Near-Optimal Scheduling in the Congested Clique

Keren Censor-Hillel   Yannic Maus   Volodymyr Polosukhin

Technion *

Abstract

This paper provides three nearly-optimal algorithms for scheduling t jobs in the CLIQUE model. First, we present a deterministic scheduling algorithm that runs in $O(\mathsf{GlobalCongestion} + \mathsf{dilation})$ rounds for jobs that are sufficiently efficient in terms of their memory. The dilation is the maximum round complexity of any of the given jobs, and the GlobalCongestion is the total number of messages in all jobs divided by the per-round bandwidth of n^2 of the CLIQUE model. Both are inherent lower bounds for any scheduling algorithm.

Then, we present a randomized scheduling algorithm which runs t jobs in $O(\mathsf{GlobalCongestion} + \mathsf{dilation} \cdot \log n + t)$ rounds and only requires that inputs and outputs do not exceed $O(n \log n)$ bits per node, which is met by, e.g., almost all graph problems. Lastly, we adjust the *random-delay-based* scheduling algorithm [Ghaffari, PODC'15] from the CONGEST model and obtain an algorithm that schedules any t jobs in $O(t/n + \mathsf{LocalCongestion} + \mathsf{dilation} \cdot \log n)$ rounds, where the LocalCongestion relates to the congestion at a single node of the CLIQUE. We compare this algorithm to the previous approaches and show their benefit.

We schedule the set of jobs on-the-fly, without a priori knowledge of its parameters or the communication patterns of the jobs. In light of the inherent lower bounds, all of our algorithms are nearly-optimal.

We exemplify the power of our algorithms by analyzing the message complexity of the state-of-the-art MIS protocol [Ghaffari, Gouleakis, Konrad, Mitrovic and Rubinfeld, PODC'18], and we show that we can solve t instances of MIS in $O(t + \log \log \Delta \log n)$ rounds, that is, in $O(1)$ amortized time, for $t \geq \log \log \Delta \log n$.

1 Introduction

Motivated by the ever-growing number of frameworks for parallel computations, we address the complexity of executing multiple jobs in such settings. Such frameworks, e.g., MapReduce [KSV10], typically need to execute a long queue of jobs. A fundamental goal of such systems is to schedule many jobs in parallel, for utilizing as much of the computational power of the system as possible. Ideally, this is done by the system in a black-box manner, without the need to modify the jobs and, more importantly, without the need to know their properties and specifically their communication patterns beforehand.

In their seminal work, Leighton, Maggs, and Rao [LMR94] studied the special case where each of the to-be-scheduled jobs is a routing protocol that routes a packet through a network along a given path. The goal in their work is to schedule t jobs such that the *length* of the schedule, i.e., the overall runtime until all t packets have reached their destination, is minimized. They showed that

---

*{ckeren, yannic.maus, po}@cs.technion.ac.il

there exists an optimal packet-routing schedule of length $O(\mathsf{congestion} + \mathsf{dilation})$, where $\mathsf{congestion}$ is the maximum number of packets that need to be routed over a single edge of the network and $\mathsf{dilation}$ is the maximum length of a path that a packet needs to travel. Clearly, both parameters are lower bounds on the length of any schedule, implying that the above schedule is asymptotically optimal. Further, Leighton, Maggs, and Rao [LMR94] showed that assigning a random delay to each packet gives a schedule of length $O(\mathsf{congestion} + \mathsf{dilation} \cdot \log{(t \cdot \mathsf{dilation})})$.

In his beautiful work, Ghaffari [Gha15] raised the question of running multiple jobs in the distributed $\mathsf{CONGEST}$ model on n nodes. Applying the random delays method [LMR94], he showed a randomized algorithm which after $O(\mathsf{dilation} \cdot \log^2 n)$ rounds of pre-computation, runs a given a set of jobs in $O(\mathsf{congestion} + \mathsf{dilation} \cdot \log n)$ rounds. Here, in a similar spirit to [LMR94], $\mathsf{congestion}$ is the maximum number of messages that need to be sent over a single edge and $\mathsf{dilation}$ is the maximum round complexity of all jobs. Further, Ghaffari [Gha15] showed that this is nearly optimal, by constructing an instance which requires $\Omega(\mathsf{congestion} + \mathsf{dilation} \cdot \log n / \log \log n)$ rounds to schedule.

In this paper, we address the t-scheduling problem in the ($\mathsf{CONGESTED}$) CLIQUE model [LPPP05], in which each of n machines can send $O(\log n)$-bit messages to any other machine in each round. Our goal is thus to devise scheduling algorithms that run t jobs in a black-box manner, such that they complete in a number of rounds that beats the trivial solution of simply running the jobs sequentially one after the other, and, ideally, reaches inherent lower bounds that we discuss later. We emphasize that we schedule all jobs' actions *on-the-fly* during their execution. Throughout the paper, we use the terminology that a job is a *protocol* that n *nodes*, $v_0, \ldots, v_{n-1}$, need to run on some input, and we use the notion of an *algorithm* for the scheduling procedure that the n *machines*, $p_0, \ldots, p_{n-1}$, execute. Each machine p_i is given the inputs of the nodes v_i^j for all jobs j, and the machines run an algorithm which simulates the protocols of their assigned nodes.

Our contributions are three algorithms for scheduling t jobs in the CLIQUE model, which exhibit trade-offs based on the parameters of $\mathsf{dilation}$, $\mathsf{LocalCongestion}$, and $\mathsf{GlobalCongestion}$ of the set of jobs, which we formally define below. Our scheduling algorithms complete within round complexities that are nearly optimal w.r.t. the appropriate parameters.

1.1 Our Contributions

No scheduling algorithm can beat the $\mathsf{dilation}$ of the set of jobs, which is the maximum runtime of a job in the set, had this job been executed standalone. Similarly, another natural lower bound is given by the $\mathsf{GlobalCongestion}$, which is the total number of messages that all nodes in all jobs send over all rounds, normalized by the n^2 per-round-bandwidth of the CLIQUE model (for simplicity, this considers the possibility that a machine sends a message to itself). The main goal is thus to get as close as possible to these parameters.

As a toy example, consider a set of jobs in which each completes within a single round. Intuitively, if the total number of messages that need to be sent by all nodes in all jobs is at most n^2, then one could hope to squeeze all of these jobs into a single round of the CLIQUE model, as n^2 is the available bandwidth per round. The main hurdle in a straightforward argument as above, lies in the fact that a machine cannot send more than n messages in a round. Thus, although we are promised that in total there no more than n^2 messages, it might be that a machine is required to send/receive $\omega(n)$ messages because the heaviest-loaded nodes of multiple jobs might be located on the same machine.

This implies that a naïve scheduling, in which each machine simulates the nodes that are located at it, is more expensive than our single-round goal scheduling, as some messages must wait for later

rounds. In the general case, these issues become more severe, as the jobs may originally require more than a single round, and it could be that each round displays an imbalance in a different set of nodes and machines.

The key ingredient in the first two scheduling algorithms that we present is hence to *rebalance* the nodes among the machines, for the sake of a more efficient simulation that deals with the possible imbalance, which also may vary from round to round. The third scheduling algorithm we present is inspired by the random-delay approach of [LMR94, Gha15]. In what follows, we present the guarantees that are obtained by our three scheduling algorithms, and discuss the trade-offs that they exhibit.

Deterministic scheduling. A crucial factor in the complexity of rebalancing the nodes among the machines is the amount of information that needs to be passed from one machine to another in order for the latter to take over the simulation of a node. To this end, we define an *M-memory efficient* job as a job where for each node, its state can be encoded in $M \log n$ bits, and that the number of messages it needs to receive in this round can be inferred from its state. In Section 3, we obtain the following deterministic algorithm for scheduling t jobs that are M-memory efficient.

Theorem 3.1. *There is a deterministic algorithm that schedules $t = \text{poly } n$ jobs that are M-memory efficient in $O(\mathsf{GlobalCongestion} + \lceil M \cdot t/n \rceil \cdot \mathsf{dilation})$ rounds.*

At a very high level, in the algorithm for Theorem 3.1, the machines rebalance nodes in each round by sending the states of nodes. The main technical effort is that the reassignment needs to be computed by the machines *on-the-fly*, and we show how to do so in a fast way.

Notice that for the case that $M \cdot t = O(n)$, the round complexity we get from Theorem 3.1 is $O(\mathsf{GlobalCongestion} + \mathsf{dilation})$, which is *optimal*. Another crucial point is that our algorithm does not require the knowledge of either the $\mathsf{GlobalCongestion}$ or the $\mathsf{dilation}$ of the set of jobs.

Randomized scheduling. If we are given a set of jobs that are not memory efficient for a reasonable value of M, it may be too expensive to rebalance the nodes among the machines in every simulated round. However, if the input of each node is not too large, we can randomly shuffle the nodes at the beginning of the simulation, and if the output is also not too large then we can efficiently unshuffle, and reach the original assignment.

To capture this, we say that a job is *I/O efficient* if its input and output can be encoded within $O(n \log n)$ bits. Notice that most graph-related problems are I/O efficient, e.g., MST [LPPP05, HPP+15, GP16, Kor16, JN18, Now19], MIS [Gha17, GGK+18, CPS20], Mininum Cut [GN18, GNT20], as well as many algebraic problems [CKK+19, Gal16]. An example of a graph problem that is not I/O efficient is *k-clique listing*, in which all nodes together have to explicitly output *all* k-cliques in the input graph [DLP12, IG17, PRS18, CGL20, CPZ19] which can be as many as $\Omega(n^k)$, thus necessitating large outputs. While the k-clique listing problem is not *output efficient*, it is *input efficient*, and as it does not require a specific node to output a specific clique, one could also run several instances of the problem by omitting the output unshuffling step of our scheduling algorithm.

We obtain the following randomized algorithm for scheduling t jobs that are I/O efficient.

Theorem 4.1. *There is a randomized algorithm in the CLIQUE model that schedules $t = \text{poly } n$ jobs that are I/O efficient in $O(t + \mathsf{GlobalCongestion} + \mathsf{dilation} \cdot \log n)$ rounds, w.h.p.[1]*

---

[1] An event occurs *w.h.p.* (*with high probability*) if for an arbitrary constant $c \geq 1$, the probability that the event occurs is at least $1 - n^{-c}$, where n is the number of machines. All our results can be adapted to any constant c at the cost of increasing the runtime by a constant factor.

As the deterministic scheduling algorithm (Theorem 3.1), the scheduling algorithm of Theorem 4.1 requires neither the knowledge of GlobalCongestion nor the knowledge of dilation.

Both of our scheduling algorithms for Theorem 3.1 and Theorem 4.1 have the machines possibly simulate the execution of nodes that are not originally assigned to them. We stress that any black-box scheduling algorithm in which each machine only simulates the nodes that are originally assigned to it must inherently suffer from another type of congestion as a lower bound on its round complexity, namely, the maximum number of messages that all nodes assigned to a single machine have to send or receive, normalized by the bandwidth n that each machine has per round. We call this the LocalCongestion of a set of jobs. We obtain the following *random-delay-based* algorithm for scheduling any t jobs, without reassigning nodes.

Theorem 4.4 (Simplified). *There is a randomized algorithm in the* CLIQUE *model that schedules* $t = \text{poly } n$ *jobs in* $O(t/n + \text{LocalCongestion} + \text{dilation} \cdot \log n)$ *rounds w.h.p.*

The stated complexity in the above simplified version of Theorem 4.4 requires the knowledge of the LocalCongestion, but this can be eliminated using a standard doubling approach, at the cost of a logarithmic multiplicative factor (see precise statement in Section 4).

The random-delay algorithm which gives Theorem 4.4 is suboptimal for a set of jobs which have a single machine with heavily-loaded nodes assigned to it, since in this case it does not exploit the entire bandwidth of the CLIQUE model. For example, for a problem with inputs of at most $O(n \log n)$ bits per node, a protocol in which a fixed leader learns the entire input takes $O(n)$ rounds, where on each round each node sends one message to the leader, who receives n messages. For n such jobs, the GlobalCongestion is n, while the LocalCongestion is n^2. In such a setting, our random-shuffling algorithm from Theorem 4.1 outperforms the random-delay algorithm from Theorem 4.4. One may suggest to replace the fixed leader by a randomly or more carefully chosen leader. However, this trick might be more complicated in the general case: suppose now that $n^{0.9}$ nodes need to learn $n^{1.1}$ messages each. For such a set of jobs, it holds that LocalCongestion $= n^{0.1}$, while GlobalCongestion $= 1$. Thus, it is more efficient to run Theorem 4.1 in this case. Another crucial example in which random-shuffling outperforms random-delays is the maximal independent set protocol that we describe below. Note that our algorithms address these cases in a black-box manner without assuming knowledge of the communication pattern.

Applications. In Section 5, we present two applications in order to exemplify our scheduling algorithms. We summarize these applications below and defer a more detailed discussion to Section 5 and Section 6.

A *maximal independent set* (MIS) of a graph $G = (V, E)$ is a set $M \subseteq V$ such that no two nodes in M are adjacent and no node of V can be added to M without violating this condition. The state-of-the-art randomized CLIQUE protocol for solving the MIS problem completes in $O(\log \log \Delta)$ rounds, w.h.p., where Δ is the maximum degree of the graph [GGK+18]. We analyze the message complexity of this protocol, and show that it does not utilize the entire bandwidth. Thus, we can schedule multiple MIS jobs efficiently using our random shuffling scheduling algorithm from Theorem 4.1, and we obtain the following theorem.

Theorem 5.1 (Multiple MIS instances). *There is a randomized algorithm in the* CLIQUE *model which solves* $t = \text{poly } n$ *instances of MIS in* $O(t + \log \log \Delta \log n)$ *rounds, w.h.p.*

Another application that exemplifies our scheduling algorithms is a variant of the *pointer jumping* problem, which is a widespread algorithmic technique [Hir76]. In the P-pointer jumping problem,

each node has a permutation on P elements. A fixed node has a value *pointer p* and should learn the result of applying these permutations one after another on p. Pointer jumping can be solved by an $O(\log n)$-round protocol in the CLIQUE model by learning the composition of all permutations (see Section 5.2). We observe that this protocol does not utilize the entire bandwidth and leverage this for obtaining an algorithm that executes multiple instances of this protocol efficiently.

Theorem 5.5 (Pointer Jumping). *For $P \leq n$, there are algorithms in the CLIQUE model that solve $t = \operatorname{poly} n$ instances of the P-pointer jumping problem deterministically in $O(\lceil P \cdot t/n \rceil \cdot \log n)$, and randomized in $O(t + \log^2 n)$ rounds, w.h.p.*

We obtain the deterministic result using our scheduling algorithm in Theorem 3.1 and the randomized result using our random-shuffling scheduling algorithm in Theorem 4.1. The proposed simple $O(\log n)$ round pointer jumping protocol also serves as an example where scheduling jobs via the random-shuffling approach of Theorem 4.1 is significantly better than the random-delay based approach of Theorem 4.4. For more details we refer to Section 5.2.

In Section 6 we discuss the amortized versions of these results, and present a small example of a set of jobs that can be scheduled with $o(1)$-amortized complexity. In light of the growing number of $O(1)$-round CLIQUE-protocols, e.g., [CDP20, Now19, GNT20], we propose the amortized complexity of solving many instances of a problem in parallel, as a valuable measure for the efficiency in future research.

1.2 Related Work

Many graph problems are studied in the CLIQUE model. There are fast protocols for the CLIQUE model for distance computations [CKK+19, Gal16], minimum spanning tree (MST) [LPPP05, GP16, Kor16, Now19], MIS [Gha17, GGK+18, CPS20], and more.

To the best of our knowledge, there are no previous works that study the scheduling of jobs in the CLIQUE model. In the past, it has been shown that running multiple instances of *the same* protocol on different inputs can result in fast algorithms for some complex problems. We survey some of these. Hegeman et al. [HPP+15] reduce the MST problem to multiple smaller instances of graph connectivity, breaking below the long-standing upper bound of $O(\log \log n)$ by Lotker et al. [LPPP05]. Further variants and improvements on the MST problem [Kor16, GP16, JN18, Now19] all exploit invoking multiple instances of sparser problems. This line of work culminated in the deterministic $O(1)$-round algorithm of Nowicki [Now19].

In [GN18], Ghaffari and Nowicki show a randomized algorithm which solves $O(n^{1-\epsilon})$ many instances of the MST problem in $O(\epsilon^{-1})$ rounds. This is used for finding the minimum cut of a graph. The state-of-the-art $O(1)$-round algorithm for the minimum cut problem, by Ghaffari et al. [GNT20], runs $\Theta(\log n)$ instances of connected components as a subroutine. The complexity of computing multiple matrix multiplications in parallel was explored by Le Gall [Gal16] and was used in the same paper to solve the *all-pairs-shortest-path problem*.

The notion of LocalCongestion is somewhat similar to the notion of Communication Degree Complexity [KNPR15]. The difference lies in the fact that the Communication Degree Complexity is an upper bound on the number of messages sent or received by any node on *any round*, while LocalCongestion is an upper bound on the *total* number of messages sent or received by any node *over all* rounds.

2   Preliminaries

The CLIQUE Model.   In the (CONGESTED) CLIQUE model, n machines $p_0, \ldots, p_{n-1}$ communicate with each other in synchronous rounds in an all-to-all fashion. In each round, any pair of machines can exchange $O(\log n)$ bits. There is usually no constraint neither on the size of the *local memory* nor on the time complexity of the *local computations*. Besides the local memory, each machine has a *read-only input buffer* and a *write-only output buffer*, as well as *read/write incoming- and outgoing- message buffers*.

Routing in the CLIQUE Model.   Lenzen's routing scheme [Len13] says that a set of messages can be routed in the CLIQUE model within $O(1)$ rounds, given that each machine sends and receives at most $O(n)$ messages. We formally state it here in its generalized version, which addresses the case of more than a linear number of messages. In the generalized version, each machine p_i holds a set of messages $M_i = \bigcup_{i' \in [n]} M_i^{i'}$, where $M_i^{i'}$ is a set of messages with the destination $p_{i'}$. The claim follows by having each node chop its set of messages M_i into chunks of n messages, each of which containing $|M_i^{i'}| n / X$ messages for each $i' \in [n]$, and applying the original routing scheme X/n times. The routing scheme could be adapted to preserve the message complexity in the following way.[2] Let $Y = \sum_{i \in [n]} |M_i| \leq n^2$ be the total number of messages. First, compute a global numbering of messages and the total number of messages Y. Then, send $O(\lceil \sqrt{Y} \rceil)$ messages to each one of the first $O(\lceil \sqrt{Y} \rceil)$ machines via intermediate nodes based on the numbering. Sort messages by the destination in the using Lenzen's sorting algorithm [Len13] over $O(\lceil \sqrt{Y} \rceil)$-clique. Finally, deliver the messages to their destinations via intermediate nodes based on the indices of messages in the sorted sequence. The round complexity of the algorithm is $O(1)$ and the message complexity of the algorithm in $O(Y + \lceil \sqrt{Y} \rceil \cdot \lceil \sqrt{Y} \rceil) = O(Y)$

Claim 2.1 (Lenzen's Routing Scheme)**.** *Let X be a globally known value and let $\mathcal{P}$ be the property that $|M_i| \leq X$ for all $i \in [n]$ and $\sum_{i \in [n]} |M_i^{i'}| \leq X$ for all $i' \in [n]$. There is an algorithm in the CLIQUE model which completes in $O(\lceil X/n \rceil)$ rounds and $O(\sum_{i \in [n]} M_i)$ messages, and delivers all messages if $\mathcal{P}$ holds, or indicates that it does not hold.*

Protocols and Jobs.   A *protocol* is run on an *input*, that is provided in a distributed manner in the *read-only input buffer* of each machine. The *complexity* of a protocol is the number of synchronous rounds until each machine has finished writing its output to its *write-only output buffer*.

A *job* is an instance of a protocol together with a given input and a job is *finished* when each machine has written its output. We generally assume that each job finishes in $O(\text{poly } n)$ rounds.

For our purposes of fast scheduling, we need to specify the internals of each synchronous round. We follow the standard description, which is usually omitted and simply referred to as a 'round'. We require that for each machine, the input and output buffers are only accessed in the first and last rounds of the protocol on that machine, respectively. In particular, this means that any further access to the input requires storing it in the local memory. Accessing the incoming- and outgoing-message buffers is not restricted to certain rounds. Each synchronous round of a protocol consists of 3 steps, in the following order.

- **Receiving Step:** Read from incoming-message buffer (or from input buffer if this is the first round), possibly modifying the local memory.
- **Computation Step:** Possibly modify local memory.

---

[2]We thank an anonymous reviewer for pointing this out.

- **Sending Step:** Write to outgoing-message buffer, (or to output buffer if this is the last round), possibly modifying the local memory.

After these 3 phases, all messages written in outgoing-message buffers are delivered into the incoming-message buffers of their targets.

The Scheduling Problem. In the *t-scheduling problem* (or simply a scheduling problem, if t is clear from the context) the objective is to execute t jobs. Since our goal is to do this in an efficient manner, we wish to allow a machine to simulate a computation that originally should take place in a different machine, in a naïve execution of the t jobs. To this end, we distinguish between the physical machine and the *nodes*, which are the virtual machines that need to execute each job. That is, for each job j we denote by $\{v_{i,j} | i \in [n]\}$ the set of nodes that need to execute job j.

Formally, in the t-scheduling problem, the input for machine p_i is composed of the inputs of all the nodes with identifiers of the form $v_{i,j}$ for each job $j \in [t]$. We also assume that each machine knows the protocol for each of the t jobs. An algorithm *solves the scheduling problem* or *schedules the jobs* when each job has finished writing its output. That is, for deterministic jobs, we require each machine p_i to write the output of nodes $v_{i,j}$ for all $j \in [t]$. For randomized jobs, the machines' output distribution for each job has to be equal to the distribution of outputs in a naïve execution of the job. In the rest of the paper, we refer to the scheduling solution as an *algorithm*, while we use the term *protocol* only for the content of a job.

Notations. Following the widespread conventions, we denote by log the logarithm base 2, and by ln the natural logarithm. Also, we denote $[n] = \{0, 1, \ldots, n-1\}$. We denote by $s_{i,j}^r$ and $t_{i,j}^r$ the number of messages sent and received by $v_{i,j}$ in round r, respectively. If job j terminates before round r, we indicate $s_{i,j}^r = t_{i,j}^r = 0$. We sometimes drop the superscript r, when it is clear from the context. We denote by ℓ_j the round complexity of job j and by $m_j = \sum_{i \in [n], r \in [\ell_j]} s_{i,j}^r = \sum_{i \in [n], r \in [\ell_j]} t_{i,j}^r$ the total number of messages sent or received during the execution of job j, i.e., the message complexity of job j. Another notation we extensively use is $m^r = \sum_{i \in [n], j \in [t]} s_{i,j}^r = \sum_{i \in [n], j \in [t]} t_{i,j}^r$, which is the number of messages all nodes in all jobs sent or received during round r.

Congestion parameters. We define the normalized GlobalCongestion as the total number of messages sent by all the jobs divided by n^2, and normalized LocalCongestion as the maximum number of messages send to or received by some node in the entire course of the execution of all jobs divided by n. Formally, dilation $= \max_{j \in [t]} \ell_j$,

$$\mathsf{GlobalCongestion} = \sum_{j \in [t]} m_j = \sum_{i \in [n]} \sum_{j \in [t]} \sum_{r \in [\ell_j]} s_{i,j}^r / n^2 = \sum_{r \in [\mathsf{dilation}]} m^r / n^2,$$

$$\mathsf{LocalCongestion} = \max \left\{ \max_{i \in [n]} \sum_{j \in [t]} \sum_{r \in [\ell_j]} s_{i,j}^r / n, \max_{i \in [n]} \sum_{j \in [t]} \sum_{r \in [\ell_j]} t_{i,j}^r / n \right\}.$$

Hoeffding bound. Some of our proofs use the following Hoeffding bound.

Claim 2.2 (Hoeffding Bound [Hoe63]). *Let $\{X_i\}_{i=1}^n$ be independent random variables with values in the interval $X_i \in [0, 1]$ and expectation of their sum bounded by $E\left[\sum_{i=1}^n X_i\right] \leq \mu$. Then for all $\epsilon > 0$*

$$\Pr\left[\sum_{i=1}^n X_i \geq (1+\epsilon)\mu\right] \leq \left(\frac{e^\epsilon}{(1+\epsilon)^{1+\epsilon}}\right)^\mu \leq e^{-\frac{\epsilon^2}{2+\epsilon}\mu}.$$

3   Deterministic Scheduling

The objective of this section is to prove the following theorem.

Theorem 3.1. *There is a deterministic algorithm that schedules $t = \text{poly}\, n$ jobs that are M-memory efficient in $O(\mathsf{GlobalCongestion} + \lceil M \cdot t/n \rceil \cdot \mathsf{dilation})$ rounds.*

The formal definition of an M-memory efficient job as used in Theorem 3.1 is as follows.

Definition 3.2 (M-**memory efficient job**). *For a given value M, an M-memory efficient job is a job in which for each node v in each round r, the state (local memory) of v at the end of the Computation Step can be encoded in $M \log n$ bits. In addition, there is a function that, given the state of node v after the Computation Step of round r, infers the number of messages it sends and receives on this round.*

Theorem 3.1 requires that jobs use at most M bits of local memory per machine. Thus, the power of the result is when $M = o(n)$, as otherwise the naïve execution of jobs one after another schedules them in $\mathsf{dilation} \cdot t$ rounds. In the case that $M \cdot t = O(n)$, the runtime becomes $O(\mathsf{GlobalCongestion} + \mathsf{dilation})$, which is optimal up to a constant factor as, clearly, any schedule for any collection of jobs requires at least $\Omega(\mathsf{GlobalCongestion} + \mathsf{dilation})$ rounds.

To schedule the jobs for Theorem 3.1, we work in epochs. Each machine p_i first simulates round 0 up to the end of the Computation Step for the nodes $v_{i,j}$, for each $j \in [t]$. This does not require any communication. Then, the epochs are such that for each round r, at the start of epoch r, all nodes in all jobs are at the end of the Computation Step of round r. Clearly, for each simulated node that finishes in round r, the machine does not need to do anything for the part that executes the beginning of round $r+1$. The reason why we execute the protocol in these *shifted* epochs, from Sending Step of round r (including) to Sending Step of round $r+1$ (excluding), lies in the fact that the bottleneck is the possible imbalance in communication.

Recall that m^r denotes the number of messages all nodes from all jobs send in round r. Since in each round of the CLIQUE model, at most n^2 messages can be exchanged, routing m^r messages cannot be done faster than $\lceil m^r/n^2 \rceil$ rounds. We aim to execute an epoch in this optimal number of $O(\lceil m^r/n^2 \rceil)$ rounds. We start with the simple case and then use it to solve the general case.

The first case is when $m^r \leq 2n^2$. In Lemma 3.4, we show that in this case, we can route all m^r messages in $O(\lceil M \cdot t/n \rceil)$ rounds. The challenge we encounter is that although $m^r \leq 2n^2$, we are not promised that the messages are balanced across the machines in the following sense. It is possible that some machine p_i, which simulates the nodes $v_{i,j}$, for all jobs $0 \leq j < t$, is required to send significantly more than n messages when summing over all messages that need to be sent by these nodes $v_{i,j}$. We overcome this issue by assigning the simulation of some of these nodes to some other machine $p_{i'}$, which originally has a smaller load of messages to send. The crux that underlies our ability to defer a simulation of a node $v_{i,j}$ to a machine $p_{i'}$ is that the state of the node does not consume too many bits. We show how to compute a well-balanced assignment of nodes to machines in Claim 3.3. This assignment allows us to execute the epoch in the claimed number of $O(\lceil M \cdot t/n \rceil)$ rounds.

In the general case, we can have $m^r > 2n^2$. We show how to carefully split up the messages that need to be sent into chunks that allow us to use multiple invocations of Lemma 3.4. This allows us to execute the epoch in the $O(\lceil m^r/n^2 + M \cdot t/n \rceil)$ rounds. As the core of our algorithm is handling the case $m^r \leq 2n^2$, now, we focus on the case $m^r \leq 2n^2$.

We start with the following notation. An *assignment* of nodes to machines corresponds to a function $\varphi\colon [n] \times [t] \mapsto [n]$, where $\varphi(i,j) = k$ says that the i-th node in job j, i.e., $v_{i,j}$, is assigned to the k-th machine p_k. We sometimes abuse notation and write that $\varphi(v_{i,j}) = p_k$ for $\varphi(i,j) = k$. We call an assignment *balanced*, if the number of nodes assigned to each machine is $O(t)$, i.e., if for each k, it holds that $|\varphi^{-1}(p_k)| = O(t)$. The (balanced) assignment $\varphi(i,j) = i$ is called the *trivial* assignment.

We denote by $S_{i,j,r}$ the state of node $v_{i,j}$ after its Computation Step in round r.

Claim 3.3 (Distributing the states)**.** *Given are t jobs that are M-memory efficient, and globally known initial and final balanced assignments, φ_s and φ_f, respectively. Assume that for each $i \in [n]$ and $j \in [t]$, machine $\varphi_s(i,j)$ holds the state $S_{i,j,r}$ of node $v_{i,j}$ after its Computation Step in round r. Then, there exists a deterministic* CLIQUE *algorithm which completes in $O(\lceil M \cdot t/n \rceil)$ rounds and moves the states according to φ_f, that is, at the end of the algorithm, for each $i \in [n]$ and $j \in [t]$, machine $\varphi_f(i,j)$ holds the state $S_{i,j,r}$ of node $v_{i,j}$.*

Proof of Claim 3.3. For each node $v_{i,j}$, denote $i' = \varphi_f(i,j)$. For each node $v_{i,j}$ such that $i'' = \varphi_s(i,j)$, machine $p_{i''}$ sends $S_{i,j,r}$ to machine $p_{i'}$. Overall, each machine p_i sends and receives $|\varphi_s^{-1}(p_i)| \cdot M = O(t \cdot M)$, $|\varphi_f^{-1}(p_i)| \cdot M = O(t \cdot M)$ messages. Thus, by Claim 2.1, it completes in $O(\lceil M \cdot t/n \rceil)$ rounds. ∎

Lemma 3.4 (Scheduling of a round with $m^r \leq 2n^2$ messages)**.** *Given are t jobs that are M-memory efficient, and given is a round number, r, for which $m^r \leq 2n^2$. Assume that for each $i \in [n]$, p_i holds $S_{i,j,r}$ for all $j \in [t]$. Then there exists a deterministic* CLIQUE *algorithm which completes in $O(\lceil M \cdot t/n \rceil)$ rounds, at the end of which, for each $i \in [n]$, p_i holds $S_{i,j,r+1}$ for all $j \in [t]$.*

The outline of the algorithm is as follows. Each machine partitions its simulated nodes into buckets of contiguous ranges of indices, such that nodes in each bucket send and receive $O(n)$ messages altogether. Thus, the messages of all nodes in the bucket can be sent or received by a single machine. We show that the number of buckets over all machines is $O(n)$. The machines collectively assign the buckets such that each machine gets $O(1)$ buckets, and they make the assignment globally known. Then, the states $S_{i,j,r}$ are distributed according to the assignment using Claim 3.3, and each machine executes the Sending Step of round r for each of its newly assigned nodes and all messages get delivered. Then, each machine executes the remainder of the protocol of its newly assigned nodes until after the Computation Step of round $r+1$. Finally, the states $S_{i,j,r+1}$ for round $r+1$ are distributed back according to the trivial assignment.

Proof of Lemma 3.4. We begin with describing the algorithm (see Algorithm 1). Afterwards, we prove the correctness and analyze the round complexity.

Algorithm 1 Simulating a round with $m^r \leq n^2$.

1: Compute the balanced assignment $\varphi\colon [n] \times [t] \mapsto [n]$.
2: Distribute the states according to the assignment φ.
3: Execute the protocol for round r accounting for φ.
4: Distribute the states back according to the trivial assignment.

The Algorithm. We first show how to split nodes into buckets. Then we show how to compute a globally known assignment φ, distribute the nodes according to φ, execute the jobs until after the next Computation Step, and assign nodes back to their initial machines.

Forming buckets (locally): Each machine p_i for each $j \in [t]$ uses $S_{i,j,r}$ to locally compute $s_{i,j}$ and $t_{i,j}$, the number of messages each node $v_{i,j}$ sends and receives in round r, respectively. This is possible by the definition of an M-memory efficient job. Let $S_i = \sum_{j=0}^{t-1} s_{i,j}$ and $T_i = \sum_{j=0}^{t-1} t_{i,j}$. Then, each machine p_i (locally and independently) applies [CDKL19, Lemma 7] (restated in Claim 3.5 for better readability) with $k = k_i = \lceil \max\{ S_i/n, T_i/n \} \rceil$ to the sequences $(s_{i,j})_{j=0}^{t-1}$ and $(t_{i,j})_{j=0}^{t-1}$, to split its nodes into k_i buckets $B_{i,0}, \ldots, B_{i,k_i-1}$ of continuous ranges of jobs' indices.

Claim 3.5 (Lemma 7 from [CDKL19]). *Let $s_0, \ldots, s_{n-1} \in \mathbb{N}$ and $t_0, \ldots, t_{n-1} \in \mathbb{N}$ be sequences of natural numbers where each number is upper bounded by s and t, respectively. Let $S = \sum_{j \in [n]} s_j$ and $T = \sum_{j \in [n]} t_j$. Then for any $k \in \mathbb{N}$, there is a partition of $[n]$ into k sets $B_0, \ldots, B_{k-1}$, such that for each i, the set B_i consists of consecutive elements, and*

$$\sum_{j \in B_i} s_j \leq 2 \left(\frac{S}{k} + s \right) \quad and \quad \sum_{j \in B_i} t_j \leq 2 \left(\frac{T}{k} + t \right).$$

Invoking Claim 3.5 with $s = n \geq s_{i,j}$, $t = n \geq t_{i,j}$, $S = S_i$, and $T = T_i$, implies that for each $i \in [n]$ and $i' \in [k_i]$, the nodes inside each bucket $B_{i,i'}$ want to send/receive at most $4n$ messages, i.e.,

$$\sum_{j \in B_{i,i'}} s_{i,j} \leq 2 \left(\frac{S}{k} + s \right) \leq 2 \left(\frac{S_i}{(S_i/s)} + s \right) = 4s = 4n, \text{ and}$$

$$\sum_{j \in B_{i,i'}} t_{i,j} \leq 2 \left(\frac{T}{k} + t \right) \leq 2 \left(\frac{T_i}{(T_i/t)} + t \right) = 4t = 4n.$$

Computing the assignment φ: We first define the assignment φ and then show how it becomes globally known. Recall that the buckets of machine p_i are numbered from 0 to $k_i - 1$ and define the following value for $i \in [n]$ and $i' \in [k_i]$:

$$f(i, i') = \left\lfloor \left(i' + \sum_{i'' < i} k_{i''} \right) / 5 \right\rfloor.$$

Then, we define the assignment φ to assign all nodes in bucket $B_{i,i'}$ to machine $p_{f(i,i')}$. Notice that this is a valid assignment because with $\sum_i S_i \leq 2n^2$ and $\sum_i T_i \leq 2n^2$ (due to $m^r \leq 2n^2$) we obtain

$$f(i, i') < \sum_{0 \leq i < n} \frac{k_i}{5} = \frac{1}{5} \sum_i \lceil \max\{ \frac{S_i}{n}, \frac{T_i}{n} \} \rceil \leq \frac{1}{5} \sum_i \left(\frac{S_i}{n} + \frac{T_i}{n} + 1 \right) \leq \frac{1}{5} \cdot 5n = n.$$

Here, the first inequality follows from $i' < k_{i'}$. Also, notice that each machine receives at most 5 different buckets because at most five pairs (i, i') are mapped to the same index by f.

Now, we want to make the assignment φ globally known to all machines. To this end, each machine p_i broadcasts the number of its buckets, k_i. Thus, machine p_i can compute $f(i, i')$ for each of its buckets $B_{i,i'}$. Then, for all $i' \in [k_i]$, machine p_i informs machine $p_{f(i,i')}$ about the smallest and the largest job number of a node in bucket $B_{i,i'}$. As the buckets $B_{i,1}, \ldots, B_{i,k_i}$ are ordered (increasingly) by the jobs' indices for all $i \in [n]$, this information is sufficient for each machine to deduce which nodes are assigned to it in φ. In the last step, each machine broadcasts the messages

that it has received, i.e., machine $p_{f(i,i')}$ broadcasts the smallest and largest job index of bucket $B_{i,i'}$ together with the index i, and each machine can deduce the full assignment φ.

Executing round $r + 1$: We now use Claim 3.3 to distribute the states $S_{i,j,r}$ from the trivial initial assignment $\varphi_s(i,j) = i$ to the globally known final assignment $\varphi_f = \varphi$. Then, each machine executes the Sending Step of round r for each of its newly assigned nodes, where a message from $v_{i,j}$ to $v_{i',j}$ is sent from $p_{\varphi(i,j)}$ to $p_{\varphi(i',j)}$. This is possible since φ is globally known. Then, each machine executes the remainder of the protocol of its newly assigned nodes until after the Computation Step of round $r + 1$. Finally, the obtained states $S_{i,j,r+1}$ for round $r + 1$ are re-distributed according to the trivial assignment by using Claim 3.3 once more, with $\varphi_s = \varphi$ and $\varphi_f(i,j) = i$.

Correctness. For each $i \in [n]$ and $j \in [t]$ the machine $p_{f(i,j)}$ receives the state $S_{i,j,r}$ and executes the Sending Step of round r, the Receiving Step of round $r+1$, and the Computation Step of round $r+1$ for node $v_{i,j}$. Thus, afterwards it holds the state $S_{i,j,r+1}$. Since this state is then sent back to p_i, the correctness follows.

Round Complexity. The partitioning of each machine's nodes into buckets is done locally without communication. Broadcasting the number of buckets (the value of k_i) can be done in a single round. We next reason about the time complexity that is required to make the assignment φ globally known. The computation of $f(i,i')$ is done locally. Informing machine $p_{f(i,i')}$ about the smallest and largest job in the bucket $B_{i,i'}$ requires for each machine p_i to send at most t messages and to receive at most 5 messages. Thus, by $\lceil t/n \rceil$ invocations of Claim 2.1, this step completes in $O(\lceil t/n \rceil)$ rounds. Since each machine p_i is assigned at most 5 buckets, and for each bucket $B_{i',j}$ it broadcasts a constant number of elements (smallest and largest job index in it together with the identifier i'), this step completes in $O(1)$ rounds.

The runtime is hence dominated by distributing the states via Claim 3.3, which takes $O(\lceil M \cdot t/n \rceil)$ rounds. All nodes in a bucket send/receive at most $4n$ messages in total and each machine executes the sending/receiving phase for at most 5 buckets, and thus these steps are done in $O(1)$ rounds by Claim 2.1. □

The next lemma deals with the general case, where total number of messages m^r might be larger than $2n^2$.

Lemma 3.6 (Scheduling of a round r.)**.** *Given are t jobs that are M-memory efficient, and given is a round number r. Assume that for each $i \in [n]$, p_i holds $S_{i,j,r}$ for all $j \in [t]$. Then there exists a deterministic CLIQUE algorithm which completes in $O(\lceil m^r/n^2 + M \cdot t/n \rceil)$ rounds, at the end of which, for each $i \in [n]$, p_i holds $S_{i,j,r+1}$ for all $j \in [t]$.*

The proof of Lemma 3.6 uses the next claim to split all jobs into chunks that send smaller numbers of messages in order to apply Lemma 3.4.

Claim 3.7. *Let $\mathcal{S}$ be a non-empty (globally known) set of consecutive indices of size at most n^c for some constant $c > 0$ and let $x > 0$. Each machine p_i has a sequence of numbers $(s_{i,j})_{j \in \mathcal{S}}$ that are all upper bounded by n. There is a deterministic algorithm in the CLIQUE model, which in $O(1)$ rounds finds the minimum index $j_0 \in \mathcal{S}$ (if it exists) that satisfies*

$$x \leq \sum_{\substack{j \in \mathcal{S}, \\ j \leq j_0}} \sum_{i=0}^{n-1} s_{i,j} \leq x + n^2. \tag{1}$$

We solve this problem in c recurrent levels. On recursion level c', which goes from c down to 1, we start with the search space $\mathcal{S}^{c'}$ of size $n^{c'}$ and finish with the search space $\mathcal{S}^{c'-1}$ of size $n^{c'-1}$.

After each iteration, we maintain the invariant that if there exists the required j_0 then $j_0 - 1 \in \mathcal{S}^{c'-1}$ and that $\mathcal{S}^{c'-1}$ is contiguous. We always maintain a search space of consecutive indices.

Next, we explain the c'-th recursion level, and for that purpose assume that the current search space $\mathcal{S}^{c'}$ is of size $n^{c'}$ for $0 \leq c' \leq c$. If this is not the case, we append dummy indices to make $\mathcal{S}^{c'}$ of the size exactly $n^{c'}$. To narrow down the search space, we compute n prefix sums $S_{\ell_1}, \ldots, S_{\ell_n}$ where $S_{\ell_{i'}}$ sums up all values with index $j < \ell_{i'}$ of all machines. The indices $\ell_0 = \min \mathcal{S}, \ldots, \ell_n = \ell_0 + n^{c'}$ are equidistantly placed in $\mathcal{S}^{c'}$. Let i' be the largest index such that $S_{\ell_{i'}} < x$. The new search space is formed by the indices $\mathcal{S}^{c'-1} = [\ell_{i'}, \ell_{i'+1})$.

After the last recursion level we obtain singleton search space $\mathcal{S}^0$. We return j_0 as that value plus one if it is less than n^c. Otherwise, respond that the required j_0 does not exist.

Proof of Claim 3.7. Algorithm: Initially we may assume that the search space $\mathcal{S}$ is of size exactly n^c. If this is not the case, we append dummy indices to the end of $\mathcal{S}$, in other words we add the indices $\{ \max \mathcal{S} + 1, \max \mathcal{S} + 2, \ldots, \max \mathcal{S} + n^c - |\mathcal{S}| \}$ to $\mathcal{S}$, obtaining the range of indices $[\min \mathcal{S}, \ldots, \min \mathcal{S} + n^c - 1]$. We proceed in c recursion levels, in each of which we decrease the size of the search space by a factor of n, while always maintaining a search space of consecutive indices.

Consider iteration c' with the search space $\mathcal{S}^{c'}$. Let ℓ_0 be the smallest index in $\mathcal{S}^{c'}$ and for $1 \leq i' \leq n$ let $\ell_{i'} = \ell_0 + i' \cdot n^{c'-1}$. Now, each machine p_i builds prefix sums $S^i_{\ell_1}, \ldots, S^i_{\ell_n}$ of its own numbers, that is

$$S^i_{\ell_{i'}} = \sum_{j \in 0 \leq j < \ell_{i'}} s_{i,j}.$$

Then, all machines send their computed prefix sum corresponding to $\ell_{i'+1}$ to machine $p_{i'}$ which sums up all received prefixes, that is, afterwards machine $p_{i'}$ holds $S_{\ell_{i'+1}} = \sum_{i \in [n]} S^i_{\ell_{i'+1}}$. In a second round of communication $S_{\ell_1}, \ldots, S_{\ell_n}$ are broadcasted and every node can determine the new search space $\mathcal{S}^{c'-1} = [\ell_{i'}, \ell_{i'+1})$ where i' is the largest number such that the prefix sums $S_{\ell_{i'}}$ add up to less than x. After c levels of recursion the search space consists of a single index ℓ. We return $j_0 = \ell + 1$. **Correctness:** By induction, before level c' the search space size is $|\mathcal{S}^{c'}| = n^{c'}$ and the largest index ℓ such that $S_\ell < x$ belongs to $\mathcal{S}^{c'}$. Thus, after c levels, the search space is a singleton ℓ. This means that $x \leq S_{j_0}$ in case $j_0 < n^c$. In case we return that j_0 does not exist, it holds that $\ell = n^c - 1$, and so the sum of all $s_{i,j}$ is indeed below x.

As each $s_{i,j}$ is upper bounded by n, we obtain $\sum_{i \in [n]} s_{i,j_0} \leq n^2$, and as $S_{j_0-1} = \sum_{j < j_0} \sum_{i \in [n]} s_{i,j} \leq x$, we obtain the claimed upper bound in Eq(1).

Round complexity: As all $s_{i,j}$ are upper bounded by n and t is polynomial in n, all numbers can be send in $O(\log n)$-bit messages. Each recursion level can be implemented in $O(1)$ rounds, thus we need $O(c) = O(1)$ rounds in total. $\square$

We continue with the proof of Lemma 3.6.

Proof of Lemma 3.6. Algorithm. A short pseudocode is given in Algorithm 2.

Algorithm 2 Scheduling of a round.

1: Split jobs into chunks $J_1, J_2, \ldots, J_k$.
2: **for** each chunk $J_{k'}$ **do**
3:     Apply Algorithm 1 on $J_{k'}$.

We use Claim 3.7 to split the jobs into $k = O(\lceil m^r/n^2 \rceil)$ chunks $J_1, \ldots, J_k$, such that the jobs in each chunk send at most $2n^2$ messages in round r over all of their nodes. Then, we iteratively apply Lemma 3.4 on each chunk to progress each job to the next round.

Forming chunks: First, each machine p_i, for each job j, uses $S_{i,j,r}$ to locally compute the number of messages $s_{i,j}$ that node $v_{i,j}$ sends in round r. Assume that chunks $J_1, \ldots, J_{k'-1}$ have been formed and let $\mathcal{S} = [t] \setminus (J_1 \cup \cdots \cup J_{k'-1})$. We apply Claim 3.7 with the index set $\mathcal{S}$, where machine p_i holds the sequence $(s_{i,j})_{j \in \mathcal{S}}$, and with $x = n^2$. If we find j_0, by the guarantee of Claim 3.7, we obtain that all jobs in a chunk $J_{k'}$, for $k' \neq k$, send at least n^2 messages and at most $2 \cdot n^2$ messages in round r. The jobs in chunk k send at most $2n^2$ messages. Otherwise, if we do not find j_0, the nodes of the jobs in $\mathcal{S}$ send less than $2n^2$ messages in round r, so we obtain the last chunk and set $J_k = J_{k'} = \mathcal{S}$. We thus have $k \leq \lceil m^r/n^2 \rceil$.

Executing round $r+1$: Since, by construction, the jobs in each chunk send at most $2n^2$ messages, we can iteratively apply Lemma 3.4 on the chunks.

Round complexity. We split the jobs into at most $k = O(\lceil m^r/n \rceil)$ chunks, where forming each chunk can be done in $O(1)$ rounds by Claim 3.7. The invocation of Lemma 3.4 on chunk $J_{k'}$ takes $O(\lceil M|J_{k'}|/n \rceil)$ rounds per chunk. Thus, the round complexity of the algorithm is

$$O(1) + \sum_{k'=1}^{k} O(\lceil M \cdot |J_{k'}|/n \rceil) = O\left(k + M \cdot \sum_{k'=1}^{k} |J_{k'}|/n\right) = O\left(\lceil m^r/n^2 + M \cdot t/n \rceil\right). \qquad \square$$

Finally, we use Lemmas 3.4 and 3.6 to obtain the near-optimal scheduling of Theorem 3.1.

Theorem 3.1. *There is a deterministic algorithm that schedules $t = \operatorname{poly} n$ jobs that are M-memory efficient in $O(\mathsf{GlobalCongestion} + \lceil M \cdot t/n \rceil \cdot \mathsf{dilation})$ rounds.*

Proof of Theorem 3.1. We repeatedly apply Lemma 3.6 until all jobs terminate. First, each machine p_i reads the input for each node $v_{i,j}$ for each $j \in [t]$, and executes the Computation Step of round $r = 0$, as a result of which it holds the state $S_{i,j,0}$ for each of its nodes. Then, we split the execution into epochs, where in epoch r all jobs move from the Computation Step of round r to the Computation Step of round $r + 1$. A single epoch is implemented via Lemma 3.6 in $O(\lceil m^r/n^2 + M \cdot t/n \rceil)$ rounds. After the epoch $r = \mathsf{dilation} - 1$, all machines compute the outputs given the respective terminating state $S_{i,j,\mathsf{dilation}-1}$ of each of its nodes.

Round complexity. The pre-processing in round $r = 0$ and the post-processing in the last round $r = \mathsf{dilation} - 1$ is done locally and does not require communication. Due to Lemma 3.6, the round complexity of executing round r for all jobs is $O(\lceil m^r/n^2 + M \cdot t/n \rceil)$, where m^r is the number of messages sent in round r. Since $\sum_{r=0}^{\mathsf{dilation}-1} m^r/n^2 = \mathsf{GlobalCongestion}$, we obtain the overall round complexity by

$$\sum_{r \in [\mathsf{dilation}]} O(\lceil m^r/n^2 + M \cdot t/n \rceil) = O\left(\mathsf{GlobalCongestion} + \mathsf{dilation} \cdot \lceil M \cdot t/n \rceil\right). \qquad \square$$

4 Randomized Scheduling

In this section we show and compare the two approaches for randomized scheduling: random shuffling (Section 4.1) and random delaying (Section 4.2). In contrast to Theorem 3.1, the results in this section do not require the jobs to be memory efficient.

4.1 Scheduling through Random Shuffling

In this subsection we use random shuffling to schedule I/O efficient jobs and we obtain the following theorem.

Theorem 4.1. *There is a randomized algorithm in the* CLIQUE *model that schedules $t = \text{poly}\, n$ jobs that are I/O efficient in $O(t + \mathsf{GlobalCongestion} + \mathsf{dilation} \cdot \log n)$ rounds, w.h.p.*

The definition of an I/O efficient job as used in Theorem 4.1 is as follows.

Definition 4.2 (I/O efficient job). *An* I/O efficient job *is a job where each node receives and produces at most $O(n \log n)$ bits of input and output.*

Algorithm. The high level overview of the algorithm for Theorem 4.1 (see Algorithm 3) consists of three steps: `Input Shuffling`, `Execution`, and `Output Unshuffling`.

Algorithm 3 Scheduling of I/O efficient job.

1: `Input Shuffling`
2: `Execution`: Run dilation many phases, where in phase r each machine p_i runs the protocol for its nodes $\{v_{\pi_j^{-1}(i),j} \mid j \in [t]\}$, and messages are routed via Claim 2.1.
3: `Output Unshuffling`

`Input Shuffling`: We iterate sequentially through the jobs. For each job, a leader machine, say, p_0, generates a random uniform permutation $\pi_j \colon [n] \mapsto [n]$. The permutation becomes globally known within two rounds by having p_0 send $\pi_j(i)$ to each p_i and then each p_i broadcasts $\pi_j(i)$ to all machines. In the last round of this subroutine, each machine p_i sends the input of $v_{i,j}$ to machine $p_{\pi_j(i)}$. A single round is sufficient because the job is I/O efficient. Thus, at the end, machine p_i holds the state of the nodes $v_{\pi_j^{-1}(i),j}$ for all $j \in [t]$. We call this subroutine `Input Shuffling`.

`Execution`: In dilation many phases we progress each job by one round. That is, each machine p_i performs all actions of the nodes that it holds, which are $v_{\pi_j^{-1}(i),j}$ for all $j \in [t]$. In order to use Claim 2.1 efficiently for each phase r, the machines need to compute a bound on the number of messages that any of them sends or receives in phase r. To this end, the machines jointly compute the value of $m^r = \sum_{j \in [t]} \sum_{i \in [n]} s_{i,j}^r$, where $s_{i,j}^r$ is the number of messages that node $v_{i,j}$ sends in round r. They do this by having each machine p_i send $\sum_{j \in [t]} s_{\pi_j^{-1}(i),j}^r$ to a leader machine, say, p_0, which then sums these values and broadcasts their sum m^r. That is, m^r is the total number of messages sent by all nodes in all jobs in round r, and we show that for each $i \in [n]$, $O(m^r/n + n \log n)$ is a bound on $\sum_{j \in [t]} s_{\pi_j^{-1}(i),j}^r$ ($\sum_{j \in [t]} t_{\pi_j^{-1}(i),j}^r$), which is the number of messages that machine p_i has to send (receive) in phase r, to be used when invoking Claim 2.1.

`Output Unshuffling`: At the end, after each machine executes the protocols until they finish, we use a single round of communication for each job to unshuffle the outputs according to π_j^{-1}. At the end of this `Output Unshuffling` subroutine, machine p_i holds the output $v_{i,j}$ for all $j \in [t]$. This finishes the description of the algorithm.

In the following lemma, we bound the number of messages that each machine has to send/receive in one phase by $X = O(m^r/n + n \cdot \log n)$.

Lemma 4.3. *Consider t jobs and a set of permutations $\{\pi_j\}_{j\in[t]}$ generated uniformly at random and let $S = \max_{i\in[n]} \sum_{j\in[t]} s^r_{\pi_j^{-1}(i),j}$ and $R = \max_{i\in[n]} \sum_{j\in[t]} t^r_{\pi_j^{-1}(i),j}$. Then, w.h.p., it holds that $X = \max\{S,R\} = O(m^r/n + n\log n)$, where $m^r = \sum_{i\in[n]} \sum_{j\in[t]} s^r_{i,j}$.*

Proof of Lemma 4.3. Let $c \geq 1$ be arbitrary large constant. Denote by $S^r_{i,j} = \sum_{i'\in[n]} s^r_{i',j} \cdot \mathbb{1}_{\pi_j(i')=i}$ the random variable whose value is the number of messages sent by machine p_i for job j (note that there is a single $i' = \pi_j^{-1}(i)$ for which $i = \pi_j(i')$, but this i' is also a random variable). These variables are bounded by n and are independent for different j. Denote by $S^r_i = \sum_{j\in[t]} S^r_{i,j}/n$ the random variable whose value is the total number of messages machine p_i sends normalized by n. Denote $c' = c + 2$. We show that the normalized number of messages machine p_i sends is bounded as $S^r_i \leq 3m^r/n^2 + 2c'\ln n$, with probability at least $1 - n^{c'}$.

First, we note that the expected normalized number of messages machine p_i sends is:

$$\mathrm{E}\left[\sum_{j\in[t]} S^r_{i,j}/n\right] = \sum_{j\in[t]}\sum_{i'\in[n]} s^r_{i',j} \cdot \mathrm{E}\left[\mathbb{1}_{\pi_j(i')=i}/n\right]$$
$$= \sum_{j\in[t]}\sum_{i'\in[n]} s^r_{i',j}/n^2 = m^r/n^2,$$

where the first equality holds due to the linearity of expectation, the second one holds since π_j is sampled uniformly and the last one is due to the definition of m^r.

Since for different j, the variables $S^r_{i,j}/n$ are independent, we use Claim 2.2 (Hoeffding Bound) with a relative error $\epsilon > 0$, which we later optimize, to bound the probability that a machine has too many messages to send.

$$\Pr\left[S^r_i > (1+\epsilon)\frac{m^r}{n^2}\right] = \Pr\left[\sum_{j\in[t]}\frac{S^r_{i,j}}{n} > (1+\epsilon)\frac{m^r}{n^2}\right] < e^{-\frac{\epsilon^2 m^r}{(2+\epsilon)n^2}}.$$

If $m^r \geq c' \cdot n^2 \ln n$, then for $\epsilon = 2$ we have that $e^{-\epsilon^2 m^r/((2+\epsilon)n^2)} \leq e^{-c'\ln n} = n^{-c'}$. In other words, w.h.p. $3m^r/n^2 = O(m^r/n^2)$ rounds are sufficient for machine p_i for sending all required messages on round r. Otherwise, we have $m^r < c' \cdot n^2 \ln n$. In this case, for $\epsilon = 2c' \cdot n^2 \ln(n)/m^r \geq 2$ we get that $e^{-\epsilon^2 m^r/((2+\epsilon)n^2)} = e^{-2\cdot c'\ln(n)/(2/\epsilon+1)} \leq n^{-c'}$. In other words, w.h.p. $(1 + 2c'\ln(n) \cdot n^2/m^r)m^r/n^2 = m^r/n^2 + 2c' \cdot \ln n = O(m^r/n^2 + \log n)$ rounds are sufficient for machine p_i for sending all of its required on round r. We conclude that $\Pr\left[S^r_i > 3m^r/n^2 + 2c'\ln n\right] < n^{-c'}$.

Denote by T^r_i the random variable whose value is the number of messages received by machine p_i normalized by n. By the same approach, we show that

$$\Pr\left[T^r_i > 3\frac{m^r}{n^2} + 2c'\ln n\right] < n^{-c'}.$$

By a union bound over S^r_i, T^r_j for all $i \in [n]$, we obtain that for some i one of the event $S^r_i > 3(m^r/n^2 + 2(c+2)\ln n)$, $T^r_i > 3(m^r/n^2 + 2(c+2)\ln n)$ happens with probability at most $2n^{-c'+1} \leq n^{-c}$ for $n \geq 2$. Notice, that $S = \max_{i\in[n]} S_i \cdot n$ and $R = \max_{i\in[n]} T_i \cdot n$, thus $X = \max\{S,R\} = O(m^r/n + n\log n)$ w.h.p. $\square$

With an upper bound at hand, on the number of messages that each machine sends or receives in phase r, we can prove that Algorithm 3 satisfies the statement of Theorem 4.1.

15

Proof of Theorem 4.1. We prove the correctness and bound the runtime of the presented algorithm (see Algorithm 3).

Correctness: After the `Input Shuffling` subroutine (Line 1), the input for node $v_{i,j}$ is stored on machine $p_{\pi_j(i)}$. For each phase $r \in [\mathsf{dilation}]$, we invoke Claim 2.1 with the computed value X, which is w.h.p. a bound the number of messages that each machine sends or receives. Thus, w.h.p. this invocation succeeds. Since $\mathsf{dilation} = O(n)$, a union bound over all phases gives that at the end of the `Execution` subroutine, each machine p_i holds the outputs of all nodes $v_{\pi^{-1}(i),j}$ for each $j \in [t]$. After `Output Unshuffling`, machine p_i holds the output for node $v_{i,j}$ for each job $j \in [t]$.

Round Complexity: The initial `Input Shuffling` (Line 1) and the `Output Unshuffling` at the end of the algorithm (Line 3) complete with t rounds each. For each phase r in the `Execution` part of the algorithm, computing m^r is done in 2 rounds. By Lemma 4.3, $X = O(m^r/n + n \log n)$ is a bound on $\sum_{j \in [t]} s^r_{\pi^{-1}(i),j}$ and $\sum_{j \in [t]} t^r_{\pi^{-1}(i),j}$, which are the number of messages that machine p_i sends and receives in phase r, respectively, for all $i \in [n]$. Thus, invoking Claim 2.1 completes in $O(m^r/n^2 + \log n)$ rounds, w.h.p. Thus, the overall round complexity of the algorithm is

$$O\left(t + \sum_{r \in [\mathsf{dilation}]} (m^r/n^2 + \log n)\right) = O\left(t + \sum_{j \in [t]} m_j/n^2 + \mathsf{dilation} \cdot \log n\right)$$

$$= O(t + \mathsf{GlobalCongestion} + \mathsf{dilation} \cdot \log n). \qquad \square$$

4.2 Scheduling through Random Delays

In this subsection we show how to use random delays approach introduced in [LMR94] to schedule round efficient jobs.

Theorem 4.4. *There is a randomized algorithm in the* CLIQUE *model, which schedules t jobs*

$$O(\mathsf{LocalCongestion} + \mathsf{dilation} \cdot \log n + t/n)$$

rounds, w.h.p., given an upper bound on the value of LocalCongestion, *and in*

$$O\big(\mathsf{LocalCongestion} + \log \mathsf{LocalCongestion} \cdot (\mathsf{dilation} \cdot \log n + t/n)\big)$$

rounds, w.h.p., if such a bound is not known.

In the algorithm, job $j \in [t]$ is executed with a delay D_j that is chosen uniformly at random from $[D]$, where $D = \lfloor \mathsf{LocalCongestion}/\ln n \rfloor$. In the crucial step of the proof, we use a Hoeffding Bound to show that this random delay implies that each node has to send and receive at most $X = O(\mathsf{LocalCongestion} \cdot n/D)$ messages per round in all jobs combined. The claim then follows by routing all messages of a single round with Lenzen's routing scheme (Claim 2.1). This approach uses that all nodes know a bound on LocalCongestion, which can be removed at the cost of a logarithmic factor with a standard *doubling*-technique.

Algorithm: We describe the algorithm for the case where LocalCongestion is known. The algorithm consists of initializing part `Sample Delays`, followed by the actual `Execution` part. Let $D = \lfloor \mathsf{LocalCongestion}/\ln n \rfloor$.

`Sample Delays`: We start by generating a random delay D_j for each job j and broadcasting it. For this, a leader node, say, p_0, samples a delay D_j uniformly at random from $[D]$ independently for each job j. Notice, that in the special case $D \le 1$ (which happens when $\mathsf{LocalCongestion} < 2 \ln n$), the delays are actually degenerated to the deterministic $D_j = 0$. `Execution` ($O(D +$

dilation) `phases`): In *phase* r we progress each job j (for which $r \geq D_j$ holds) from round $r - D_j$ to round $r - D_j + 1$. Each machine p_i executes the protocol of round $r - D_j$ for job j. To deliver the messages efficiently, we use the algorithm from Claim 2.1, which requires the bound X on $\max_{i \in [n]} \{ \sum_{j \in [n]} s_{i,j}^{r-D_j}, \sum_{j \in [n]} t_{i,j}^{r-D_j} \}$, the number of messages machine sends or receives. If $\mathsf{LocalCongestion} < 2 \ln n$, the number of messages to send or receive is clearly bounded by $O(n \log n)$. In the general case, we show that this bound is $O(\mathsf{LocalCongestion} \cdot n / D)$ w.h.p.

   `Doubling`: To remove the requirement on the knowledge of $\mathsf{LocalCongestion}$, we use a standard doubling technique. We try to run the algorithm until success while doubling the estimation of $\mathsf{LocalCongestion}$ in each attempt, starting from a guess of $\mathsf{LocalCongestion} = 1$. The algorithm detects failure when the algorithm from Claim 2.1 fails.

Algorithm 4 Scheduling of jobs.

 1: `Sample delays`: Independently UAR pick $D_j \in [D]$ and broadcast the values
 2: `Execution` : Run $O(D + \mathsf{dilation})$ phases, where in phase r progress each job j that satisfies $r \geq D_j$ by one round where the messages of all jobs are routed with Claim 2.1.

In the proof of the following lemma, we bound the number of messages that each machine has to send/receive in one phase by $X = O(\mathsf{LocalCongestion} \cdot n / D)$.

Lemma 4.5. *Given* t *jobs and a set of delays* $\{D_j\}_{j \in [t]}$ *sampled uniformly at random from* $[D]$ *for* $D = \lfloor \mathsf{LocalCongestion} / \ln n \rfloor \geq 1$, *let* $S = \max_{i \in [n]} \sum_{j \in [t]:\, r \geq D_j} s_{i,j}^{r-D_j}$, $R = \max_{i \in [n]} \sum_{j \in [t]:\, r \geq D_j} t_{i,j}^{r-D_j}$, *and* $X = \max \{ S, R \}$.

   Then, w.h.p., it holds that $X = O(\mathsf{LocalCongestion} \cdot n / D)$, *where* $m^r = \sum_{i \in [n]} \sum_{j \in [t]} s_{i,j}^r$.

Proof. Let $c \geq 1$ be arbitrary large constant. Denote by $S_{i,j} = \sum_{r' \in [\mathsf{dilation}]} s_{i,j}^{r'} \cdot \mathbb{1}_{D_j + r' = r}$ the random variable whose value is the number of messages sent by machine p_i for job j on round r. These variables are independent for different values of j, as the delays D_j are independent. They are also bounded by n, which means that the variables $S_{i,j}/n$ are also independent and belong to $[0, 1]$. Denote by $S_i = \sum_{j \in [t]} S_{i,j}/n$ the random variable whose value is the number of messages sent by machine p_i, normalized by n. Denote $c' = c + 2$. We show that the normalized number of messages machine p_i sends is bounded as $S_i^r \leq (1 + 2 \cdot c')\mathsf{LocalCongestion}/D$ with probability at least $1 - n^{c'}$.

   First, we note that the expected normalized number of messages machine p_i sends is:

$$\mathrm{E}[S_i] = \mathrm{E}\left[\sum_{j \in [t]} S_{i,j}/n \right] = \frac{\sum_{j \in [t]} \sum_{r' \in [\mathsf{dilation}]} s_{i,j}^{r'} \cdot \mathrm{E}[\mathbb{1}_{D_j + r' = r}]}{n}$$

$$\leq \frac{\sum_{j \in [n]} \sum_{r' \in [\mathsf{dilation}]} s_{i,j}^{r'}}{D \cdot n} = \frac{\mathsf{LocalCongestion}}{D},$$

where the second transition is due to the linearity of expectation, the third follows from delays being uniformly selected and the last one is due to the definition of $\mathsf{LocalCongestion}$.

   Since D_j are independent, we use Claim 2.2 (Hoeffding Bound) with $\epsilon = 2 \cdot c'$, we bound the

probability of S_i being larger than the expected value by

$$\Pr[S_i \geq (1 + 2 \cdot c')(\mathsf{LocalCongestion}/D)] = \Pr[\sum_{j \in [t]} (S_{i,j}/n) \geq (1 + 2 \cdot c')(\mathsf{LocalCongestion}/D)]$$

$$\leq e^{-\frac{(2 \cdot c')^2}{2 + 2 \cdot c'} \frac{\mathsf{LocalCongestion}}{D}} = e^{-\frac{4c'^2}{2 + 2c'} \ln n} \leq n^{-c'},$$

where the second transition is due to Claim 2.2 and the third is due to the selection of $D \leq \frac{\mathsf{LocalCongestion}}{\ln n}$.

Denote by T_i the random variable whose value is the number of messages received by machine p_i on round r normalized by n. Using a similar approach, it holds that

$$\Pr\left[T_i \geq (1 + 2 \cdot c')\mathsf{LocalCongestion}/D\right] \leq n^{-c'}.$$

By a union bound over all S_i and T_i, we obtain that for some i, the probability that S_i or T_i are more than $\mathsf{LocalCongestion}/X$ is bounded by n^{-c}. Since $S = \max_{i \in [n]} S_i^r$, $R = \max_{i \in [n]} T_i^r$, and $X = \max\{S, R\}$ it w.h.p. holds that $X = O(\mathsf{LocalCongestion} \cdot n/D)$. $\qquad\square$

The following simple routing primitives are used in the random-delay based algorithm of Theorem 4.4.

Definition 4.6. *(Multiple broadcast problem.) Each machine $p_i \in V$ is given a set M_i of m_i messages of size $O(\log n)$ bits each. The goal is to deliver each message to all the machines.*

Lemma 4.7. *There is an algorithm in the CLIQUE model, which solves the multiple broadcast problem in $O\big(\lceil \sum_{i \in [n]} m_i/n \rceil\big)$ rounds.*

Proof. The pseudocode is given in Algorithm 5. First, on Line 1, each machine p_i broadcasts m_i, the number of messages it has. Given the information it receives, the machine p_i locally computes $y_i = \sum_{i'=0}^{i-1} m_{i'}$, the number of messages the machines with preceding identifiers $i' < i$ have. This allows each machine to compute indices of its messages in the global numbering. We split the execution into $\lceil \sum_{i \in [n]} m_i/n \rceil = \lceil y_n/n \rceil$ phases. On phase k, a batch of messages with indices $[k \cdot n, \min\{(k+1) \cdot n, \sum_{i \in [n]} m_i\})$ are broadcasted in two rounds. In the first round, the i'-th message of the current batch (e.g. the message number $k \cdot n + i'$) is sent to machine $p_{i'}$ (Line 4). In the second round, each machine broadcasts the message it received in the previous round (Line 5).

Algorithm 5 Multiple broadcasts.

1: Each machine p_i broadcasts m_i.
2: Each machine p_i locally computes its $y_i = \sum_{i'=0}^{i-1} m_{i'}$.
3: **for** $k \leftarrow 0$ to $\lfloor y_n/n \rfloor$ **do**
4:      For each $i' \in [n]$ message number $k \cdot n + i'$ in global numbering is sent to the machine $p_{i'}$.
5:      Each $p_{i'}$ broadcasts the message it receives.

In the first round of each phase, at most one message is received by each machine, in particular only 1 message between any pair of machines. In the second round of each phase, each machine sends at most 1 message to each other machine. Hence, the entire execution completes in $O(\lceil \sum_{i=0}^{n-1} m_i/n \rceil)$ rounds. $\qquad\square$

Proof of Theorem 4.4. We prove the correctness and bound the runtime for the aforementioned algorithm (Algorithm 4).

First, in the special case $\mathsf{LocalCongestion} < 2 \ln n$, the number of messages each machine has to send over the entire execution for all jobs combined and in particular in each round is bounded by $2 \cdot n \ln n$. Thus, a straightforward execution of one round of all jobs with Claim 2.1 completes in $2 \ln n$ rounds, and the entire execution takes $O(\mathsf{dilation} \cdot \log n)$ rounds. From now on we assume $D \geq 2$.

Correctness. In each phase $r \in [D + \mathsf{dilation}]$, we invoke Claim 2.1 with a bound of $X = O(\mathsf{LocalCongestion}/D)$, which due to Lemma 4.5 bounds w.h.p. the number of messages each node sends or receives. Thus, due to the union bound over n rounds, all of them succeed w.h.p.

Round complexity. Broadcasting t values during `Sample Delay` (Line 1) takes $O(\lceil t/n \rceil)$ rounds by Lemma 4.7. For each phase $r \in [D + \mathsf{dilation}]$, by Lemma 4.5 $X = O(\mathsf{LocalCongestion} \cdot n/D)$ is a bound on the number of messages that machine p_i sends and receives in phase r for each $i \in [n]$ w.h.p. and by applying union bound over the $D + \mathsf{dilation} = O(\mathrm{poly}\,(n))$ rounds, this holds on each round w.h.p. Thus, invoking the algorithm from Claim 2.1 completes in $O(\lceil \mathsf{LocalCongestion}/D \rceil) = O(\mathsf{LocalCongestion}/D)$. Thus, overall, for $D = \lfloor \mathsf{LocalCongestion}/\ln n \rfloor$ the algorithm terminates in $O(\lceil t/n \rceil) + (\mathsf{dilation} + D) \cdot O(\mathsf{LocalCongestion}/D) = O(t/n + \mathsf{dilation} \log n + \mathsf{LocalCongestion})$ rounds w.h.p.

Doubling. Since the algorithm succeeds w.h.p. when our estimate is at least equal to the value of $\mathsf{LocalCongestion}$, we finish within $O(\log \mathsf{LocalCongestion})$ attempts. Thus, w.h.p., the round complexity of this approach is $\sum_{\kappa=0}^{O(\log \mathsf{LocalCongestion})} O(t/n + 2^\kappa + \mathsf{dilation} \cdot \log n) = O(\mathsf{LocalCongestion} + \log \mathsf{LocalCongestion} \cdot (t/n + \mathsf{dilation} \cdot \log n))$. □

5 Applications: MIS & Pointer Jumping

In this section we apply the scheduling algorithms developed in Sections 3 and 4 on protocols which solve MIS (Section 5.1) and Pointer Jumping (Section 5.2). We analyze the round complexity of the developed algorithms.

5.1 Maximal Independent Set

A *maximal independent set (MIS)* of a graph $G = (V, E)$ is a subset of nodes $M \subseteq V$ such that no two nodes in M are connected by an edge and adding any node to M would break this property. In this subsection, we show that we can efficiently solve multiple MIS instances using our scheduling algorithm from Theorem 4.1.

Theorem 5.1 (Multiple MIS instances)**.** *There is a randomized algorithm in the* CLIQUE *model which solves $t = \mathrm{poly}\,n$ instances of MIS in $O(t + \log \log \Delta \log n)$ rounds, w.h.p.*

To prove our result, we prove that the MIS protocol for the CLIQUE model given in [GGK$^+$18], which completes in $O(\log \log \Delta)$ rounds, uses $O(n^2)$ messages in all rounds combined, which we state as follows.

Theorem 5.2 (Analysis of the MIS protocol of [GGK$^+$18, Theorem 1.1])**.** *There is a randomized MIS protocol in the* CLIQUE $+$ Lenzen's Routing *model which completes in $O(\log \log \Delta)$ rounds and sends $O(n^2)$ messages, w.h.p.*

Given Theorem 5.2, we prove Theorem 5.1 as follows.

Proof of Theorem 5.1. By Theorem 5.2, a set of t jobs of the MIS protocol of [GGK$^+$18] have dilation $= O(\log \log \Delta)$ and $\mathsf{GlobalCongestion} = \frac{t \cdot n^2}{n^2} = t$, w.h.p. By Theorem 4.1, w.h.p., we can schedule the t jobs in a number of rounds bounded by $O(t + \mathsf{GlobalCongestion} + \mathsf{dilation} \cdot \log n) = O(t + \log \log \Delta \log n)$. $\qquad\square$

Remark. Theorem 5.1 also shows that the random-shuffling approach may be more efficient than random-delays. In the MIS protocol of [GGK$^+$18] which we use here, a leader node is used for collecting some of the edges of the graph. Potentially, since the leader node may receive $O(n)$ messages during the $O(\log \log \Delta)$ rounds of the protocol, applying the random-delay scheduling of Theorem 4.4 on t such MIS jobs results in a complexity of $O(t \log \log \Delta + \log \log \Delta \log n)$ rounds. This run-time is asymptotically worse than the one obtained by the algorithm from Theorem 5.1 for $t = \Omega(\log n)$. Moreover, it is no better then the naïve execution of the protocol multiple times one after another.

It remains to prove Theorem 5.2.

Proof of Theorem 5.2. The correctness and the round complexity follow from [GGK$^+$18]. We analyze the message complexity of the protocol. For this we must describe the protocol, which returns a set $M \subseteq V$, initially empty.

The Protocol (See Algorithm 6).

Random ranking: First, a leader node v^* generates a uniform random permutation $\pi \colon [n] \mapsto [n]$ and makes it globally known within 2 rounds by sending each node v_i the value of $\pi(i)$ which v_i then broadcasts to everyone. The value $\pi(i)$ is called the *rank* of v_i and does not change during the algorithm.

Degree reduction by simulating greedy steps: The second part of the protocol is a loop, which, as shown in [GGK$^+$18, Theorem 1.1], uses $O(\log \log \Delta)$ iterations w.h.p., to reduce the maximum degree of *active* nodes to $\Delta' = \min \{ \Delta, \mathrm{poly} \log n \}$.

Algorithm 6 The MIS algorithm of [GGK+18].

1: The leader v^* generates a uniform random permutation $\pi\colon [n] \mapsto [n]$ and sends $\pi(i)$ to v_i.
2: Each node v_i broadcasts $\pi(i)$.
3: $k \leftarrow 0$
4: **while** The maximum active degree of active nodes is at least $\Delta' = \min\{\Delta, \operatorname{poly}\log n\}$ **do**
5:      $M_0 \leftarrow \emptyset$, $M_k \leftarrow M_{k-1}$ if $k \geq 1$
6:      $N_0 \leftarrow \emptyset$, $N_k \leftarrow N_{k-1}$ if $k \geq 1$
7:      Every edge $\{v_i, v_{i'}\}$ with both endpoints active and $\pi(i) \leq \pi(i') \leq \frac{n}{\Delta^{\alpha^k}}$ is sent to v^* by v_i.
8:      **while** There exists a node $v_i \notin M_k \cup N_k$ with $\pi(i) \leq \frac{n}{\Delta^{\alpha^k}}$ **do**
9:          Add v_i with the smallest rank $\pi(i)$ among the undecided nodes to M_k.
10:          All the neighbors of v_i that are known to v^* are added to N_k.
11:      The leader v^* informs the nodes in $M_k \setminus M_{k-1}$ that they are such.
12:      The nodes in $M_k \setminus M_{k-1}$ are added to M, and they inform their neighbors that they are such and become inactive.
13:      The nodes in $N_G(M_k \setminus M_{k-1})$ inform their neighbors that they are such and become inactive.
14:      $k \leftarrow k + 1$.
15: **for** k from 0 to $O(\log\log \Delta')$ **do**
16:      Each active node v_i sends all adjacent edges from H^{2^k} to each of its neighbors in H^{2^k}.
17: Each active node v_i simulates $O(\log \Delta')$ rounds of the MIS protocol of [Gha16] locally. The chosen nodes are added to M and they become inactive along with their neighbors.
18: The leader v^* learns all remaining edges, locally computes an MIS over them and informs the nodes, which are then added to M.

In each iteration $k \geq 0$, we produce a set $M_k \subseteq V$, which is initially empty for $k = 0$ and is initially M_{k-1} for $k \geq 1$. The nodes in M_k are afterwards added to the resulting MIS, M. We also use a set N_k which is initially empty for $k = 0$ and is initially N_{k-1} for $k \geq 1$, of nodes that will not be in M. Initially, all nodes are *active*. A node that is in $M_k \cup N_k$ is *decided* and becomes *inactive*, and otherwise it remains *active*. A constant $\alpha = 3/4$ is set.

In each iteration k, all edges $\{v_i, v_{i'}\}$ where both endpoints are active and have ranks $\pi(i) \leq \pi(i') \leq n/\Delta^{(\alpha^k)}$ are sent to the leader v^* by v_i. The leader v^* now applies greedy MIS steps, as follows. As long as there is an active node v_i with $\pi(i) \leq n/\Delta^{(\alpha^k)}$, the node with the smallest rank is added to M_k and all of its neighbors that are known to v^* are added to N_k. After these greedy steps, the leader v^* informs the nodes in M_k that they are such. These nodes are added to M and become *inactive*, and they inform their neighbors, which join N_k and become inactive as well.

The loop terminates when the maximum degree of active nodes is at most $\Delta' = \min\{\Delta, \operatorname{poly}\log n\}$. To check that this condition is met, each node sends its degree to the leader and the leader broadcasts the decision. This requires 1 round. We denote by $H = (V', E')$ the graph of maximum degree bounded by Δ' that is induced by the remaining active nodes.

Small degrees (the graph H): First, each active node generates $O(\log^2 \Delta')$ random bits. These random bits are from now sent along with the node's identifier whenever the latter is sent in a message. Then, each node of H learns its $O(\log \Delta')$-hop neighborhood in H. To this end, we proceed in $O(\log\log \Delta')$ iterations, where after iteration k, each node in H knows its 2^k-hop neighborhood in H. In iteration k, each node sends its edges in H^{2^k} to its neighbors in H^{2^k}. Notice that by induction over k, at the beginning of iteration k, each node knows its neighbors in H^{2^k}, and at the end of the iteration, it knows its neighbors in $H^{2^{k+1}}$.

After learning its $O(\log \Delta')$-hop neighborhood, each active node locally simulates $O(\log \Delta')$ rounds of the randomized MIS protocol of [Gha16]. Each iteration of this protocol requires $O(\log \Delta')$ random bits by each node, which are the ones generated by the node at the beginning of this step. Each node chosen to the MIS is added to M and becomes inactive along with its neighbors. Notice that all nodes compute the same MIS locally, because each node knows a sufficiently large neighborhood, including $O(\log^2 \Delta')$ globally consistent random bits for each node in the neighborhood.

Wrapping-up part: Finally, the leader v^* learns the remaining graph induced by active nodes, and locally computes an MIS and informs the nodes, who are then added to M. This finishes the description of the algorithm.

Message Complexity.

Random ranking: Since this part takes 2 rounds, it clearly sends at most $O(n^2)$ messages.

Degree reduction (simulation greedy steps): Let G_0 be the subgraph induced by nodes with ranks $\pi(v_i) \leq \frac{n}{\Delta}$. Since the maximum degree in G is Δ, the number of edges in G_0 is bounded by $\frac{n}{\Delta} \cdot \Delta = n$. This implies that at most $O(n)$ messages are sent to the leader v^* in the first iteration.

For $k \geq 1$, let $r_k = n/\Delta^{(\alpha^k)}$, and let $G_k = (V_k, E_k)$ be the subgraph that is induced by nodes with ranks in the range $[r_{k-1}, r_k]$ that are still active after iteration $k - 1$. In [GGK+18, Theorem 1.1], it is shown that G_k has at most $O(n)$ edges, w.h.p., which implies that at most $O(n)$ messages are sent to the leader v^* in iteration k. Informing the nodes in $M_k \setminus M_{k-1}$ that they should join M_k requires at most $O(n)$ messages. Notice, that the leader does not inform nodes in $N_k \setminus N_{k-1}$, as they informed by their neighbors in $M_k \setminus M_{k-1}$. Checking the loop condition required $O(n)$ messages. Since [GGK+18, Lemma 3.1] implies that after $O(\log \log \Delta)$ iterations of the loop, the degree in the graph induced by active nodes is at most $n \log n/(n/\Delta^{\alpha^{O(\log \log \Delta)}}) = \text{poly} \log n = \Delta'$ and the loop terminates, this gives a total of $O(n \log \log \Delta)$ of such messages, w.h.p. Over the entire execution of the protocol, each node v_i is informed at most $\deg_G(v_i)$ times by one of its neighbors that such a neighbor enters the MIS or becomes inactive. Thus, over the entire course of the algorithm this requires $O(n^2)$ messages.

Small degrees: For $k = O(\log \log \Delta')$ the maximum degree in $H^{2^k} = H^{\text{poly} \log \Delta'}$ is bounded by $\Delta'^{\text{poly} \log \Delta'}$. To send one identifier together with $O(\log^2 \Delta')$ random bits we need $1 + O(\log^2 \Delta'/\log n)$ $O(\log n)$-bit messages. Thus, for each k, each active node sends at most

$$O\big((1 + \log^2 \Delta'/\log n)\big)\Delta'^{\text{poly} \log \Delta'} = 2^{\text{poly} \log \Delta'}$$

messages. For the entire $O(\log \log \Delta')$ rounds, each active node sends $O(\log \log \Delta)' \cdot 2^{\text{poly} \log \Delta'} = 2^{\text{poly} \log \Delta'}$ messages. As $\Delta' = \text{poly} \log n$ holds, the number of messages sent by each active node to learn its $O(\log \Delta')$-hop neighborhood is $2^{\text{poly} \log \log n} = O(n)$. This implies $O(n^2)$ messages in total. The simulation of [Gha16] to decide whether to join M is then done locally, without communication.

Wrapping-up part: In [Gha17, Lemma 2.11], it is shown that the graph induced by active nodes after learning $O(\log \Delta')$-hop neighborhoods in H and simulating $O(\log \Delta')$ iterations of the MIS algorithm from [Gha16] has at most $O(n)$ edges. Thus, learning the remaining edges by the leader and informing nodes about the leader's decision requires $O(n)$ messages. Notice that in the CLIQUE model this would require some routing scheme. However, in the CLIQUE + Lenzen's Routing model this is part of the model definition.

Thus, overall the algorithm sends $O(n^2)$ messages. □

5.2 Pointer Jumping

In this subsection we address the pointer jumping problem, widely used in parallel and distributed data structures [Hir76].

Definition 5.3 (*P*-pointer jumping)**.** *In a P-pointer jumping problem, each node v_i is given a permutation $\pi_i \colon [P] \mapsto [P]$. A fixed node $v_{i'}$ is given a number $x \in [P]$, The aim of the algorithm is for $v_{i'}$ to learn the composition of the permutations applied on p, i.e., $(\pi_{n-1} \circ \pi_{n-2} \circ \cdots \circ \pi_0)(p) = \pi_{n-1}(\pi_{n-2}(\ldots \pi_0(p)\ldots))$*

In the following claim we show a simple deterministic $O(\log n)$-round CLIQUE protocol for solving *P*-pointer jumping with a complexity of $O(P \cdot n)$ messages.

Claim 5.4 (Pointer jumping)**.** *For $P = O(n)$, there is a deterministic $O(P)$-memory efficient protocol in the CLIQUE + Lenzen's Routing model which solves the pointer jumping problem in $O(\log n)$ rounds and $O(P \cdot n)$ messages.*

Algorithm 7 Pointer jumping.

---

   for $k = 0$ to $\lceil \log n \rceil$ **do**
      for $i \in [n]$ in parallel **do**
         if i has at least $k > 0$ trailing zeros in binary representation **then**
            v_i computes $\pi_i \circ \pi_{i+1} \circ \cdots \circ \pi_{\min\{i+2^k,n\}-1}$.
         if i has exactly $k < \lceil \log n \rceil$ trailing zeros in binary representation **then**
            v_i sends $\pi_i \circ \pi_{i+1} \circ \cdots \circ \pi_{\min\{i+2^k,n\}-1}$ to v_{i-2^k}.
   $v_{i'}$ sends p to v_0.
   v_0 replies $v_{i'}$ with $(\pi_{n-1} \circ \pi_{n-2} \circ \cdots \circ \pi_0)(p)$.

Proof of Claim 5.4.

    Algorithm and correctness. The pseudo-code for the simple well known protocol for the pointer jumping problem is presented in Algorithm 7. At a high level, first v_0 learns the composition of the permutations, then $v_{i'}$ sends the entry p to v_0 and it responds with the final output. To learn the composition of the permutations we proceed in $\lceil \log n \rceil + 1$ iterations. On each iteration except the first, each node v_i which receives a permutation from $v_{i+2^{k-1}}$, computes the composition of the permutation it possesses and the received permutation, that is, it composes the permutation $\pi_i \circ \pi_{i+1} \circ \cdots \circ \pi_{i+2^{k-1}-1}$ with the permutation $\pi_{i+2^{k-1}} \circ \cdots \circ \pi_{i+2^k-1}$. Each node v_i which has exactly k trailing zeros in the identifier and currently knows the composition of 2^k permutations $\pi_i \circ \pi_{i+1} \circ \cdots \circ \pi_{\min\{i+2^k,n\}-1}$ sends it to the node v_{i-2^k}. Each node sends and receives at most $P = O(n)$ messages. Clearly, after $\lceil \log n \rceil$ iterations, node v_0 possesses the composition $\pi_0 \circ \pi_1 \circ \cdots \circ \pi_{n-1}$.

    Memory-efficiency. To compose the received permutation with the current permutation, we store both permutations and the output permutation in the local memory. Thus we require $O(P \log n)$ bits of the local memory[3] . Given only the index of the round, it is possible for each node to deduce the number of messages each node sends to it.

---

[3]In the CLIQUE model this algorithm requires some routing scheme, but Claim 2.1 is not known to run in $O(P \log n)$ bits of memory. However, our results apply in the potentially more powerful model of CLIQUE + Lenzen's Routing [KS20], in which each node is allowed to send and receive n messages in each round. Therefore, no additional memory overhead of routing algorithm is required.

Round complexity. The algorithm finishes within $O(\log n)$ iterations. In each iteration, each node sends and receives $P = O(n)$ messages, thus each iteration completes in $O(1)$ rounds.

Message complexity. In k-th iteration of the algorithm, $O(n/2^k)$ nodes send P messages each. Thus, the protocol uses $\sum_{k=0}^{\lceil \log n \rceil - 1} O(n/2^k) \cdot O(P) = O(P \cdot n)$ messages. □

Applying our deterministic scheduling algorithm and our random shuffling algorithm, we obtain the following theorem on the complexity of solving multiple instances of the pointer jumping problem.

Theorem 5.5 (Pointer Jumping). *For $P \leq n$, there are algorithms in the CLIQUE model that solve $t = \text{poly } n$ instances of the P-pointer jumping problem deterministically in $O(\lceil P \cdot t/n \rceil \cdot \log n)$, and randomized in $O(t + \log^2 n)$ rounds, w.h.p.*

Proof of Theorem 5.5. The first part of the theorem follows immediately from Claim 5.4 and Theorem 3.1. By Theorem 3.1, running t instances of the protocol from Claim 5.4 completes in $O(t \cdot P \cdot n/n^2 + \lceil P \cdot t/n \rceil \cdot \log n) = O(\lceil P \cdot t/n \rceil \cdot \log n)$ rounds. This gives the first claim.

Since each node of the job consumes only $P \log n \leq n \log n$ bits of input and produces $\log n$ bits of output, it is P I/O efficient. Thus, by Theorem 4.1, running t instances of the protocol from Claim 5.4 completes in $O(t + t \cdot P \cdot n/n^2 + \log n \cdot \log n) = O(t + \log^2 n)$ rounds, w.h.p., which gives the second claim. □

The proposed simple $O(\log n)$ round pointer jumping protocol also serves as an example where scheduling jobs via the random-shuffling approach of Theorem 4.1 is significantly better than the random-delay based approach of Theorem 4.4. Since multiple nodes receive $\Omega(n \log n)$ messages in the execution of the protocol, if we apply the random-delay scheduling algorithm from Theorem 4.4 we solve t instances of the problem in $O(t \cdot n \cdot \log n + \log^2 n)$ rounds, which is no better than sequentially running one instance after another.

6 Discussion

Our results suggest that the amortized complexity, i.e., the runtime of solving many instances of a problem divided by the number of instances, is a valuable measure for the efficiency of protocols in the CLIQUE model. Our interest in obtaining protocols with fast amortized complexities stems from the growing number of problems which admit $O(1)$-round CLIQUE-protocols, e.g., [CDP20, Now19, GNT20], whose amortized complexity could potentially be shown to go below constant, as well as from problems that are still not known to have a constant worst-case complexity. We now elaborate on this viewpoint.

We give MIS as an example of a problem which can be solved with a good amortized complexity. The best known protocol [GGK+18] requires $O(\log \log \Delta)$ rounds. Theorem 5.1 shows that running $t = \text{poly } n$ instances of MIS completes in $O(t + \log \log \Delta \log n)$ rounds. For $t = \Omega(\log \log \Delta \log n)$, the second part of the complexity "amortizes out" and we obtain that we run t instances of the MIS problem in $O(t)$ rounds. Basically, we show that the amortized complexity of the MIS problem is $O(1)$ rounds.

Note that the amortized complexity should not be optimized isolated from other measures. For example, consider the trivial $O(n)$-round protocol for pointer jumping, in which in the i-th round, the i-th node applies its permutation to the *current pointer* and sends the result to the next node. It requires only $O(n)$ messages. Thus, it is trivial to run $t \leq n^2$ instances of this pointer jumping

protocol in only $O(n)$ rounds, leading to an amortized complexity of $O(1/n) = o(1)$. However, the *latency* of this algorithm is an unacceptable $O(n)$ rounds. Instead, Theorem 5.5 shows that the pointer jumping problem has an acceptable amortized complexity of $O(1)$ rounds and a small latency of $O(\log^2 n)$ rounds.

For certain protocols, Theorem 3.1 might even yield $o(1)$ amortized complexity. For example, consider a job in which it is required to compute the $\sqrt{n}$-bin histogram of some given data. In the trivial 2-round protocol, each node locally builds a histogram of its input and sends the number of elements in its i-th bin to v_i. For all $i \in [\sqrt{n}]$, node v_i sums the received values and broadcasts the result. Clearly, such an algorithm is $O(\sqrt{n})$-memory efficient and uses $O(n\sqrt{n})$ messages. Our algorithm from Theorem 3.1 executes t instances of this protocol in $O(\lceil t/\sqrt{n} \rceil)$ rounds. Whenever $t = o(\sqrt{n})$, this gives an $o(1)$ amortized round complexity with constant latency.

The reader may notice that for some sets of jobs, it may be that some ad-hoc routing could be developed for efficient scheduling. We emphasize that, in contrast, the power of our algorithms is that they *do not* require tailoring the protocols for the sake of scheduling them within a given set of jobs. This is pivotal for obtaining a general framework, because knowing in advance the setting in which a protocol would be executed is an unreasonable assumption that we do not wish to make.

References

[CDKL19] Keren Censor-Hillel, Michal Dory, Janne H. Korhonen, and Dean Leitersdorf. Fast approximate shortest paths in the congested clique. In *PODC*, pages 74–83. ACM, 2019.

[CDP20] Artur Czumaj, Peter Davies, and Merav Parter. Simple, deterministic, constant-round coloring in the congested clique. In *PODC*, pages 309–318. ACM, 2020.

[CGL20] Keren Censor-Hillel, François Le Gall, and Dean Leitersdorf. On distributed listing of cliques. In *PODC*, pages 474–482. ACM, 2020.

[CKK$^+$19] Keren Censor-Hillel, Petteri Kaski, Janne H. Korhonen, Christoph Lenzen, Ami Paz, and Jukka Suomela. Algebraic methods in the congested clique. *Distributed Comput.*, 32(6):461–478, 2019.

[CPS20] Keren Censor-Hillel, Merav Parter, and Gregory Schwartzman. Derandomizing local distributed algorithms under bandwidth restrictions. *Distributed Comput.*, 33(3-4):349–366, 2020.

[CPZ19] Yi-Jun Chang, Seth Pettie, and Hengjie Zhang. Distributed triangle detection via expander decomposition. In *SODA*, pages 821–840. SIAM, 2019.

[DLP12] Danny Dolev, Christoph Lenzen, and Shir Peled. "tri, tri again": Finding triangles and small subgraphs in a distributed setting - (extended abstract). In *DISC*, pages 195–209, 2012.

[Gal16] François Le Gall. Further algebraic algorithms in the congested clique model and applications to graph-theoretic problems. In *DISC*, pages 57–70, 2016.

[GGK+18]   Mohsen Ghaffari, Themis Gouleakis, Christian Konrad, Slobodan Mitrovic, and Ronitt Rubinfeld. Improved massively parallel computation algorithms for MIS, matching, and vertex cover. In *PODC*, pages 129–138. ACM, 2018.

[Gha15]   Mohsen Ghaffari. Near-optimal scheduling of distributed algorithms. In *PODC*, pages 3–12. ACM, 2015.

[Gha16]   Mohsen Ghaffari. An improved distributed algorithm for maximal independent set. In *SODA*, pages 270–277. SIAM, 2016.

[Gha17]   Mohsen Ghaffari. Distributed MIS via all-to-all communication. In *PODC*, pages 141–149. ACM, 2017.

[GN18]   Mohsen Ghaffari and Krzysztof Nowicki. Congested clique algorithms for the minimum cut problem. In *PODC*, pages 357–366. ACM, 2018.

[GNT20]   Mohsen Ghaffari, Krzysztof Nowicki, and Mikkel Thorup. Faster algorithms for edge connectivity via random 2-out contractions. In *SODA*, pages 1260–1279. SIAM, 2020.

[GP16]   Mohsen Ghaffari and Merav Parter. MST in log-star rounds of congested clique. In *PODC*, pages 19–28. ACM, 2016.

[Hir76]   Daniel S. Hirschberg. Parallel algorithms for the transitive closure and the connected component problems. In *STOC*, pages 55–57. ACM, 1976.

[Hoe63]   Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *J. Am. Stat. Assoc.*, 58(301):13–30, 1963.

[HPP+15]   James W. Hegeman, Gopal Pandurangan, Sriram V. Pemmaraju, Vivek B. Sardeshmukh, and Michele Scquizzato. Toward optimal bounds in the congested clique: Graph connectivity and MST. In *PODC*, pages 91–100. ACM, 2015.

[IG17]   Taisuke Izumi and François Le Gall. Triangle finding and listing in CONGEST networks. In *PODC*, pages 381–389. ACM, 2017.

[JN18]   Tomasz Jurdzinski and Krzysztof Nowicki. MST in $O(1)$ rounds of congested clique. In *SODA*, pages 2620–2632. SIAM, 2018.

[KNPR15]   Hartmut Klauck, Danupon Nanongkai, Gopal Pandurangan, and Peter Robinson. Distributed computation of large-scale graph problems. In *SODA*, pages 391–410. SIAM, 2015.

[Kor16]   Janne H. Korhonen. Deterministic MST sparsification in the congested clique. *CoRR*, abs/1605.02022, 2016.

[KS20]   Fabian Kuhn and Philipp Schneider. Computing shortest paths and diameter in the hybrid network model. In *PODC*, pages 109–118. ACM, 2020.

[KSV10]   Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In *SODA*, pages 938–948. SIAM, 2010.

[Len13]   Christoph Lenzen. Optimal deterministic routing and sorting on the congested clique. In *PODC*, pages 42–50. ACM, 2013.

[LMR94]   Frank Thomson Leighton, Bruce M. Maggs, and Satish Rao. Packet routing and job-shop scheduling in $O(\text{congestion} + \text{dilation})$ steps. *Comb.*, 14(2):167–186, 1994.

[LPPP05]   Zvi Lotker, Boaz Patt-Shamir, Elan Pavlov, and David Peleg. Minimum-weight spanning tree construction in $O(\log \log n)$ communication rounds. *SIAM J. Comput.*, 35(1):120–131, 2005.

[Now19]   Krzysztof Nowicki. A deterministic algorithm for the MST problem in constant rounds of congested clique. *CoRR*, abs/1912.04239, 2019.

[PRS18]   Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. On the distributed complexity of large-scale graph computations. In *SPAA*, pages 405–414. ACM, 2018.