}$.

Moreover, by union bound, these bounds on $d_{H_i}(v)$ hold for all the vertices simultaneously with probability at least $1-n^{-2}$.

In conclusion, the degree of each vertex v in H_i is strongly concentrated around its expectation y_v, or is far from y_v by an $O(\log n)$ additive term. As discussed above, since the expected number of edges in H_i is $O(n \log^2 n/\varepsilon^4)$, we have that with probability at least $1 - n^{-2}$ the number of edges in H_i is $O(n \log^2 n/\varepsilon^4 + n \log n)$, as desried.

C. Proof Missing from Section 4

C.1. Proof of Lemma 9

From Observation 4, the number of the edges incident to $V^i_{\leq k}$ is at most $k \cdot |V^i_{\leq k}|$. In one iteration, from Lemma 8 it follows that whp Algorithm 4 removes all the vertices with degree less than $(2+\varepsilon')k$. Let S^i ("S" standing for "survived") be the number of non-frozen vertices of coreness at most k that have degree at least $(2+\varepsilon')k$ in the i^{th} iteration. Then, it holds

$$|S^i| \leq \frac{2k \cdot |V^i_{\leq k}|}{(2+\varepsilon')k} \leq (1 - \varepsilon'/4) \cdot |V^i_{\leq k}|. \tag{2}$$

Let F be the vertices of coreness at most k frozen by INITIALIZE of Algorithm 4. By definition, we have

$$|V^{i+1}_{\leq k}| = |S^i| + |F|,$$

and hence from Eq. (2)

$$|V^{i+1}_{\leq k}| \leq (1 - \varepsilon'/4) \cdot |V^i_{\leq k}| + |F|.$$

This inequality together with Lemma 7 implies

$$|V^{i+1}_{\leq k}| \leq (1 - \varepsilon'/4) \cdot |V^i_{\leq k}| + \frac{|V^1_{\leq k}|}{2^{\sqrt{\delta \log n}}}. \tag{3}$$

Now, consider a set $V^i_{\leq k}$. If for some $i < R$ it holds

$$|V^i_{\leq k}| \leq \left(2^{\sqrt{\delta \log n}}\right)^{-\varepsilon'/40} |V^1_{\leq k}|, \tag{4}$$

then Eq. (1) follows from the fact that the sizes of sets $V^j_{\leq k}$ are non-increasing as j grows.

If Eq. (4) does not hold for every $i < R$, then together with Eq. (3) we derive

$$|V^{i+1}_{\leq k}| \leq (1 - \varepsilon'/4) \cdot |V^i_{\leq k}| + \left(2^{\sqrt{\delta \log n}}\right)^{-1+\varepsilon'/40} |V^i_{\leq k}| \leq (1 - \varepsilon'/8) \cdot |V^i_{\leq k}|,$$

for sufficiently large n. The last chain of inequalities implies

$$|V^T_{\leq k}| \leq (1 - \varepsilon'/8)^T |V^1_{\leq k}| \leq \left(2^{\sqrt{\delta \log n}}\right)^{-\varepsilon'/40} |V^1_{\leq k}|,$$

as desired.

C.2. MPC Implementation

In this section we discuss how to implement Algorithm 4 in $O(\log \log n)$ MPC rounds while using $O(n^\delta)$ memory per machine.

INITIALIZE. To detect frozen vertices we only need to compute the degree of each vertex. This can be done in $O(\log_{n^\delta} n) = O(1/\delta)$ many MPC rounds.

PEELINGBELOW-k in $O(\log T)$ many rounds. To execute PEELINGBELOW-k, and hence Algorithm 4, in $O(\log T) \in O(\log \log n)$ MPC rounds we proceed as follows. First, each vertex collects all the relevant information it needs for executing PEELINGBELOW-k. Then, each vertex v uses the collected information to execute PEELINGBELOW-k with respect to v.

Sample T fresh batches of non-frozen edges. Each batch is selected independently of other batches, and an edge is included in a batch with probability p. Now each non-frozen vertex v propagates its incident samples to all the vertices that in the T iterations of the loop of PEELINGBELOW-k depend on the labeling of v. This propagation is applied as follows. The samples of vertex v are denoted by $B_1(v)$. Then, each vertex v requests samples from each vertex in $B_1(v)$. The sample set that v receives in this way is denoted by $B_2(v)$. More generally, each vertex v requests $B_i(u)$ for all $u \in B_i(v)$, obtaining $B_{i+1}(v)$ this way. Observe that $B_i(v)$ corresponds to all the information that v needs to execute 2^i iteration of the loop of PEELINGBELOW-k. So, after $O(\log T)$ steps each vertex gathers all the required information. This approach is also known as graph exponentiation procedure (Lenzen & Wattenhofer, 2010; Ghaffari, 2017).

It remains to analyze the size of $B_i(v)$ and the number of requests each v receives.

Size of $V_i(v)$. Recall that each vertex of degree more than $2k \cdot 2^{\sqrt{\delta \log n}}$ is frozen and it does not gather samples in this process. It implies that each relevant vertex whp has at most $S \stackrel{\text{def}}{=} T\frac{40 \log n \cdot 2^{\sqrt{\delta \log n}}}{(\varepsilon')^2}$ many incident samples. Hence, for constants δ and ε' and for sufficiently large n, we have

$$|B_{\log i}(v)| \leq S^{2^i} \in O\left(n^{\delta/2^{T-i}}\right).$$

Hence, $|B_{\log T}(v)| \in O\left(n^{\delta}\right)$.

Number of requests for $B_i(v)$. For each $i \leq \log T - 1$, v gets some number of requests asking for $B_i(v)$. However, each vertex asking for $B_i(v)$ is in $B_i(v)$. Therefore, each v gets at most $|B_i(v)|$ requests and to each of them responds by sending $\tilde{O}(|B_i(v)|)$ bits. This implies that, for $i \leq \log T - 1$, each v sends $\tilde{O}(n^{\delta})$ bits of information.

D. Arboricity-Dependent Orientation

In this section we prove the following result

Theorem 12. *For a graph with arboricity λ, there is an MPC algorithm that in $\tilde{O}(\sqrt{\log n})$ rounds, computes an orientation of the edges such that each vertex has outdegree at most $(2+\varepsilon)\lambda$. This algorithm uses n^{δ} memory per machine, for an arbitrary constant $\delta \in (0,1)$, and the total memory of $\tilde{O}(\lambda n)$.*

The proof of this theorem is in large similar to the proof of Theorem 2 presented in Section 4. We first recall similarities, and then discuss new ideas compared to those discussed in Section 4. Note that the main difference between the memory requirement of Theorem 2 and of Theorem 12 is that the latter can be implemented by using the total memory that does not depend on $n^{1+\delta}$.

D.1. Similarity with Theorem 2

We begin by stating a well-known fact, which is in the same spirit as Observation 4.

Observation 13. *A graph on n vertices of arboricity λ contains at most λn edges.*

As in the case of k-core decomposition, we first show how to use Observation 13 to obtain an orientation of the edges such that each vertex has outdegree of at most $(2+\varepsilon)\lambda$. We apply the following process iteratively, until the graph becomes empty:

(1) Let S be the set of all the vertices that currently have degree at most $(2+\varepsilon)\lambda$.

(2) Orient arbitrary each edge whose both endpoints are in S. All the other edges incident to S orient outward from S.

(3) Remove all the vertice in S.

Observe that if the starting graph has arboricity λ, then each graph obtained after removing S has arboricity at most λ. Let H be a graph obtained after performing zero or more iterations of this process. Let $n' \stackrel{\text{def}}{=} |V(H)|$. Then, from Observation 13 it follows that the sum of degrees of the vertices of H is at most $2\lambda n'$. This implies that $|V(H) \setminus S| \leq 2\lambda n'/((2+\varepsilon)\lambda)$, and hence $|S| \geq \varepsilon n'/(2+\varepsilon)$. Therefore, for constant ε, after $O(\log n)$ iterations the graph becomes empty, and moreover, all the edges are oriented as required.

To turn this process into an MPC algorithm that uses sublinear memory per machine we design Algorithm 5, that is very similar to Algorithm 4.

Input : G : a graph
ε : approximation parameter
λ : arboricity
T : number of iterations
Output : Orient some of the edges of G

/* INITIALIZE: */
Freeze all vertices of degree more than $2\lambda \cdot 2^{5T}$.
Mark as frozen each edge with both endpoints frozen.
$\varepsilon' \stackrel{\text{def}}{=} \varepsilon/3$

/* ORIENTING: */
$p \leftarrow \min\left\{1, \frac{10 \log n}{(\varepsilon')^2 \lambda}\right\}$
for T *steps* **do**
  Sample each of the non-frozen edges with probability p. Let G' be the sampled graph.
  Let S be the set of all non-frozen vertices $v \in G'$ for which $d_{G'}(v) < (2 + 2\varepsilon')p\lambda$.
  Orient arbitrary each edge whose both endpoints are in S. All the other edges incident to S orient outward from S.
  Remove S from G.
return the obtained orientation

Algorithm 5: Orienting edges

By following the same steps as in the proof of Lemma 9, it can be shown that the following holds.

Lemma 14. *Let V^i be the set of vertices of G (including frozen ones) that are not yet removed by the i^{th} iteration of* ORIENTING *of Algorithm 5. Then, it holds*

$$|V^T| \leq 2^{-\varepsilon' T/8}|V^1|. \tag{5}$$

D.2. New Approach Compared to Section 4

Lemma 14 enables us to obtain the desired orientation while not depending on $n^{1+\delta}$ in the total-memory complexity. To see how, recall first that Lemma 9 essentially states that after T iterations the number of vertices of coreness at most k (the set $V_{\leq k}$ in its statement) reduces significantly. However, if the set $V_{\leq k}$ is small, the total number of all the vertices reduces only by little after T iterations. On the other hand, Lemma 14 *does* claim that the total number of all the vertices reduces significantly (as a function of T) after T iterations. To turn this property into a proof of Theorem 12, instead of dividing the simulation into phases of equal length, we start with $T = 1$ and gradually increase the phase-size. We begin with small T to ensure that the total memory usage is $\tilde{O}(n) \in \tilde{O}(\lambda n)$. Then, after each phase the total number of vertices gets significantly reduced, enabling us to simulate even more iterations in the next phase.

Formally, we define a sequence of *phase-lengths*, where a phase-length T' refers the number of iterations simulated in a given phase, i.e., T' is an input in Algorithm 5. Fix a phase. Let its length be T' and let n' be the number of vertices at the beginning of the phase. To simulate this phase, we collect a T'-hop neighborhood (while ignoring the frozen vertices) of each of the n' vertices. We refer a reader to Appendix C.2 for an explanation of how these neighborhood are collected. As explained in Appendix C.2, this requires the total memory of

$$O(n' \cdot 2^{cT'^2 + T' \log\log n}), \tag{6}$$

for some absolute constant c. Our goal in the rest is to show that there is a way to set phase-lengths such that: (1) for each phase the total memory requirement for gathering these T'-hop neighborhoods for all the vertices is $\tilde{O}(n)$; and, (2) the total number of phases is $\tilde{O}(\sqrt{\log n})$.

Batches of phases. We split phases into *batches* consisting of consecutive phases. In batch j, all the phases have equal length T_j. We set $T_j = j$. Let n_j be the number of vertices at the beginning of the j-th batch of phases. We execute $(8/(\delta \cdot \varepsilon'))(4c + 2\log\log n)$ phases in the first batch. Note that the total memory requirement for collecting T_1-neighborhoods per phase of the first batch is $\tilde{O}(n)$.

By the end of the first batch, by Lemma 14 the number of remaining vertices is at most $n_2 = n/(2^{(4c+2\log\log n)/\delta})$, that can also be written as $n/(2^{(cT_2^2+T_2\log\log n)/\delta})$. Hence, when the second batch of phases begins, for which the phase-length is $T_2 = 2$, the total memory requirement per phase coming from Eq. (6) is $O(n)$. To "prepare" the number of vertices for the third batch so that the total memory requirement per phase coming from Eq. (6) is again $O(n)$, we execute $(8/(\varepsilon'\delta))(3c + \log\log n)$ phases in the second batch. After that, the number of vertices is at most

$$n_3 \le \frac{n_2}{2^{(3cT_2+T_2\log\log n)/\delta}} = \frac{n}{2^{(cT_2^2+3cT_2+2T_2\log\log n)/\delta}} \le \frac{n}{2^{(c(T_2+1)^2+(T_2+1)\log\log n)/\delta}} = \frac{n}{2^{(cT_3^2+T_3\log\log n)/\delta}}.$$

In general, for each $j > 1$, in batch j we execute $(8/(\delta \cdot \varepsilon'))(3c + \log\log n) \in O(\log\log n)$ phases.

Upper-bound on the number of batches. By definition, batch j simulates at least $(8/(\delta\cdot\varepsilon'))\cdot 3cT_j$ of the $\log n$ iterations of the process described in Appendix D.1. Since $\varepsilon' = \varepsilon/3 \le 1/3$ and $c \ge 1$, we have that each batch simulates at least $72 \cdot T_j/\delta$ iterations. Hence, after at most $\sqrt{\delta\log n}/6$ batches the algorithm simulates all the $\log n$ iterations.

Memory complexity. Since each phase has length at most $\sqrt{\delta\log n}/6$, by following the same arguments as in Appendix C.2, it implies that each phase can be executed with n^δ memory per machine.

Now we upper-bound the total memory requirement. Recall that we set the number of phases per batch in a way that expression Eq. (6) is $\tilde{O}(n)$; in fact, Eq. (6) is $O(n)$ starting from the second batch. Hence, to collect the corresponding T-neighborhood we need $\tilde{O}(n)$ total memory. To perform sampling and store the graph, it suffices to use memory $O(\lambda n)$. Hence, the total memory requirement is $\tilde{O}(\lambda n)$.

Round complexity. Next, by using the method of graph exponentiation, each phase can be executed in $O(\log T_j)$ MPC rounds. Thus, batch j can be executed in $O(\log\log n \cdot \log T_j) \in O(\log^2\log n)$ MPC rounds. Hence, the round complexity of the entire algorithm is $O(\sqrt{\log n}\cdot\log^2\log n)$, as desired.

This completes the analysis.

E. The Missing Details from Section 5

In this section we provide the full details of our result on densest sugraph. In Appendix E.1, we begin by defining the primal and the dual LP of densest subgraph. In Appendix E.2 we discuss how to solve this dual LP by using employing the Multplicative Weights Updates method. In Appendix E.4 we state our main algorithm that runs in $\tilde{O}(\sqrt{\log n})$ MPC rounds. We show that this algorithm simulates the described MWU approach. In Appendix E.5 these results are combined into a proof of Theorem 3.

E.1. LP View

We now recall the definition of the LP $\textsc{Dual}(D)$ from Section 5.1. The LP formulation of the densest subgraph problem (Charikar, 2000) is given as follows

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

The dual LP of the LP above is

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

In the sequel, we focus on solving this dual LP. We reduce this task to the one of solving the following feasibility problem, that is obtained from the dual by fixing the value of z to D

$$\begin{aligned} \alpha_{eu} + \alpha_{ev} &\geq 1 \qquad \forall e = \{u, v\} \in E \\ \sum_{e \text{ incident on } v} \alpha_{ev} &\leq D \qquad \forall v \in V \\ \alpha_{ev} &\geq 0 \qquad \forall e, v \end{aligned}$$

Let DUAL(D) denote this feasibility question.

Interpretation of DUAL(D): The feasibility task stated by DUAL(D) can be seen as the following multi-commodity flow question. Consider a bipartite graph $H = (X \cup Y, E_H)$ in which one partition consists of the edges and the other partition consists of the vertices of G, i.e., $X = E(G)$ and $Y = V(G)$. There is an edge between $e \in X$ and $v \in Y$ iff $e = \{u, v\}$ for some u. Let X be the set of sinks and Y be the set of sources. Is there a way to send D units of flow from each of the sources directly to the sinks (this corresponds to the second type of constraints of DUAL(D)) so that each sink receives at least one unit of flow (this corresponds to the first type of constraints of DUAL(D))? The answer is positive iff DUAL(D) is feasible.

E.2. Multiplicative Weights Update Method

In this section we show how to apply the Multiplicative Weights Update (MWU) method to solve DUAL(D). We start by recalling the MWU approach to deciding feasibility of fractional covering problems, and then discuss how to use it to obtain an approximate solution to DUAL(D).

E.2.1. THE MWU METHOD FOR FEASIBILITY OF COVERING LPS

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 $Ax \geq 0$ for all $x \in P$, does there exist $y \in P$ such that $Ay \geq 1$?

Table 2. Definition of feasibility covering LP.

This feasibility problem can be solved by applying the MWU method. To instantiate this method, it is required to provide access to the following oracle:

ORACLE(w):
Given a vector $w \in \mathbb{R}^r_{\geq 0}$: return a vector x such that $w^T Ax \geq \|w\|_1$; or, if such x does not exist, then report "fail".

It is also said that ORACLE(w) solves a given feasibility task on average, i.e., the constraints are satisfied on average as defined by w. Given an access to such ORACLE, the MWU algorithm proceeds in R iterations as follows.

Let ρ be such that $(Ax)_i \leq \rho$ for any $x \in P$. The value ρ is called *width*. For the i-th constraint of Ax and for each of the R iterations the algorithm maintains weight w_i^t. Initially, $w_i^1 = 1$ for each i. Then, in iteration t the algorithm invokes

ORACLE(w^t). If the oracle reports "fail", the algorithm reports that the covering LP is infeasible. Otherwise, let x^t be the vector returned by the oracle. Then, the algorithms sets weights for the next iteration to be $w_i^{t+1} \stackrel{\text{def}}{=} w_i^t(1-\varepsilon(Ax)_i/\rho)$. This update rule leads to the following.

Observation 15. *For each $t \geq 1$ and each i it holds*

$$w_i^t(1-\varepsilon) \leq w_i^{t+1} \leq w_i^t.$$

The following is a well-known result about MWU.

Theorem 16 ((Arora et al., 2012))**.** *Consider a feasibility covering LP problem given by Table 2. Let ρ be the width of that problem and $\varepsilon > 0$ an approximation parameter. There exists an absolute constant c such that after $R = c \cdot \frac{\rho \log r}{\varepsilon^2}$ iterations the MWU method either correctly reports that the covering problem is infeasible, or it holds that $\bar{x} \stackrel{\text{def}}{=} \frac{\sum_{t=1}^{R} x^t}{R}$ is such that $A\bar{x} \geq (1-\varepsilon)1$.*

E.2.2. APPLYING MWU TO SOLVE DUAL(D)

To solve DUAL(D) by using the MWU method, we let the convex set P be the set of points corresponding to all but the first constraint of DUAL(D), i.e., we let

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

Then, we use MWU to decide whether there exists a point $\alpha \in P$ such that $\alpha_{eu} + \alpha_{ev} \geq 1$ for all $e = \{u, v\} \in E$.

We now discuss how to implement ORACLE(w) for this problem. Recall that w corresponds to the constraints that we are aiming to satisfy. Hence, w is indexed by E. For each vertex v, ORACLE(w) selects an edge $e_v^\star$ such that

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

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

E.2.3. REDUCING THE WIDTH

In Appendix E.2.2 we showed how to use MWU to solve DUAL(D). In that setup α_{ev} can be D, and hence $\alpha_{ev} + \alpha_{eu}$ can be as large as $2D$. This implies that the width of DUAL(D) is $2D$. Since D can be as large as $\Theta(n)$, Theorem 16 implies that for some input graphs it would be needed to run $\Theta\left(n \log n/\varepsilon^2\right)$ iterations of ORACLE$(\cdot)$. Our goal is to reduce this number of iterations to $O\left(\log n/\varepsilon^2\right)$. To that end, we define a new feasibility problem that has the width of $O(1)$. We refer to this problem by DUAL$_2(D)$ and define as

$$\begin{aligned} \alpha_{eu} + \alpha_{ev} &\geq 1 && \forall e = \{u, v\} \in E \\ \sum_{e \text{ incident on } v} \alpha_{ev} &\leq D && \forall v \in V \\ \alpha_{ev} &\leq 2 && \forall e, v \\ \alpha_{ev} &\geq 0 && \forall e, v \end{aligned} \tag{7}$$

DUAL$_2(D)$ was formulated in (Bahmani et al., 2014). Observe that, compared to DUAL(D), DUAL$_2(D)$ contains an additional family of constraints of the form $\alpha_{ev} \leq 2$, for all e, v. As a result, there is an oracle ORACLE$_2(\cdot)$ for DUAL$_2(D)$ whose width is 4. Therefore, to solve DUAL$_2(D)$ it suffices to invoke ORACLE$_2(\cdot)$ for $O\left(\log n/\varepsilon^2\right)$ many times. We now describe ORACLE$_2(\cdot)$, that is very similar to ORACLE$(\cdot)$.

E.2.4. ORACLE FOR DUAL$_2(D)$

Define P_2 as

$$P_2 = \{\alpha \in \mathbb{R}^{E \times V} : \sum_{e \text{ incident on } v} \alpha_{ev} \leq D, \alpha_{ev} \leq 2, \text{ and } \alpha_{ev} \geq 0\}.$$

If $\text{DUAL}_2(D)$ is feasible, the task of $\text{ORACLE}_2(p)$ is to output $\alpha^\star \in P_2$ such that

$$\sum_{e=\{u,v\}} w_e \left(\alpha^\star_{eu} + \alpha^\star_{ev}\right) \geq \|w\|_1. \tag{8}$$

Rewrite Eq. (8) as

$$\sum_{e=\{u,v\}} w_e \left(\alpha^\star_{eu} + \alpha^\star_{ev}\right) = \sum_{v\in V} \sum_{e\in E(v)} w_e \alpha^\star_{ev} \geq \|w\|_1. \tag{9}$$

We implement $\text{ORACLE}_2(w)$ as follows. Consider a vertex v. Let e_i be the edge incident to v that among all the edges incident to v has the i-th largest value in w. Let $t = \lfloor D/2 \rfloor$. Given v, assign the following values to $\alpha^\star$: $\alpha^\star_{e_i v} = 2$ for each $1 \leq i \leq t$; $\alpha^\star_{e_{t+1} v} = D - 2t$; and, $\alpha^\star_{e_i v} = 0$ for all $i > t+1$. Perform this assignment for all the vertices.

Clearly, $\alpha^\star \in P_2$. Also, it is easy to verify that if $\text{DUAL}_2(D)$ is feasible, then the described implementation leads to the output that satisfies Eq. (9).

Lemma 17. *Let $\varepsilon > 0$ be an approximation parameter. If G has a subgraph of density at most D, then there exists an absolute constant c and an algorithm that after $c \cdot \log n/\varepsilon^2$ invocations of ORACLE_2 outputs a vector $\bar{\alpha} \in \mathbb{R}^{E\times V}$ such that the constraint Eq. (7) of $\text{DUAL}_2(D)$ is satisfied $(1-\varepsilon)$ approximately, i.e., $\bar{\alpha}_{eu} + \bar{\alpha}_{ev} \geq 1 - \varepsilon$ for all $e = \{u, v\} \in E$, while the other constraints are satisfied exactly.*

E.3. From Fractional Dual to Integral Primal

Solving $\text{DUAL}_2(D)$ results in a fractional (dual) solution. In (Bahmani et al., 2014) is shown how to obtain a densest subgraph given a fractional solution to $\text{DUAL}_2(D)$ for appropriately chosen D.

Lemma 18. *Let $\varepsilon' \in (0, 1/12)$ be an approximation parameter and H be a graph. Consider an MWU algorithm for finding a $(1-\varepsilon')$-approximation of $\text{DUAL}_2(D)$ corresponding to graph H. Let R be the number of MWU iterations. Let w^t be the vector w used to invoke the corresponding oracle in the t-th iteration of MWU. Then, there exists algorithm ROUNDDENSEST that as the input gets $\{w_t\}_{t\in[R]}$ and has the following properties:*

(A) ROUNDDENSEST *can be executed in $O(1)$ MPC rounds.*

(B) Let $D^\star_H$ be the maximum subgraph density of H. For $i \in \mathbb{N}$, let $\tilde{D} = (1+\varepsilon')^i$ be the smallest value such that $\tilde{D} \geq D^\star_H$. Then, ROUNDDENSEST *outputs $(1 - 7\varepsilon')$ approximate densest subgraph.*

We now briefly recall the implementation of ROUNDDENSEST, and refer the reader to (Bahmani et al., 2014) for full details.

Let w^t and D be the input to ROUNDDENSEST as described in Lemma 18. For each w^t we apply the following process independently.

Let $\tilde{w}^t$ be a scaled version of w^t such that $\tilde{w}^t_e \stackrel{\text{def}}{=} w^t_e / \max_h w^t_h$ for each e. For each edge e such that $\tilde{w}^t_e \leq \varepsilon'/m^2$ set $\tilde{w}^t_e = 0$. Round down each $\tilde{w}^t_e$ to the nearest power of $(1+\varepsilon')$. Note that at this point there are only $O\left(\frac{\log m}{\varepsilon'}\right)$ different values of $\tilde{w}^t_e$.

Define G^t_i to be the subgraph of G containing only the edges such that $\tilde{w}^t_e \geq (1+\varepsilon')^i$. Let H^t_i be the graph obtained from G^t_i by removing all the vertice of G^t_i that have degree at most $\lfloor D/2 \rfloor$.

Among all the graphs H^t_i, output one having the largest density.

Remark: We note that the actual algorithm presented in (Bahmani et al., 2014) does not construct all the subgraphs H^t_i, but chooses first t and i so that H^t_i leads to the guarantee of Lemma 18. To choose such w^t and such i requires additional computation that for the sake of simplicity we avoid here. Hence, as in our application there are only $O(\log n/\varepsilon'^2)$ different vectors w^t and for each of them only $O(\log m/\varepsilon')$ different values of i that we have to consider, we execute the above procedure for all of the H^t_i in parallel.

E.4. The Main Algorithm

Before we state our main algorithm, we prove the following result that essentially allows us to assume that the maximum subgraph density of the graph that we have to process is $O(\frac{\log n}{\varepsilon^2})$.

Lemma 19. *Let $D^\star$ to be the density of a densest subgraph of G. Let $\tilde{D} \geq 20\frac{\log n}{\varepsilon^2}$, and let H be a graph obtained from G by keeping each edge of G with probability $\min\{1, \tilde{D}/D^\star\}$ and independently of other edges. Then, whp the following holds:*

(A) Let S be a densest subgraph of G. Then, the density of S in H is at least $\min\{D^\star, (1-\varepsilon)\tilde{D}\}$.

(B) Let T be a subgraph of G of density at most $(1-2\varepsilon)D^\star$. Then, the density of T in H is less than $(1-\varepsilon)\min\{D^\star, \tilde{D}\}$.

(C) No subgraph of H has density more than $\min\{D^\star, (1+\varepsilon)\tilde{D}\}$.

The proof of Lemma 19 is deferred to Appendix E.6.

Combining Lemma 19 with the results derived in the previous sections lead to Algorithm 6. It is not hard to see that this algorithm can be implemented so to find a $(1+\varepsilon)$-approximate densest subgraph of G in $O\left(\frac{\log n}{\varepsilon^2}\right)$ MPC rounds. Moreover, all the steps but Line 5 of the algorithm can be easily simulated in $O(1)$ MPC rounds. Hence, to obtain the round complexity as stated by Theorem 3, in this section we show how to simulate T iterations of MWU in only $O(\log\log n)$ rounds of MPC when the memory per machine is n^δ, where T is defined as

$$T \stackrel{\text{def}}{=} \frac{\sqrt{\delta \log n}}{5}. \tag{10}$$

To achieve that, we proceed as follows.

Input : G : a graph
$\varepsilon \in (0, 1/70)$: approximation parameter
Output : A $(1+70\varepsilon)$-approximate densest subgraph

1 Let $\mathcal{D} \leftarrow \{(1+\varepsilon)^i : 0 \leq i \leq \log_{1+\varepsilon} n\}$ be the list of density candidates.
2 $D_{\text{sparse}} \leftarrow 20\frac{\log n}{\varepsilon^2}$
3 **foreach** $D \in \mathcal{D}$ **in parallel do**
4 Obtain H_D from G by keeping each edge of G independently with probability $\min\{1, (1+\varepsilon)D_{\text{sparse}}/D\}$.
5 Run MWU on $\textsc{Dual}_2(\min\{D, (1+\varepsilon)^2 D_{\text{sparse}}\})$ defined wrt H_D. Let R be the number of MWU iterations. Let $\{w_D^t\}_{t\in[R]}$ be the vectors used to invoke the MWU oracle.
6 Invoke RoundDensest with $\{w_D^t\}_{t\in[R]}$ (see Lemma 18). Let S_D be the subgraph returned by this invocation.
7 **return** $V(S_D)$ having the largest density in G among all $D \in \mathcal{D}$

Algorithm 6: A parallel algorithm for finding a $(1+70\varepsilon)$-approximate densest subgraph.

We split the execution of MWU into *phases*. Each phase groups consecutive iterations of MWU. At the beginning of each phase we choose a small subset of the edges on which will be executed the corresponding MWU oracle. We will show that there is a choice of the edges so that: the number of the edges chosen by *each* vertex is "small", and this edge-sparsification process affects the outcome of the MWU method by only $(1-O(\varepsilon))$ factor. We next describe one phase (Appendix E.4.1) and after discuss how to simulate a phase in MPC (Appendix E.5.2).

E.4.1. A Phase of MWU

Each phase of MWU consists of T iterations. Fix one such phase. Our goal now it to design a new oracle for this phase, that we call R-$\textsc{Oracle}_2$ ("R" coming from "restricted"), that has a restricted access to the graph but provides almost the same guarantee as $\textsc{Oracle}_2$ (guarantee in the sense of Eq. (9)). This restricted access will enable us to consider only a sparse subgraph of our graph while executing a phase of MWU. We will use this sparsification to design an efficient MPC simulation of one phase. We begin by defining R-$\textsc{Oracle}_2$.

Definition of R-$\textsc{Oracle}_2$ Let $\hat{w}$ be the value of w at the beginning of the phase. For a vertex v, order its incident edges $E(v)$ as $e'_1, e'_2, \ldots$ such that $\hat{w}_{e'_i} \geq \hat{w}_{e'_j}$ for each $i \leq j$. Define

$$d^{\text{R}}_{\max} \stackrel{\text{def}}{=} \frac{D^2 \cdot 2^T}{\varepsilon} + \lceil D \rceil, \tag{11}$$

and let $E^{\mathrm{R}}(v)$ be the first $d^{\mathrm{R}}_{\max}$ of those ordered edges, i.e.,

$$E^{\mathrm{R}}(v) \stackrel{\text{def}}{=} \{e'_i : 1 \le i \le \min\{d(v), d^{\mathrm{R}}_{\max}\}\}. \tag{12}$$

When R-ORACLE$_2(w)$ is invoked, let $e_i \in E^{\mathrm{R}}(v)$ be such that among all the edges from $E^{\mathrm{R}}(v)$ the edge e_i has the i-th largest value in p. Let α^{R} be the output vector. Let $t = \lfloor D/2 \rfloor$. Given v, assign the following values to α^{R}: $\alpha^{\mathrm{R}}_{e_i v} = 2$ for each $1 \le i \le t$; $\alpha^{\mathrm{R}}_{e_{t+1} v} = D - 2t$; and, $\alpha^{\mathrm{R}}_{ev} = 0$ for all the other edges $e \in E(v)$. Perform this assignment for all the vertices. Note that the only difference between ORACLE$_2$ (see Appendix E.2.4 for a definition) and R-ORACLE$_2$ is that ORACLE$_2$ takes into account $E(v)$ while R-ORACLE$_2$ takes into account only $E^{\mathrm{R}}(v)$. Also note that R-ORACLE$_2$ is defined w.r.t. to the current phase.

Guarantee for R-ORACLE$_2$ The main task here is to show that within the same phase ORACLE$_2$ and R-ORACLE$_2$ provide almost the same guarantee in the sense of Eq. (9).

Lemma 20. *Fix a phase of MWU. Given vector w in an iteration of the phase, let α^R be the output of* R-ORACLE$_2(w)$ *and $\alpha^\star$ be the output of* ORACLE$_2(w)$. *If $\alpha^\star$ satisfies Eq. (9), then α^R satisfies*

$$\sum_{v \in V} \sum_{e \in E(v)} w_e \alpha^R_{ev} \ge (1-\varepsilon)\|w\|_1. \tag{13}$$

This lemma together with Lemma 17 implies the approximation guarantee of R-ORACLE$_2$.

Lemma 21. *If G has a subgraph of density at most D, then there exists an algorithm that after $R \in O(\log n/\varepsilon^2)$ invocations of* R-ORACLE$_2$ *outputs a vector $\bar{\alpha}^R \in \mathbb{R}^{E \times V}$ such that the constraint Eq. (7) of* DUAL$_2(D)$ *is satisfied $(1-2\varepsilon)$ approximately, i.e., $\bar{\alpha}^R_{eu} + \bar{\alpha}^R_{ev} \ge 1 - 2\varepsilon$ for all $e = \{u, v\} \in E$, while the other constraints are satisfied exactly. Furthermore, these invocations are split into R/T phases.*

Proof. Let $R = c \cdot \log n/\varepsilon^2$ be the number of invocations of ORACLE$_2$ stated by Lemma 17. Split these invocations into phases, each phase (except the last one) consisting of T consecutive invocations.

As G contains a subgraph of density at most D, in each invocation ORACLE$_2$ outputs a vector that satisfies Eq. (9). Then, by Lemma 20, the output α^{R} of R-ORACLE$_2$ is such that it satisfies Eq. (13). That implies that R-ORACLE$_2$ solves the version of DUAL$_2(D)$ where the constraint Eq. (7) is replaced by $\alpha_{eu} + \alpha_{ev} \ge 1 - \varepsilon$. Hence, by Theorem 16,

$$\bar{\alpha}^{\mathrm{R}}_{eu} + \bar{\alpha}^{\mathrm{R}}_{ev} \ge (1-\varepsilon)^2 \ge 1 - 2\varepsilon,$$

as desired. □

To prove Lemma 20, we will need the following result.

Theorem 22 ((Frank & Gyárfás, 1978))**.** *Let $D^\star$ be the maximum density of a subgraph of G. Then, the edges of G can be oriented so that the maximum indegree of each vertex is $\lceil D^\star \rceil$.*

Proof of Lemma 20. Recall that for each v the invocation of ORACLE$_2(w)$ sets $\alpha^\star$ to be non-zero for at most D of the edges incident to v. Those edges correspond to the ones having largest values in w. However, if $d(v) > d^{\mathrm{R}}_{\max}$, some of those edges might not belong to $E^{\mathrm{R}}(v)$ and hence would not be considered by R-ORACLE$_2(w)$ while defining α^{R}. As a consequence, α^{R} might not satisfy Eq. (9). We will show that, nevertheless, α^{R} violates Eq. (9) only by ε factor, i.e., we will show that Eq. (13) holds.
Note that if $d(v) \le d^{\mathrm{R}}_{\max}$, then $E^{\mathrm{R}}(v)$ contains all the edges incident to v (see Eq. (12)), and hence $\alpha^{\mathrm{R}}_{ev} = \alpha^\star_{ev}$ for every $e \in E(v)$. Therefore, in the rest we focus on the vertices having degree more than $d^{\mathrm{R}}_{\max}$.

Consider a vertex v such that $d(v) > d^{\mathrm{R}}_{\max}$. Let $X_v \stackrel{\text{def}}{=} \sum_{e \in E(v)} w_e \alpha^{\mathrm{R}}_{ev}$ be the contribution of v to the LHS Eq. (13). Let $F \stackrel{\text{def}}{=} 2^T$. Throughout a phase, from Observation 15 the value of w_e increases by at most $(1-\varepsilon)^{-T}$ and decreases by at most $(1-\varepsilon)^T$ compared to its initial value $\hat{w}_e$. Recall that, initially, $E^{\mathrm{R}}(v)$ contained $d^{\mathrm{R}}_{\max}$ edges incident to v with *the largest* values in $\hat{w}$. Therefore, for $\varepsilon \le 1/4$, during any point of the phase we have

$$\forall e' \in E^{\mathrm{R}}(v) \text{ and } \forall e \notin E^{\mathrm{R}}(v) \text{ it holds } w_e \le F w_{e'}. \tag{14}$$

Hence, we have $\sum_{e\in E(v)} w_e\alpha^\star_{ev} \le F\cdot X_v$. So, w.r.t. v the output of ORACLE$_2$ potentially contributes F times more, i.e., $F\cdot X_v$, to the LHS of Eq. (9) than the output of R-ORACLE$_2$. We now show, via a proper amortization, that $F\cdot X_v$ is only a small fraction of what the neighbors of v in $E^{\mathrm{R}}(v)$ contribute to the LHS of Eq. (9).

Amortization scheme Consider any neighbor z of v such that $\{v,z\}\in E^{\mathrm{R}}(v)$. Let e_z be an edge of $E^{\mathrm{R}}(z)$ with the largest value in w. We claim that

$$\forall e\notin E^{\mathrm{R}}(v) \text{ it holds } w_e \le F w_{e_z}. \tag{15}$$

To prove Eq. (15) consider two cases: $\{v,z\}\in E^{\mathrm{R}}(z)$ and $\{v,z\}\notin E^{\mathrm{R}}(z)$.

- **Case $\{v,z\}\in E^{\mathrm{R}}(z)$.** By definition $w_{\{v,z\}}\le w_{e_z}$. Since $\{v,z\}\in E^{\mathrm{R}}(v)$ and $w_e\le F w_{e'}$ for any $e'\in E^{\mathrm{R}}(v)$ and any $e\notin E^{\mathrm{R}}(v)$ (see Eq. (14)), the claim Eq. (15) follows.
- **Case $\{v,z\}\notin E^{\mathrm{R}}(z)$.** As $\{v,z\}$ was not selected to $E^{\mathrm{R}}(z)$, we have $\hat{w}_{e_z}\ge \hat{w}_{\{v,z\}}$. Also $\hat{w}_{\{v,z\}}\ge \hat{w}_e$ for any $e\notin E^{\mathrm{R}}(v)$, and hence $\hat{w}_{e_z}\ge\hat{w}_e$. Since within the phase w_e can increase by at most $(1-\varepsilon)^{-T}$ and w_{e_z} decrease by at most $(1-\varepsilon)^T$, we have $w_{e_z}\ge F w_e$, as desired.

Now, select *any* FD^2/ε neighbors of v in $E^{\mathrm{R}}(v)$. Let that set be A_v. Recall that $\alpha^\star_v$ has at most D non-zeros, and also observe that $\alpha^{\mathrm{R}}_{e_z z}=\|\alpha^{\mathrm{R}}\|_\infty$ for each vertex z. Then, for each $z\in A_v$ from Eq. (15) we have

$$\sum_{e\in E(v)} w_e\alpha^\star_{ev} \le X_v + DFw_{e_z}\alpha^{\mathrm{R}}_{e_z z},$$

and hence from $|A_v|=FD^2/\varepsilon$

$$\sum_{e\in E(v)} w_e\alpha^\star_{ev} \le X_v + \sum_{z\in A_v} w_{e_z}\cdot\frac{\varepsilon}{D}\alpha^{\mathrm{R}}_{e_z z}. \tag{16}$$

We say that the vertices in A_v *amortize* for v. Observe that from the RHS of Eq. (16), $z\in A_v$ amortizes for v by only ε/D fraction of $\alpha^{\mathrm{R}}_{e_z z}$. Therefore, as long as z amortizes for at most D distinct vertices, it will in total amortize at most ε fraction of its contribution to the LHS of Eq. (9), which would be sufficient to prove this lemma. But, is there a way to define sets A_v so that each vertex amortizes for at most D distinct vertices? Next we show that the answer to this question is affirmative.

Amortizing for at most D vertices. Let $G^{\mathrm{R}}=(V,E^{\mathrm{R}})$ be the graph such that $E^{\mathrm{R}}\stackrel{\text{def}}{=}\bigcup_{v\in V}E^{\mathrm{R}}(v)$. Since G^{R} is a subgraph of G, the maximum subgraph density of G^{R} is at most D. From Theorem 22, there exists an orientation of the edges of G^{R} such that the indegree of each vertex is at most $\lceil D\rceil$. Consider one such orientation. As already discussed, vertices whose degree in G is at most $d^{\mathrm{R}}_{\max}$ do not need to be amortized for. So, select a vertex v whose degree in G is more than $d^{\mathrm{R}}_{\max}$. Choose A_v to be the set of any FD^2/ε neighbors of v in $E^{\mathrm{R}}(v)$ whose edges are outgoing from v. (Note that A_v of this size always exists as $|E^{\mathrm{R}}(v)|=d^{\mathrm{R}}_{\max}$ and the indegree of v is at most $\lceil D\rceil$.) The set A_v corresponds to the amortization described above.

Note that the existence of this orientation is needed only for the purpose of the analysis. □

E.5. Proof of Theorem 3

In this section we prove Theorem 3. We begin by inserting an implementation for Line 5 of Algorithm 6. This implementation is based on R-ORACLE$_2$ and is given in Algorithm 7.

E.5.1. CORRECTNESS

Let $D^\star$ be the maximum subgraph density of G. Let $\mathcal{D}$ be as defined on Line 1 of Algorithm 6, and let $D'\in\mathcal{D}$ be the smallest value such that $D'\ge D^\star$. Let $H_{D'}$ be as defined on Line 4, and let $D^\star_H$ be the maximum subgraph density of $H_{D'}$. Note that the sampling probability $(1+\varepsilon)D_{\text{sparse}}/D'$ can be written as $\tilde{D}/D^\star$ for $\tilde{D}=(1+\varepsilon)D_{\text{sparse}}\cdot D^\star/D'\in[D_{\text{sparse}},(1+\varepsilon)D_{\text{sparse}}]$. Hence, by Lemma 19 (C), whp $D^\star_H\le\min\{D^\star,(1+\varepsilon)^2D_{\text{sparse}}\}$. This implies that DUAL$_2$ defined on Line 5 of Algorithm 6 has a feasible solution.

The invocation of MWU on Line 5 is implemented by Algorithm 7. This algorithm uses R-ORACLE$_2$ to solve a given DUAL$_2$, and hence by Lemma 21 outputs a $(1-2\varepsilon)$ approximation to DUAL$_2$ defined on Line 5 of Algorithm 6. The

Input : $\text{DUAL}_2(D)$ defined as in Line 5 of Algorithm 6
Output : A sequence of vectors w^t used to solve the input $\text{DUAL}_2(D)$ by MWU

1 Let $R \leftarrow O(\log n/\varepsilon^2)$ be as defined in Lemma 21.
2 Let T be as defined in Eq. (10).
/* PHASES: */
3 Initialize $w^1 \leftarrow 1$, for each e
4 **for** $i \leftarrow 1$ *to* R/T **do**
5 Define the edge-sets $E^{\text{R}}(v)$ wrt to $w^{(i-1)T+1}$ as defined in Eq. (12).
6 Execute T iterations of MWU on the input DUAL_2 by using R-ORACLE_2 defined wrt to these $E^{\text{R}}(v)$ sets. Let $w^{(i-1)T+j+1}$ be the vector w after the j-th iteration of this MWU.
7 **return** the probability vectors $\{w^t\}_{t\in[R]}$ used to invoke R-ORACLE_2

Algorithm 7: Solving $\text{DUAL}_2(D)$ by MWU that as the oracle uses R-ORACLE_2.

vectors used to invoke R-ORACLE_2 in Algorithm 7 are then used on Line 6 to invoke ROUNDDENSEST with the goal to output a subgraph of $H_{D'}$ of density "close" to $D^\star_H$.

Next we derive a value of ε' for which the guarantee of Lemma 18 (B) for ROUNDDENSEST holds, where ε' is as defined in Lemma 18. By Lemma 19 (A), whp we have $D^\star_H \geq \min\{D^\star, (1-\varepsilon)\tilde{D}\}$, where from our derivation above $\tilde{D} \geq D_{\text{sparse}}$. Hence, for $\varepsilon \leq 1/5$, DUAL_2 defined on Line 5 of Algorithm 6 is invoked by density at most $(1+\varepsilon)^2/(1-\varepsilon) \leq 1+5\varepsilon$ larger than $D^\star_H$. This conclusion and the fact that the output of Algorithm 7 corresponds to $(1-2\varepsilon)$ approximation of DUAL_2 implies that it suffices to set $\varepsilon' = 5\varepsilon$. Finally, ROUNDDENSEST outputs vertex subset $S_{D'}$ such that it is a $(1-35\varepsilon)$ approximation of a densest subgraph of $H_{D'}$. By Lemma 19 (B), $S_{D'}$ is whp a $(1-70\varepsilon)$-approximate densest subgraph of G. Therefore, at Line 7 the algorithm returns a subgraph of G of density at least $(1-70\varepsilon)D^\star$.

Remark: Observe that in our computation we never verify whether the MWU should output "fail". That is, Algorithm 7 executes all the phases of MWU even if the underlying LP does not have a feasible solution. Nevertheless, this does not affect the correctness of our algorithm as for our use we only require that sufficiently dense subgraph is found. This dense subgraph is a certificate that the output of Algorithm 6 is a dense subgraph as well.

E.5.2. MPC IMPLEMENTATION

Each edge of G is copied $|\mathcal{D}|$ times, where $\mathcal{D}$ is defined on Line 1 of Algorithm 6. The algorithm processes $D \in \mathcal{D}$ on separate copies of the edges. H_D (Line 4) is obtained by including each edge of G independently of other edges, so the copy of each edge corresponding to D performs this operation locally.

When H_D is obtained, it is passed to Algorithm 7 via the corresponding instance of DUAL_2. Although Algorithm 7 gets DUAL_2 on its input, we do not actually need to construct this LP, but rather implement the required instance of R-ORACLE_2. This implementation is given within the loop on Line 4.

On Line 5 of Algorithm 7, for each vertex is constructed $E^{\text{R}}(v)$. Set $E^{\text{R}}(v)$ corresponds to at most $d^{\text{R}}_{\max}$ edges incident to v with their values largest in w. See Eq. (11) for the definition of $d^{\text{R}}_{\max}$, and also recall that for D in Algorithm 7 holds $D \in O(\frac{\log n}{\varepsilon^2})$ (see Line 5 of Algorithm 6). To construct all $E^{\text{R}}(v)$, for each edge $e = \{u, v\}$ we make two triples (u, w_e, v) and (v, w_e, u). Also, for each vertex v we make one extra triple (v, ∞, v). We then sort all the triples in ascending order with respect to the first coordinate, and in descending order with respect to the second. All the edges corresponding to $E^{\text{R}}(v)$ start at triple (v, ∞, v) and extend for $|E^{\text{R}}(v)|$ triples in the sorted sequence. Observe that $|E^{\text{R}}(v)| \leq d^{\text{R}}_{\max} \ll n^\delta$, hence $E^{\text{R}}(v)$ lies on one machine, or on two consecutive machines in the sorted order. Therefore, after the sorting, one additional round of computation suffices to fetch the edges of $E^{\text{R}}(v)$ that are not on the same machine as (v, ∞, v). For constant δ, these steps can be implemented in $O(1)$ MPC rounds. We refer a reader to (Goodrich et al., 2011) for details on how to efficiently perform sorting in MPC.

Line 6 of Algorithm 7 is implemented by gathering T-hop neighborhood in a similar way as described in Appendix C.2. The main difference between gathering T-hop neighborhood for Line 6 and in Appendix C.2 is that $B_i(v)$ defined for Line 6 might get significantly many more requests than what is its size, and on each of the requests should send $B_i(v)$; for

instance, consider the case where the underlying graph is a star. In (Ghaffari & Uitto, 2019) was shown how to handle all these requests in $O(1)$ MPC rounds with n^δ memory per machine.

As stated by Lemma 18, Line 6 of Algorithm 6 can be executed in $O(1)$ MPC rounds.

E.6. Proof of Lemma 19

Observe that for $\tilde{D} \geq D^\star$ we have that H equals G and the statement follows directly. Therefore, in the rest of the proof we assume that $\tilde{D} < D^\star$. We prove each of the points separately.

Proof of A. Let S be a subgraph of G on n_S vertices of density $D^\star$. Let $\tilde{S}$ be the subgraph of H on $V(S)$. We will show that $\tilde{S}$ has $(1 \pm \varepsilon)\tilde{D}n_S$ many edges.

By definition, $|E(S)| = D^\star n_S$. We then have

$$\mathbb{E}\left[|E(\tilde{S})|\right] = \frac{\tilde{D}}{D^\star}|E(S)| = \tilde{D}n_S.$$

Chernoff bound implies

$$\Pr\left[||E(\tilde{S})| - \tilde{D}n_S| > \varepsilon\tilde{D}n_S\right] < n^{-6}. \tag{17}$$

That is, $\tilde{S}$ has density $(1 \pm \varepsilon)\tilde{D}$ with probability at least $1 - n^{-6}$.

Proof of B. Fix an integer k, $1 \leq k \leq n$. Consider any subgraph T of G that consists of exactly k vertices and has density at most $(1 - 2\varepsilon)D$. By definition, we have $|E(T)| \leq (1 - 2\varepsilon)D^\star k$. Let $\tilde{T}$ be the subgraph of H on the vertex set $V(T)$. Then,

$$\mathbb{E}\left[|E(\tilde{T})|\right] = \frac{\tilde{D}}{D^\star}|E(T)| \leq (1 - 2\varepsilon)\tilde{D}k.$$

Hence, from Chernoff bound we derive

$$\Pr\left[|E(\tilde{T})| - (1 - 2\varepsilon)\tilde{D}k \geq \varepsilon\tilde{D}k\right] < n^{-6k}. \tag{18}$$

Observe that Eq. (18) implies that T has density at least $(1 - \varepsilon)\tilde{D}$ with probability at most n^{-6k}. Taking a union bound over all the subgraphs T of G on k vertices of density at most $(1 - 2\varepsilon)D^\star$, we obtain that none of them have density at least $(1 + \varepsilon)\tilde{D}$ in H with probability at least $\binom{n}{k}n^{-6k} \leq n^k n^{-6k} = n^{-5k}$. Taking a union bound over all k, we have that with probability at least $\sum_{k=1}^{n} n^{-5k} \leq n^{-4}$ no such subgraph T has has density at least $(1 - \varepsilon)\tilde{D}$ in H. The proof follows by taking a union bound of this event and Eq. (17).

Proof of C. Our proof of this property is similar to the one of B.

Fix an integer k, $1 \leq k \leq n$. Consider any subgraph T of G that consists of exactly k vertices. By definition, we have $|E(T)| \leq D^\star k$. Let $\tilde{T}$ be the subgraph of H on the vertex set $V(T)$. Then,

$$\mathbb{E}\left[|E(\tilde{T})|\right] = \frac{\tilde{D}}{D^\star}|E(T)| \leq \tilde{D}k.$$

Hence, from Chernoff bound we derive

$$\Pr\left[|E(\tilde{T})| - \tilde{D}k \geq \varepsilon\tilde{D}k\right] < n^{-6k}. \tag{19}$$

Observe that Eq. (19) implies that T has density at least $(1 + \varepsilon)\tilde{D}$ with probability at most n^{-6k}. The rest of the proof now follows by applying the same union bound as in the proof of B.