Distributed computations with global edges of limited bandwidth

Volodymyr Polosukhin

Distributed computations with global edges of limited bandwidth

Research Thesis

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science

Volodymyr Polosukhin

Submitted to the Senate
of the Technion — Israel Institute of Technology
Tevet 5781 Haifa January 2020

This research was carried out under the supervision of Prof. Keren Censor-Hillel, in the Faculty of Computer Science.

Some results in this thesis have been published as articles by the author and research collaborators in conferences and journals during the course of the author's masters research period, the most up-to-date versions of which being:

Keren Censor-Hillel, Dean Leitersdorf, and Volodymyr Polosukhin. Distance computations in the hybrid network model via oracle simulations. *CoRR (To appear in STACS 2021)*, abs/2010.13831, 2020.

Keren Censor-Hillel, Yannic Maus, and Volodymyr Polosukhin. Near-optimal scheduling in the congested clique. *CoRR (To appear in SIROCCO 2021)*, abs/2102.07221, 2021.

Acknowledgements

I am grateful to my advisor professor Keren Censor-Hillel for guiding me through the fields of theoretical computer science with patience and attention. I would like to thank my amazing collaborators Dean Leitersdorf and Yannic Maus for their contribution and time. Last but not least I would like to thank my parents, family, and friends for their personal support and grammar help both from near and far.

Contents

List of Figures

Abstract

In this thesis, we study the effect of global links with limited bandwidth on distributed communication. We research this question in two distributed models. One model is the well-known Congested Clique model, and another model is the recently introduced Hybrid network model.

The Congested Clique model is a distributed model where n machines execute the protocol in synchronous communication rounds. On each round, each pair of machines can exchange a single $O(\log n)$-bit long message in each direction. In other words, each pair of machines is connected by a global communication link with a bandwidth of $O(\log n)$ bits. The complexity measure in this model is the number of rounds until the last machine terminates. The natural goal is to speed up algorithms by using as much of the model's $O(n^2)$-message bandwidth as possible.

In this thesis, we address a complementary question of running in parallel as many distributed jobs as possible. Formally, we solve the distributed scheduling problem. In this problem, the system receives jobs, i.e., protocols with the corresponding inputs, which are provided in a distributed manner. The system is required to produce the output for each job in a distributed fashion, i.e., each machine outputs part of the output corresponding to its input.

Our contributions are deterministic and randomized algorithms for the distributed scheduling problem. The runtime of those algorithms under certain assumptions is close to the natural lower bounds for this problem.

In addition to the Congested Clique model, we study the Hybrid network model. It lays down the theoretical foundations for networks that combine two possible modes of communication: high-bandwidth communication with neighbors in the graph via local links and low-bandwidth all-to-all communication via global links. In this model, we address fundamental problems of distance computation. We achieve fast algorithms for distance-related problems due to *density-aware* communication protocols that utilize global links.

Our contributions include *exact* weighted shortest path from $n^{1/3}$ sources, approximations for eccentricities and diameter, and distance approximations from multiple sources.

Chapter 1

Introduction

In this thesis, we study synchronous distributed models which allow all-to-all communication. For such models, we say that the underlying communication graph is a clique. This, for example, models overlay networks, like the Internet, where point-to-point communication is guaranteed by underlying protocols. We study all-to–to-all communication using the well-known CONGESTED CLIQUE model [LPPP05] and the recently introduced HYBRID model [AHK$^+$20].

Motivated by the ever-growing number of frameworks for parallel computation, 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. The CONGESTED CLIQUE model is strongly related to the important MapReduce model [HP15a], and here we address the *distributed scheduling problem* in the CONGESTED CLIQUE model.

In this model, there are n nodes. Each node is allowed to send and receive a single message of $O(\log n)$ bits to and from each node in one round.

While there are still many problems whose round complexity in the CONGESTED CLIQUE model is open question, many problems are known to have constant round complexity in the CONGESTED CLIQUE model [Now19, GNT20, CFG$^+$19, CDP20], and some of them even deterministically [Now19, CDP20]. We ask the question which is complementary to the round complexity of a single task, which is what is the amortized complexity of problems in this model and how fast can we run multiple independent jobs in the CONGESTED CLIQUE model.

To this end, we develop a framework, which executes in a black-box manner distributed jobs in the CONGESTED CLIQUE model. We analyse the round complexity of the algorithm in terms of natural lower bounds for the scheduling problem. The interesting metrics include the minimum number of rounds required to send all the messages produced by the system assuming full utilization of the n^2 bandwidth, or the number of rounds till the slowest job terminates.

As our algorithms aim to fully utilise the bandwidth of the Congested Clique model, we emphasize the importance of the message complexity of the scheduled protocols. Our main observation is that reassigning nodes between machines allows to better exploit the bandwidth of the Congested Clique model and thus allows to develop fast scheduling algorithms. More details are elaborated upon in the Section 1.1.

An additional model we study is the Hybrid model [AHK+20]. Recently, active research started in this field [AHK+20, KS20, FHS20]. This model is motivated by hybrid data centers (hybrid DCNs) [WAK+10, CWC11, HLL+13, HHLX15], and Class-Based Hybrid software-defined networks (SDNs) [VVB14]. From a theoretical prospective, such a network is modeled as a graph with n nodes (with identifiers taken from $[n]$) and m edges. Communication occurs in synchronous rounds. On each round over each graph edge, λ messages can pass. Additionally, each node can send γ messages to an arbitrary node by its identifier. Each message is $O(\log n)$-bits long. One can view this model as a union of a *local* network with congestion on edges and a*global* clique network with limited capacity over the nodes.

The previous works [AHK+20, KS20] mostly assume $\lambda = \infty$ and $\gamma = O(\log n)$, which makes the Hybrid model a combination of the Local model and the Node-Capacitated Clique model [AGG+19]. It is worth mentioning, that most algorithms in these models are randomized and the $\gamma = O(\log n)$ choice is on the one hand small, and on the other hand large enough to apply popular concentration bounds. We stick with this definition.

In the Hybrid model, we study fundamental problems of distance computation. It was observed in [KS20] that while global or local communication networks only do not allow sublinear solutions for distance related problems, their combination does.

We improve upon results for the multiple source shortest path problem of previous work [AHK+20, KS20]. Our main algorithmic contributions are communication primitives designed for the Hybrid network model. They are based on the observation that denser regions of the graph can receive more information via the global edges. More details are provided in Section 1.2.

1.1 Scheduling in the Congested Clique model

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 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 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. We notice that the work of [Gha15], as well as the results we present in this thesis, work in distributed settings and do not require a priory knowledge of the communication patterns of the scheduled jobs.

Some complex problems are known to be reducible to multiple instances of potentially more simple problems. For example, the long-standing $O(\log\log n)$ upper bound for the MST problem [LPPP05] was broken by reducing MST to multiple instances of the graph connectivity problem [HPP$^+$15]. The state-of-the-art exact APSP algorithm [Gal16] uses a reduction of the distance product to multiple instances of matrix multiplication. The beautiful work of [GN20] solves the minimum cut problem by running multiple instances of MST in parallel and the state-of-the-art algorithm for minimum cut [GNT20] runs multiple instances of the graph connectivity algorithm.

Naturally, 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 CONGESTED 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 CONGESTED 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.

1.1.1 Deterministic Scheduling

The key ingredient in the deterministic algorithm is 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.

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 3.2, we obtain the following deterministic algorithm for scheduling t jobs that are M-memory efficient.

Theorem 1.1 (Deterministic Scheduling)**.** *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.*

At a very high level, in the algorithm for Theorem 1.1, 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 1.1 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.

1.1.2 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 1.2 (Random Shuffle Scheduling)**.** *There is a randomized algorithm in the* Congested Clique *model that schedules* $t = \operatorname{poly} n$ *jobs that are I/O efficient in* $O(t + \mathsf{GlobalCongestion} + \mathsf{dilation} \cdot \log n)$ *rounds, w.h.p.*

As the deterministic scheduling algorithm (Theorem 1.1), the scheduling algorithm of Theorem 1.2 requires neither the knowledge of $\mathsf{GlobalCongestion}$ nor the knowledge of $\mathsf{dilation}$.

Both of our scheduling algorithms for Theorem 1.1 and Theorem 1.2 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 $\mathsf{LocalCongestion}$ of a set of jobs. We obtain the following *random-delay-based* algorithm for scheduling any t jobs, without reassigning nodes. This algorithm is based on the technique developed by Leighton et al. [LMR94] and inspired by its application in the Congest model by Ghaffari [Gha15].

Theorem 1.3 (Random Delay Scheduling)**.** *There is a randomized algorithm in the* Congested 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* $\mathsf{LocalCongestion}$*, and in* $O(\mathsf{LocalCongestion} + \log \mathsf{LocalCongestion} \cdot (\mathsf{dilation} \cdot \log n + t/n))$ *rounds, w.h.p., if such a bound is not known.*

Originally, the algorithm requires the knowledge of $\mathsf{LocalCongestion}$, but we eliminate it by a standard doubling technique.

The random-delay algorithm which gives Theorem 1.3 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 Congested 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 $\mathsf{GlobalCongestion}$ is n, while the $\mathsf{LocalCongestion}$ is n^2. In such a setting, our random-shuffling algorithm from Theorem 1.2 outperforms the random-delay algorithm from Theorem 1.3. 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 $\mathsf{LocalCongestion} = n^{0.1}$, while $\mathsf{GlobalCongestion} = 1$. Thus, it is more efficient to run Theorem 1.2 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.

1.1.3 Applications

In Section 3.4, we present two applications in order to exemplify our scheduling algorithms.

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 CONGESTED 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 1.2, and we obtain the following theorem.

Theorem 1.4 (Multiple MIS instances)**.** *There is a randomized algorithm in the* CONGESTED CLIQUE *model which solves $t = \operatorname{poly} n$ 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 CONGESTED CLIQUE model by learning the composition of all permutations (see Section 3.4.2). 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 1.5 (Pointer Jumping)**.** *For $P \leq n$, there are algorithms in the* CONGESTED 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 1.1 and the randomized result using our random-shuffling scheduling algorithm in Theorem 1.2. 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 1.2 is significantly better than the random-delay based approach of Theorem 1.3. For more details we refer to Section 3.4.2.

1.2 Distance computation in the Hybrid model

An especially useful concept for distance computations in the Hybrid model is the notion of a *skeleton graph* [UY91, RZ11, LPP19, AHK$^+$20, KS20]. The nodes of this graph are obtained by marking each node of the original graph independently uniformly with probability n^{x-1} for $0 < x < 1$. They are called *skeleton nodes.* Two skeleton nodes are connected with an edge if they are are close to each other in the original graph (at a hop-distance of $\tilde{O}(n^{1-x})$). A skeleton graph is known to have many useful properties for distance computations. The most important one is that each long enough path has at least one skeleton node.

1.2.1 Simulating powerful models

In a previous work [KS20], Kuhn and Schneider noticed that in the Hybrid model, skeleton nodes have not only useful distance computation properties, but also important communication properties. Specifically, they introduced the notion of *helper nodes.* For some skeleton node, helper nodes are a set of $\tilde{O}(n^{1-x})$ nodes in $\tilde{O}(n^{1-x})$ neighborhood of the node, each of which helps $\tilde{O}(1)$ skeleton nodes. This allows to simulate the powerful Congested Clique model over the skeleton graph and use the simulation to apply the known results to the Hybrid model. First, each sender scatters the messages to its helpers using local edges. Then the sender's helpers send (via intermediate nodes) the messages to the receiver's helpers using global edges. Finally, receiver gathers the received messages using local edges. However, this approach does not fully exploit the fact that the *denser* regions of the graph have higher *capacity* then the sparser ones. To easily use this property and encapsulate new communication primitives, we introduced an Oracle and a Tiered Oracles models.

Imagine that there is a node in the graph capable of learning the entire graph in a single round. This node can solve different graph problems fast. This intuition leads to a definition of the Oracle model. Roughly speaking, in the Oracle model there is a node ℓ, the *oracle*, which can receive $\deg(v)$ messages from each node v, within *a single round.* We cannot afford to directly simulate the Oracle model as it requires too much communication in the Hybrid model. Instead, we simulate the Oracle model over a skeleton graph. Behind the proof of the Theorem 1.6, lies the intuition that in the skeleton graph there is at least one node (the node with maximum degree) with the degree at least average degree of the skeleton graph.

Theorem 1.6 (Oracle Simulation)**.** *Given a graph $G = (V, E)$, for every constant $0 < x < 1$, there is an algorithm which simulates one round of the* Oracle *model, on* *a skeleton graph $S_x = (M, E_S)$, in $\tilde{O}(n^{1-x} + n^{2x-1})$ rounds of the* Hybrid *model on G, w.h.p.*

While efficiently simulating an oracle is powerful, it still does not exploit the full capacity of the Hybrid model. This observation brings us to enhance the Oracle model and introduce the Tiered Oracles model, which consists of multiple oracles with varying abilities. In a nutshell, in the Tiered Oracles model, in each round every node v can send (the same) $\deg(v)$ messages to all nodes u with $\deg(u) \geq \deg(v)/2$. This basically means that nodes are bucketed according to degrees and each node is an oracle for all nodes in the buckets below it. One can notice that the node with the highest degree in the graph is equivalent to the oracle in the Oracle model, but here, the other nodes in the graph also have some *partial* oracle capabilities. Again, in the Theorem 1.7 we show how to simulate the Tiered Oracles model over the skeleton graph in the Hybrid model. Behind the proof lies the intuition that if we send a message to a random node in the graph, *denser* regions have a higher probability to accept it.

Theorem 1.7 (Tiered Oracles Simulation)**.** *Given a graph $G = (V, E)$, for every constant $0 < x < 1$, there is an algorithm which simulates one round of the* Tiered Oracles *model, on a skeleton graph $S_x = (M, E_S)$, in $\tilde{O}(n^{1-x} + n^{2x-1})$ rounds of the* Hybrid *model on G, w.h.p.*

One thing to notice about the round complexities of the simulations, is that they require at least $\tilde{\Omega}(n^{1/3})$ rounds. They also terminate in $\tilde{\Theta}(n^{1/3})$ rounds w.h.p. for $x = 2/3$. It is also true for the Congested Clique simulation [KS20]. This leads to a strong feeling that for some problems the round complexity might be $\tilde{\Theta}(n^{1/3})$.

1.2.2 Shortest path algorithms

Here we show how to use the simulations of Oracle, Tiered Oracles, Congested Clique and Local models over a skeleton graph to solve exact SSSP, exact $n^{2/3}$-RSSP and exact $n^{1/3}$-SSP and use reductions to obtain approximations for n^x-SSP, eccentricities and diameters. Table 1.1 is visual summary of our end results with comparison to related work.

Exact SSSP

In the *single source shortest path* (SSSP) problem, we are given a single source and require each node to learn the distance from itself to the source. We notice that using the Oracle model, we can easily deterministically compute SSSP, by making the oracle to learn the entire graph, computing locally SSSP and informing other nodes about their distance to source. Using this simple algorithm together with Theorem 1.6, we obtain SSSP on the skeleton graph, which we extend to the SSSP solution on the entire graph and thus obtain Theorem 1.8.

Problem	Variant	Approximation	This work	Previous works
SSSP	weighted	exact	$\tilde{O}(n^{1/3})$	$\tilde{O}(n^{2/5})$[KS20], $\tilde{O}(\sqrt{SPD})$[AHK$^+$
	weighted	$1+\epsilon$		$\tilde{O}(n^{1/3}\cdot\epsilon^{-6})$[AHK$^+$20]
	weighted	$(1/\epsilon)^{O(1/\epsilon)}$		n^{ϵ}[AHK$^+$20]
n^x-RSSP	unweighted	$\tilde{O}(n^{1-x/2})$	$\tilde{\Omega}(n^{x/2})$	
	weighted	exact	$\tilde{O}(n^{1/3}+n^{2x-1})$	
	weighted	$2+\epsilon$		$\tilde{O}(n^{1/3}+n^{2x-1})$ [KS20]
$n^{1/3}$-SSP	unweighted	$1+\epsilon$		$\tilde{O}(n^{1/3}/\epsilon)$ [KS20]
	weighted	exact	$\tilde{O}(n^{1/3})$	
	weighted	$3+\epsilon$		$\tilde{O}(n^{1/3}/\epsilon)$ [KS20]
n^x-SSP	unweighted	$\tilde{O}(n^{1-x/2})$		$\tilde{\Omega}(n^{x/2})$[KS20]
	unweighted	$1+\epsilon$	$\tilde{O}(n^{1/3}/\epsilon+n^{x/2})$	
	unweighted	$2+\epsilon$		$\tilde{O}(n^{1/3}/\epsilon+n^{x/2})$ [KS20]
	weighted	3	$\tilde{O}(n^{1/3}+n^{x/2})$	
	weighted	$3+\epsilon$		$\tilde{O}(n^{0.397}+n^{x/2})$ [KS20]
	weighted	$7+\epsilon$		$\tilde{O}(n^{1/3}/\epsilon+n^{x/2})$ [KS20]
eccentricities	unweighted	$1+\epsilon$	$\tilde{O}(n^{1/3}/\epsilon)$	
	weighted	3	$\tilde{O}(n^{1/3})$	
diameter	unweighted	exact		$\tilde{\Omega}(n^{1/3})$ [KS20]
	unweighted	$1+\epsilon$	$\tilde{O}(n^{1/3}/\epsilon)$	$\tilde{O}(n^{0.397}/\epsilon)$ [KS20]
	unweighted	$3/2+\epsilon$		$\tilde{O}(n^{1/3}/\epsilon)$ [KS20]
	weighted	$2-\epsilon$		$\tilde{\Omega}(n^{1/3})$ [KS20]
	weighted	2	$\tilde{O}(n^{1/3})$	$\tilde{O}(n^{2/5})$ [KS20]
	weighted	$2+\epsilon$		$\tilde{O}(n^{1/3}\cdot\epsilon^{-6})$ [AHK$^+$20]
	weighted	$2\cdot(1/\epsilon)^{O(1/\epsilon)}$		n^{ϵ}[AHK$^+$20]

Table 1.1: Comparison of our results. SPD is the length of the shortest path diameter. The results for n^x-RSSP, and weighed diameter approximation upper bounds from previous works are implicit in [AHK$^+$20, KS20]. Our upper bound for $n^{2/3}$-RSSP is tight up to poly-logarithmic factors due to our lower bound. Our approximations for n^x-SSP are also tight up to poly-logarithmic factors for $x \geq 2/3$, due to [KS20].

Theorem 1.8 (Exact SSSP)**.** *Given a weighted graph $G = (V, E)$, there is an algorithm in the* Hybrid *model that computes an exact weighted SSSP in $\tilde{O}(n^{1/3})$ rounds w.h.p.*

This result should be compared with the previous state-of-the-art algorithms for exact weighted SSSP in $\tilde{O}(n^{2/5})$ rounds [KS20], and in $\tilde{O}(\sqrt{SPD})$ rounds [AHK$^+$20], where SPD is the length of the shortest path diameter. Further, it improves upon the $\tilde{O}(n^{1/3}/\epsilon^6)$ round algorithm for a $(1+\epsilon)$-approximation of weighted SSSP [AHK$^+$20], in both the runtime and in being exact. We stress that this is a warm up, and later on we extend this result to shortest path distances from $O(n^{1/3})$ sources, instead of a single source, in the same round complexity of $\tilde{O}(n^{1/3})$.

Exact $n^{2/3}$-RSSP

We present an algorithm which solves all pairs shortest paths (APSP) using one round of the Tiered Oracles model and $O(\log n)$ rounds of the Congested Clique model[1]. We then utilize our Tiered Oracles model simulation, along with a previously known simulation of the Congested Clique model from [KS20], to simulate the APSP algorithm over skeleton graphs in the Hybrid model. Our efficient computation of APSP over a skeleton graph in the Hybrid model then leads to computing multi-source shortest paths from *random* sources in the Hybrid model.

Shortest paths from random sources is a crucial stepping stone for our later results. We show that computing shortest path distances from random sources to the entire graph, allows us to subsequently obtain fast algorithms for other distance problems. We call the problem of computing distances from sources sampled with probability n^{x-1} i.i.d n^x-RSSP.

Theorem 1.9 (Exact n^x-RSSP)**.** *Given a graph $G = (V, E)$, $0 < x < 1$, and a set of nodes M sampled independently with probability n^{x-1}, there is an algorithm in the* Hybrid *model that ensures that every $v \in V$ knows the exact, weighted distance from itself to every node in M within $\tilde{O}(n^{1/3} + n^{2x-1})$ rounds w.h.p.*

We complement Theorem 1.9 with a lower bound, following the lines of [AHK$^+$20, KS20], for approximating distances from many random sources, to any reasonable approximation factor, which tightly matches the upper bound when $x = 2/3$.

Theorem 1.10 (Lower Bound RMSSP)**.** *Let $p = \Omega(\log n/n)$ and $\alpha < \sqrt{n/p} \cdot \log(n)/2$. Any α-approximate algorithm for computing unweighted shortest paths from random sources sampled independently with probability p in the* Hybrid *network model takes $\Omega(\sqrt{p \cdot n}/\log n)$ rounds w.h.p.*

Exact $n^{1/3}$-SSP

We leverage our tight up to polylogarithmic factors algorithm for shortest paths from a set M of $\tilde{O}(n^{2/3})$ random sources in order to obtain exact weighted shortest paths from any *given* set U of $n^{1/3}$ sources. We achieve this by adapting the behavior of the given fixed source nodes to the density of their neighborhoods, as follows. A source node $s \in U$ in a sparse neighborhood broadcasts the distances to all the random source nodes from M it sees in its neighborhood. A source node $s \in U$ in a dense neighborhood takes control of one of the random sources in M in its neighborhood and uses it as a proxy in order to communicate enough information to all the other nodes in the graph so that they could determine their distances from s. We remark that this proxy approach is a *key insight* which we later encapsulate as a general tool in the Hybrid model and may potentially be of independent interest. Our approach gives the following theorem.

[1]The Congested Clique is a synchronous distributed model where every two nodes in the graph can exchange messages of $O(\log n)$ bits in every round.

Theorem 1.11 (Exact $n^{1/3}$ Sources Shortest Paths)**.** *Given a weighted graph $G = (V, E)$, and a set of sources U, such that $|U| = O(n^{1/3})$, there exists an algorithm, at the end of which each $v \in V$ knows its distance from every $s \in U$, which runs in $\tilde{O}(n^{1/3})$ rounds w.h.p.*

Theorem 1.11 raises an interesting open question of whether the complexity of SSSP in the Hybrid model is below that of computing shortest paths from $\tilde{O}(n^{1/3})$ sources.

Approximate MSSP

We also exploit our aforementioned solution for computing APSP on the skeleton graph to obtain approximate distances from a larger set of given sources (n^x-SSP), as follows.

Theorem 1.12 (Approximate Multiple Source Shortest Paths)**.** *Given a graph $G = (V, E)$, a set of sources U, where $|U| = \tilde{\Theta}(n^y)$ for some constant $0 < y < 1$, and a value $0 < \epsilon$, there is an algorithm in the* Hybrid *model which ensures that every node $v \in V$ knows an approximation to its distance from every $s \in U$, where the approximation factor is $(1 + \epsilon)$ if G is unweighted and 3 if G is weighted. The complexity of the algorithm is $\tilde{O}(n^{1/3}/\epsilon + n^{y/2})$ rounds, w.h.p.*

This result improves both in round complexity and approximation factors upon the previous results in [KS20]. The reason for this is that we compute APSP over skeleton graphs using the efficient, exact algorithm from the Tiered Oracles oracle model, while [KS20] simulate the slower, approximate algorithms of [CKK+19, CDKL19] in the Congested Clique model. Particularly, this result is tight up to polylogarithmic factors for $y \geq 2/3$ due to a lower bound of [KS20].

Approximate eccentricities and diameter

We can also approximate unweighted eccentricities by a combination of computing shortest path distances from $n^{2/3}$ random sources and performing local explorations using the local edges of the model. For approximating weighted eccentricities, this is insufficient, and here our approach is to additionally broadcast required information from each random source node regarding its $\tilde{O}(n^{1/3})$-hop neighborhood in the graph. We obtain the following result.

Theorem 1.13 (Approximate Eccentricities)**.** *Given a graph $G = (V, E)$, there is an algorithm in the* Hybrid *model that computes a $(1 + \epsilon)$-approximation of unweighted and 3-approximation of weighted eccentricities in $\tilde{O}(n^{1/3}/\epsilon)$ rounds, w.h.p.*

The unweighted eccentricities approximation directly implies a $(1+\epsilon)$ approximation for unweighted diameter. It is well known that one can approximate the diameter using a solution to SSSP, and so as a byproduct of Theorem 1.8 we get the algorithm to compute 2-approximation of the diameter in $\tilde{O}(n^{1/3})$ rounds w.h.p.

Theorem 1.14 (Approximate Diameter)**.** *Let $G = (V, E)$ be an unweighted graph, and let $\epsilon > 0$. There exists an algorithm in the* Hybrid *model which computes a $(1 + \epsilon)$-approximation of the unweighted diameter and 2-approximation of weighted diameter in $\tilde{O}(n^{1/3}/\epsilon)$ rounds, w.h.p.*

Notably, $\tilde{\Omega}(n^{1/3})$ rounds are necessary for a $(2 - \epsilon)$-approximation for weighted diameter [KS20]. Our algorithm in Theorem 1.14 thus raises the interesting open question of whether one can go below this complexity for a 2-approximation.

Chapter 2

Additional Related work

The Congested Clique model.

Many graph problems are studied in the Congested Clique model. There are fast protocols for the Congested 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 Congested 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.*

Hybrid models.

The Hybrid network model was studied in [AHK$^+$20, KS20, FHS20]. In [FHS20], distance results are obtained in one of the harsher variants of the model, where the local edges are restricted to have $\log n$ bandwidth. However, these apply only to extremely sparse graphs of at most $n + O(n^{1/3})$ edges and cactus graphs. In [ALSY90, GHSS17], a slightly different models of hybrid nature are studied.

Augustine et al. [AGG$^+$19] proposed the NODE-CAPACITATED CLIQUE model, which is similar to the CONGESTED CLIQUE model, but each node has $\log n$ bandwidth. This model is also a special case of the generalised HYBRID model [AHK$^+$20] without local edges. This allows one to use the results from the NODE-CAPACITATED CLIQUE model in the HYBRID model without modifications.

Distributed Distance Computations.

Distance related problems have been extensively studied in many distributed models. For example, in the CONGEST model, there is a long line of research on APSP [PRT12, LP13, LP15, ARKP18, Elk20, BN19, AR19, AR20] which culminated in tight, up to polylogarithmic factors, $\tilde{O}(n)$ round exact weighted APSP randomized algorithm of Bernstein and Nanongkai [BN19] and a $\tilde{O}(n^{4/3})$ round deterministic algorithm of Agarwal and Ramachandran [AR20]. [PRT12, LP13] develop an $\tilde{O}(n)$ round algorithm, optimal up to polylogarithmic factors, for unweighted APSP. The study of approximate SSSP algorithms was the focus of many recent papers [SHK$^+$12, BKKL17, HKN16] and lately Becker et al. [BKKL17] showed the solution which is close to the lower bound of Das Sarma et al. [SHK$^+$12]. In case of exact SSSP, after recent works [Elk20, GL18, FN18], there still is a gap between upper and lower bounds. The diameter and eccentricities problems are studied in the CONGEST model in [PRT12, FHW12, ACK16, ACD$^+$20].

In the CONGESTED CLIQUE model, k-SSP, APSP and diameter are extensively studied in [Nan14, CKK$^+$19, Gal16, CLT20] and approximate versions of the k-SSP and APSP problem are solved in polylogarithmic [CDKL19] and even polyloglogaritmic [DP20] time. In the more restricted BROADCAST CONGESTED CLIQUE model, in which each message a node sends in a round is the same for all recipients, distance computations are researched by [HP15b, BKKL17].

Chapter 3

Scheduling in the Congested Clique model

3.1 Preliminaries

The Congested 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 Congested Clique Model. Lenzen's routing scheme [Len13] says that a set of messages can be routed in the CONGESTED 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.

Claim 3.1.1 (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* CONGESTED 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 $O(\operatorname{poly} n)$ 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.
- **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 thesis, 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 $s^r_{i,j}$ and $t^r_{i,j}$ 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 $\mathsf{GlobalCongestion}$ as the total number of messages sent by all the jobs divided by n^2, and normalized $\mathsf{LocalCongestion}$ 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, $\mathsf{dilation} = \max_{j\in[t]} \ell_j$,

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

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

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

Claim 3.1.2 (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 $E\left[\sum_{i=1}^n X_i\right] \le \mu$. Then for all $\epsilon > 0$*

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

3.2 Deterministic Scheduling

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

Theorem 1.1 (Deterministic Scheduling)**.** *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.*

The formal definition of an M-memory efficient job as used in Theorem 1.1 is as follows.

Definition 3.2.1 (**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 1.1 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 1.1, 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 CONGESTED 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 3.2.3, 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 3.2.2. 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 3.2.3. 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 3.2.2 (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* Congested 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 3.2.2. 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 3.1.1, it completes in $O(\lceil M \cdot t/n \rceil)$ rounds. ■

Lemma 3.2.3 (Scheduling of a round with $m^r \le 2n^2$ messages)**.** *Given are t jobs that are M-memory efficient, and given is a round number, r, for which $m^r \le 2n^2$. Assume that for each $i \in [n]$, p_i holds $S_{i,j,r}$ for all $j \in [t]$. Then there exists a deterministic* Congested 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 3.2.2, 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 3.2.3. We begin with describing the algorithm (see Algorithm 3.1). Afterwards, we prove the correctness and analyze the round complexity.

Algorithm 3.1 Simulating a round with $m^r \le n^2$.

```
1: Compute the balanced assignment $\varphi\colon [n] \times [t] \mapsto [n]$.
2: Distribute the states according to the assignment $\varphi$.
3: Execute the protocol for round $r$ accounting for $\varphi$.
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 3.2.4 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 3.2.4 (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*

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

Invoking Claim 3.2.4 with $s = n \ge s_{i,j}$, $t = n \ge t_{i,j}$, $S = S_i$, 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.,

$$\sum_{j \in B_{i,i'}} s_{i,j} \le 2\left(\frac{S}{k} + s\right) \le 2\left(\frac{S_i}{(S_i/s)} + s\right) = 4s = 4n, \text{ and}$$
$$\sum_{j \in B_{i,i'}} t_{i,j} \le 2\left(\frac{T}{k} + t\right) \le 2\left(\frac{T_i}{(T_i/t)} + t\right) = 4t = 4n.$$

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 $\sum_i S_i \le 2n^2$ and $\sum_i T_i \le 2n^2$ (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 3.2.2 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 3.2.2 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 3.1.1, 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 3.2.2, 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 3.1.1. ■

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

Lemma 3.2.5 (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* Congested 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 3.2.5 uses the next claim to split all jobs into chunks that send smaller numbers of messages in order to apply Lemma 3.2.3.

Claim 3.2.6. *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* Congested Clique *model, which in $O(1)$ rounds finds the minimum index $j_0 \in \mathcal{S}$ (if it exists) that satisfies*

$$x \leq \sum_{\substack{j \in \mathcal{S}, \\ j \leq j_0}} \sum_{i=0}^{n-1} s_{i,j} \leq x + n^2. \tag{3.1}$$

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 3.2.6. **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 $[\min \mathcal{S}, \dots, \min \mathcal{S} + n^c - 1]$. 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 $1 \le i' \le n$ let $\ell_{i'} = \ell_0 + i' \cdot n^{c'-1}$. Now, each machine p_i builds prefix sums $S^i_{\ell_1}, \dots, S^i_{\ell_n}$ of its own numbers, that is

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

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 $S_{\ell_1}, \dots, S_{\ell_n}$ 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 $x \le S_{j_0}$ 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 $\sum_{i \in [n]} s_{i,j_0} \le n^2$, and as $S_{j_0 - 1} = \sum_{j < j_0} \sum_{i \in [n]} s_{i,j} \le x$, we obtain the claimed upper bound in Eq(3.1).

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. ■

We continue with the proof of Lemma 3.2.5.

Proof of Lemma 3.2.5. **Algorithm.** A short pseudocode is given in Algorithm 3.2.

Algorithm 3.2 Scheduling of a round.

```
1: Split jobs into chunks $J_1, J_2, \dots, J_k$.
2: for each chunk $J_{k'}$ do
3:     Apply Algorithm 3.1 on $J_{k'}$.
4: end for
```

We use Claim 3.2.6 to split the jobs into $k = O(\lceil m^r / n^2 \rceil)$ chunks $J_1, \dots, J_k$, 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 3.2.3 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 $J_1, \dots, J_{k'-1}$ have been formed and let $\mathcal{S} = [t] \setminus (J_1 \cup \dots \cup J_{k'-1})$. We apply Claim 3.2.6 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 3.2.6, 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 3.2.3 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 3.2.6. The invocation of Lemma 3.2.3 on chunk $J_{k'}$ takes $O(\lceil M|J_{k'}|/n \rceil)$ rounds per chunk. Thus, the round complexity of the algorithm is

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

Finally, we use Lemmas 3.2.3 and 3.2.5 to obtain the near-optimal scheduling of Theorem 1.1.

Theorem 1.1 (Deterministic Scheduling)**.** *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 1.1. We repeatedly apply Lemma 3.2.5 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 3.2.5 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 3.2.5, 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

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

3.3 Randomized Scheduling

In this section we show and compare the two approaches for randomized scheduling: random shuffling (Section 3.3.1) and random delaying (Section 3.3.2). In contrast to

Theorem 1.1, the results in this section do not require the jobs to be memory efficient.

3.3.1 Scheduling through Random Shuffling

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

Theorem 1.2 (Random Shuffle Scheduling)**.** *There is a randomized algorithm in the* Congested Clique *model that schedules* $t = \operatorname{poly} n$ *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 1.2 is as follows.

Definition 3.3.1 (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 1.2 (see Algorithm 3.3) consists of three steps: `Input Shuffling`, `Execution`, and `Output Unshuffling`.

Algorithm 3.3 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 3.1.1.
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 $\pi_j : [n] \mapsto [n]$. 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 $\mathsf{dilation}$ 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 3.1.1 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 $m^r = \sum_{j\in[t]} \sum_{i\in[n]} s^r_{i,j}$, where $s^r_{i,j}$ is the number of messages that node $v_{i,j}$ sends in round r. They do this by having each machine p_i send $\sum_{j\in[t]} s^r_{\pi_j^{-1}(i),j}$ 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 $\sum_{j\in[t]} s^r_{\pi_j^{-1}(i),j}$ $(\sum_{j\in[t]} t^r_{\pi_j^{-1}(i),j})$, which is the number of messages that machine p_i has to send (receive) in phase r, to be used when invoking Claim 3.1.1.

`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 3.3.2. *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 3.3.2. 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:

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

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 3.1.2 (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 $\Pr\left[S_i^r > 3m^r/n^2 + 2c' \ln n\right] < n^{-c'}$.

Denote by T_i^r 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

$$\Pr\left[T_i^r > 3\frac{m^r}{n^2} + 2c' \ln n\right] < n^{-c'}.$$

By a union bound over S_i^r, T_j^r for all $i \in [n]$, we obtain that for some i one of the event $S_i^r > 3(m^r/n^2 + 2(c+2)\ln n)$, $T_i^r > 3(m^r/n^2 + 2(c+2)\ln n)$ 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.

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 3.3 satisfies the statement of Theorem 1.2.

Proof of Theorem 1.2. We prove the correctness and bound the runtime of the presented algorithm (see Algorithm 3.3).

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 3.1.1 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 3.3.2, $X = O(m^r/n + n\log n)$ 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 3.1.1 completes in $O(m^r/n^2 + \log n)$ rounds, w.h.p. Thus, the overall round complexity of the algorithm is

$$\begin{aligned} O(t + \sum_{r\in[\mathsf{dilation}]} (m^r/n^2 + \log n)) &= O(t + \sum_{j\in[t]} m_j/n^2 + \mathsf{dilation} \cdot \log n) \\ &= O(t + \mathsf{GlobalCongestion} + \mathsf{dilation} \cdot \log n). \blacksquare \end{aligned}$$

3.3.2 Scheduling through Random Delays

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

Theorem 1.3 (Random Delay Scheduling)**.** *There is a randomized algorithm in the* Congested 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* $\mathsf{LocalCongestion}$*, and in* $O(\mathsf{LocalCongestion}+\log\mathsf{LocalCongestion}\cdot(\mathsf{dilation}\cdot\log n+t/n))$ *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 3.1.1). This approach uses that all nodes know a bound on $\mathsf{LocalCongestion}$, 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 $\mathsf{LocalCongestion}$ 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 (`$O(D+\mathsf{dilation})$` phases):` In *phase* r we progress each job j (for which $r \ge D_j$ 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 3.1.1, 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 3.1.1 fails.

Algorithm 3.4 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 $r \ge D_j$ by one round where the messages of all jobs are routed with
   Claim 3.1.1.
```

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 3.3.3. *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 $S = \max_{i\in[n]} \sum_{j\in[t]\colon r\geq D_j} s_{i,j}^{r-D_j}$, $R = \max_{i\in[n]} \sum_{j\in[t]\colon r\geq D_j} t_{i,j}^{r-D_j}$, 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:

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

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 3.1.2 (Hoeffding Bound) with $\epsilon = 2\cdot c'$, we bound the probability of S_i being larger than the expected value by

$$\begin{aligned}
\Pr[S_i \geq (1+2\cdot c')(\mathsf{LocalCongestion}/D)] &= \Pr[\sum_{j\in[t]} (S_{i,j}/n) \geq (1+2\cdot c')(\mathsf{LocalCongestion}/D)] \\
&\leq e^{-\frac{(2\cdot c')^2}{2+2\cdot c'}\frac{\mathsf{LocalCongestion}}{D}} = e^{-\frac{4c'^2}{2+2c'}\ln n} \leq n^{-c'},
\end{aligned}$$

where the second transition is due to Claim 3.1.2 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 $X = O(\mathsf{LocalCongestion}\cdot n/D)$. ■

The following simple routing primitives are used in the random-delay based algorithm of Theorem 1.3.

Definition 3.3.4. (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 3.3.5. *There is an algorithm in the* Congested Clique *model, which solves the multiple broadcast problem in $O(\lceil \sum_{i\in[n]} m_i/n \rceil)$ rounds.*

Proof. The pseudocode is given in Algorithm 3.5. 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 3.5 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.
6: end for
```

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. ■

Proof of Theorem 1.3. We prove the correctness and bound the runtime for the aforementioned algorithm (Algorithm 3.4).

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 3.1.1 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 3.1.1 with a bound of $X = O(\mathsf{LocalCongestion}/D)$, which due to Lemma 3.3.3 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 3.3.5. For each phase $r \in [D + \mathsf{dilation}]$, by Lemma 3.3.3 $X = O(\mathsf{LocalCongestion} \cdot n/D)$ 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 $D + \mathsf{dilation} = O(\text{poly}\,(n))$ rounds, this holds on each round w.h.p. Thus, invoking the algorithm from Claim 3.1.1 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 $\sum_{\kappa=0}^{O(\log \mathsf{LocalCongestion})} O(t/n + 2^\kappa + \mathsf{dilation} \cdot \log n) = O(\mathsf{LocalCongestion} + \log \mathsf{LocalCongestion} \cdot (t/n + \mathsf{dilation} \cdot \log n))$. ■

3.4 Applications: MIS & Pointer Jumping

In this section we apply the scheduling algorithms developed in Sections 3.2 and 3.3 on protocols which solve MIS (Section 3.4.1) and Pointer Jumping (Section 3.4.2). We analyze the round complexity of the developed algorithms.

3.4.1 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 1.2.

Theorem 1.4 (Multiple MIS instances)**.** *There is a randomized algorithm in the* Congested Clique *model which solves $t = \text{poly}\, n$ 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 Congested 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 3.1 (Analysis of the MIS protocol of [GGK+18, Theorem 1.1])**.** *There is a randomized MIS protocol in the* Congested Clique + Lenzen's Routing *model which completes in $O(\log\log\Delta)$ rounds and sends $O(n^2)$ messages, w.h.p.*

Given Theorem 3.1, we prove Theorem 1.4 as follows.

Proof of Theorem 1.4. By Theorem 3.1, a set of t jobs of the MIS protocol of [GGK+18] have $\mathsf{dilation} = O(\log\log\Delta)$ and $\mathsf{GlobalCongestion} = \frac{t \cdot n^2}{n^2} = t$, w.h.p. By Theorem 1.2, w.h.p., we can schedule the t jobs in a number of rounds bounded by $O(t + \mathsf{GlobalCongestion} + \mathsf{dilation} \cdot \log n) = O(t + \log\log\Delta \log n)$. ■

Remark. Theorem 1.4 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 1.3 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 1.4 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 3.1.

Proof of Theorem 3.1. 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 3.6).

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 $\Delta' = \min \{ \Delta, \operatorname{poly} \log n \}$.

Algorithm 3.6 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:    end while
12:    The leader $v^*$ informs the nodes in $M_k \setminus M_{k-1}$ that they are such.
13:    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.
14:    The nodes in $N_G(M_k \setminus M_{k-1})$ inform their neighbors that they are such and become inactive.
15:    $k \leftarrow k + 1$.
16: end while
17: for $k$ from 0 to $O(\log\log\Delta')$ do
18:    Each active node $v_i$ sends all adjacent edges from $H^{2^k}$ to each of its neighbors in $H^{2^k}$.
19: end for
20: 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.
21: 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

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

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 Congested Clique model this would require some routing scheme. However, in the Congested Clique + Lenzen's Routing model this is part of the model definition.

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

3.4.2 Pointer Jumping

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

Definition 3.4.1 (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 Congested Clique protocol for solving P-pointer jumping with a complexity of $O(P\cdot n)$ messages.

Claim 3.4.2 (Pointer jumping)**.** *For $P = O(n)$, there is a deterministic $O(P)$-memory efficient protocol in the* Congested Clique + Lenzen's Routing *model which solves the pointer jumping problem in $O(\log n)$ rounds and $O(P\cdot n)$ messages.*

Algorithm 3.7 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 $\pi_i \circ \pi_{i+1} \circ \cdots \circ \pi_{\min\{i+2^k, n\}-1}$.
    end if
    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 $v_{i-2^k}$.
    end if
  end for
end for
$v_{i'}$ 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 3.4.2.

Algorithm and correctness. The pseudo-code for the simple well known protocol for the pointer jumping problem is presented in Algorithm 3.7. 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[2] . Given only the index of the round, it is possible for each node to deduce the number of messages each node sends to it.

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. ■

[2]In the Congested Clique model this algorithm requires some routing scheme, but Claim 3.1.1 is not known to run in $O(P \log n)$ bits of memory. However, our results apply in the potentially more powerful model of Congested 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.

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 1.5 (Pointer Jumping)**.** *For $P \leq n$, there are algorithms in the* Congested 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 1.5. The first part of the theorem follows immediately from Claim 3.4.2 and Theorem 1.1. By Theorem 1.1, running t instances of the protocol from Claim 3.4.2 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 1.2, running t instances of the protocol from Claim 3.4.2 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 1.2 is significantly better than the random-delay based approach of Theorem 1.3. 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 1.3 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.

Chapter 4

Distance Computations in the Hybrid Network Model via Oracle Simulations

4.1 Preliminaries

We provide here some definitions and claims that are critical for reading the main part of the paper. We use the following variant of the HYBRID model, introduced in [AHK$^+$20].

Definition 4.1.1 (HYBRID Model)**.** In the HYBRID model, a synchronous network of n nodes with identifiers in $[n]$, is given by a graph $G = (V, E)$. In each round, every node can send and receive λ messages of $O(\log n)$ bits to/from each of its neighbors (over *local edges*) and an additional γ messages in total to/from any other nodes in the network (over *global edges*). If in some round more than γ messages are sent via global edges to/from a node, only γ messages selected adversarially are delivered.

We follow the previous work of [AHK$^+$20, KS20] and consider $\lambda = \infty, \gamma = O(\log n)$. Notice that the HYBRID model can also capture the classic LOCAL[3] model, with $\lambda = \infty, \gamma = 0$, the classic CONGEST model, with $\lambda = O(1), \gamma = 0$, the CONGESTED CLIQUE model, with $\lambda = O(1), \gamma = 0$ and G being a clique, the CONGESTED CLIQUE + Lenzen's Routing with $\lambda = 0, \gamma = n$ and the NODE-CAPACITATED CLIQUE model [AGG$^+$19], with $\lambda = 0, \gamma = O(\log(n))$.

Many of our results hold for weighted graphs $G = (V, E, w)$. We assume an edge weight is given by a function $w \colon E \mapsto \{1, 2, \ldots, W\}$ for a W which is polynomial in n. When we *send* an edge as part of a message in any algorithm, we assume it is sent along with its weight.

[3]The LOCAL and CONGEST models are synchronous distributed models where every two neighbors in the graph can exchange messages of unlimited size or of $O(\log n)$ bits, respectively, in each round.

4.1.1 Notations and Problem Definitions

We use the following definitions related to graphs. Given a graph $G = (V, E)$ and a pair of nodes $u, v \in V$, we denote by $hop(u, v)$ the hop distance between u and v, by $N^h_G(v)$ the h-hop neighborhood of v, by $d^h_G(u, v)$ the weight of the lightest path between u and v of at most h-hops, and if there is no path of at most h-hops then $d^h_G(u, v) = \infty$. In the special case of $h = 1$, we denote by $N_G(v)$ the neighbors of v and in the special case of $h = \infty$, we denote by $d_G(u, v)$ the weight of the lightest path between u and v. We also denote by $\deg_G(v)$ the degree of v in G. Whenever it is clear from the context we drop the subscript of G and just write N, N^h, d, d^h or $\deg(v)$.

We define the following problems in the Hybrid model.

Definition 4.1.2 (k-Source Shortest Paths (k-SSP))**.** Given a graph $G = (V, E)$, and a set $S \subseteq V$ of k sources. Every $u \in V$ is required to learn the distance $d_G(u, s)$ for each $s \in S$. The case where $k = 1$, is called the *single source shortest paths* problem (SSSP).

Definition 4.1.3 (n^x-Random Sources Shortest Path (n^x-RSSP))**.** Given a graph $G = (V, E)$, and a set $M \subseteq V$ of sources, such that each $v \in V$ is sampled independently with probability n^{x-1} to be in M. Every $u \in V$ is required to learn the distance $d_G(u, s)$ for each $s \in M$.

In the approximate versions of these problems, each $u \in V$ is required to learn an (α, β)-approximate distance $\widetilde{d}(u, v)$ which satisfies $d(u, v) \leq \widetilde{d}(u, v) \leq \alpha \cdot d(u, v) + \beta$, and in case $\beta = 0$, $\widetilde{d}(u, v)$ is called an α-approximate distance.

Definition 4.1.4 (Eccentricity and diameter)**.** Given a graph $G = (V, E)$ and node $v \in V$, the *eccentricity* of v is the farthest shortest path distance from v, i.e., $ecc(v) = \max_{u \in V} d(v, u)$ and the diameter $D = \max_{v \in V} \{ ecc(v) \}$ is the maximum eccentricity. An α-approximation of all eccentricities is a function $\widetilde{ecc}(v)$ which satisfies $ecc(v)/\alpha \leq \widetilde{ecc}(v) \leq ecc(v)$ for all nodes v. An α-approximation of the diameter is a value $\widetilde{D}$ which satisfies $D/\alpha \leq \widetilde{D} \leq D$.

4.1.2 Communication

A useful communication routine which we use in this section is Claim 4.1.5 *(Communication)*. It was proved for a weaker Node-Capacitated Clique model [AGG+19, Theorem 2.2], and thus directly holds in the Hybrid model.

Claim 4.1.5 (Aggregate And Broadcast)**.** *Let S be some ground set and f be a globally known function, which maps some multiset $S' \subseteq S$ to a value $f(S') \in S$. Assume that there exists a globally known function g, such that for any multiset S' and any partition $S_1, \cdots, S_\ell$ of S' holds $f(S) = g(f(S_1), \ldots, f(S_\ell))$. Assume also that each node u has an input value $val(u) \in S$. Then there is an algorithm in the* Hybrid *model, which runs for $O(\log n)$ rounds, after which each node u knows $f(v_1, \ldots, v_n)$.*

For example, we use Claim 4.1.5 to sum numbers that each node has and make the result globally known.

A tool we need for our exact $n^{1/3}$-SSP algorithm is a *token dissemination* algorithm, given in [AHK+20, Theorem 2.1].

Definition 4.1.6 (Token Dissemination Problem)**.** The problem of making k distinct tokens globally known, where each token is initially known to one node, and each node initially knows at most ℓ tokens is called the (k,ℓ)-*Token Dissemination (TD) problem.*

Claim 4.1.7 (Token Dissemination)**.** *[KS20, Theorem 2.2] There is an algorithm that solves (k,ℓ)-TD in the* Hybrid *model in $\tilde{O}(\sqrt{k}+\ell)$ rounds, w.h.p.*

4.1.3 Skeleton Graphs

In a nutshell, given a graph $G=(V,E)$, a skeleton graph $S_x=(M,E_S)$, for some constant $0<x<1$, is generated by letting every node in V independently join M with probability n^{x-1}. Two nodes in M have an edge in E_S if there exists a path between them in G of at most $h=\tilde{O}(n^{1-x})$ hops. This graph w.h.p. satisfies many useful properties in terms of distance computation, which for simplicity of presentation we add to its definition, provided below. A crucial property is that for any two nodes, if the shortest path between them in G has more than h hops, then there exists a shortest path between them in G on which every roughly h nodes there is a node from M (all such skeleton properties hold w.h.p.).

Definition 4.1.8 (Skeleton Graph, Combined Definition of [AHK+20, KS20])**.** Given a graph $G=(V,E)$ and a value $0<x<1$, a graph $S_x=(M,E_S)$ is called a skeleton graph in G, if all of the following hold.

1. Each $v\in V$ is included to M independently with probability n^{x-1}.
2. $\{v,u\}\in E_S$ if and only if there is a path of at most $h=\tilde{\Theta}(n^{1-x})$ edges between v,u in G.
3. Every node $v\in M$ knows all its incident edges in E_S.
4. S_x is connected.
5. For any two nodes $v,v'\in M$, $d_S(v,v')=d_G(v,v')$.
6. For any two nodes $u,v\in V$ with $hop(u,v)\geq h$, there is at least one shortest path P from u to v in G, such that any sub-path Q of P with at least h nodes contains a node $w\in M$.
7. $|M|=\tilde{O}(n^x)$.

The following claim summarizes what is proven in [AHK+20] regarding the construction of a skeleton graph from a set of random marked nodes, w.h.p.

Claim 4.1.9 (Skeleton from Random Nodes). *Given a graph $G = (V, E)$, a value $0 < x < 1$, and a set of nodes M marked independently with probability n^{x-1}, there is an algorithm in the* Hybrid *model which constructs a skeleton graph $S_x = (M, E_S)$ in $\tilde{O}(n^{1-x})$ rounds w.h.p. If also given a single node $s \in V$, it is possible to construct $S_x = (M \cup \{ s \}, E_S)$, without damaging the properties of S_x.*

We extract the following basic claim, used in the proof of [AHK+20, Theorem 2.7] for a $(1+\epsilon)$-approximation for SSSP, and slightly extend it to use for multiple source problem and arbitrary approximation factors. It states that given a skeleton graph and a set of sources, if every skeleton node knows any approximation to its distance from every source, then it is possible to efficiently reach a state where every node in the graph knows the approximation for its own distance from any of the sources. We give the proof for the sake of self-containment. The idea is that each node locally explores its $\tilde{O}(n^{1-x})$ neighborhood and identifies for each source the best skeleton node in its neighborhood to go through.

Claim 4.1.10 (Extend Distances). *[AHK+20, Theorem 2.7] Let $G = (V, E)$, let $S_x = (M, E_S)$ be a skeleton graph, and let $V' \subseteq V$ be the set of source nodes. If for each source node $s \in V'$, each skeleton node $v \in M$ knows the (α, β)-approximate distance $\tilde{d}(v, s)$ such that $d(v, s) \le \tilde{d}(v, s) \le \alpha d(v, s) + \beta$, then each node $u \in V$ can compute for all source nodes $s \in V'$, a value $\tilde{d}(u, s)$ such that $d(u, s) \le \tilde{d}(u, s) \le \alpha d(u, s) + \beta$ in $\tilde{O}(n^{1-x})$ rounds.*

Proof. First, each node $u \in V$ learns $\tilde{d}(v, s)$ for each source node $s \in V'$ and skeleton node $v \in M$ in its $h = \tilde{\Theta}(n^{1-x})$-hop neighborhood. Then, node $u \in V$ approximates its distance to each source $s \in V'$ using Eq. (4.1), as follows.

$$\tilde{d}(u, s) = \min \left\{ d^h(u, s), \min_{v \in M \cap N_G^h(u)} \left\{ d^h(u, v) + \tilde{d}(v, s) \right\} \right\} \tag{4.1}$$

For each node $u \in V$, skeleton node $v \in M \cap N_G^h$, and source $s \in V'$, $d^h(u, v) \ge d(u, v)$, it holds that $\tilde{d}(v, s) \ge d(v, s)$ and $d^h(u, s) \ge d(u, s)$, by triangle inequality $d(u, v) + d(v, s) \ge d(u, s)$, thus $\tilde{d}(u, s) \ge d(u, s)$. To show that $\tilde{d}(v, s) \le \alpha \cdot d(v, s) + 2\beta$, we consider two cases. If there is a shortest path of at most h hops between s and v, then its length appears as $d^h(u, s)$ in the Eq. (4.1). Otherwise, by the Property 6 of the definition of the skeleton graph, there is a shortest path from u to s on which there is a node $v' \in M$ in one of its h first notes. This means that $d(u, v') + d(v', s) = d(u, s)$ and also $d^h(u, v') = d(u, v')$ (as each sub-path of shortest path is a shortest path) and so for this $v = v'$ the corresponding element $d^h(u, v') + \tilde{d}(v', s)$ appears as an option in Eq. (4.1). This implies that $\tilde{d}(u, s) \le d^h(u, v') + \tilde{d}(v', s) \le d(u, v') + \alpha \cdot d(v', s) + \beta =$

$\alpha \cdot (d(u,v') + d(v',s)) + \beta \leq \alpha \cdot d(u,s) + \beta$. Thus, in both cases $\tilde{d}(u,s) \leq \alpha \cdot d(u,s) + \beta$, and therefore $\tilde{d}$ is an (α, β)-approximation for the shortest path.

Learning the information in the h-hop neighborhood requires $h = \tilde{O}(n^{1-x})$ rounds and the rest is done locally, so overall the algorithm runs in $\tilde{O}(n^{1-x})$ rounds of the Hybrid model. ■

4.2 Oracles in the Hybrid model

This section is split into three parts. Initially, as preliminaries, we show simulations of the Local and Congested Clique models in the Hybrid model, citing [KS20] for the Congested Clique simulation. Then, we devote a section to each of the two new oracle models in order to introduce them and present their simulations in the Hybrid model.

4.2.1 Model Simulation Preliminaries

We will use simulations of the Local and Congested Clique models, as follows.

Lemma 4.2.1 (Local Simulation)**.** *Given a graph $G = (V, E)$, and a skeleton graph $S_x = (M, E_S)$, it is possible to simulate one round of the* Local *model over S_x within $\tilde{O}(n^{1-x})$ rounds in G in the* Hybrid *model. That is, within $\tilde{O}(n^{1-x})$ rounds in G in the* Hybrid *model, any two adjacent nodes in S_x can communicate any amount of data between each other.*

The proof follows trivially due to the definition of the Hybrid model and Property 2 in the definition of a skeleton graph S_x, since in S_x two skeleton nodes are connected if they are within $\tilde{\Theta}(n^{1-x})$ hops in the original graph G. Thus, one round of the Local model over S_x is obtained in the Hybrid network in $\tilde{O}(n^{1-x})$ rounds, by having neighboring skeleton nodes communicate through the local edges.

Lemma 4.2.2 (Congested Clique Simulation)**.** *[KS20, Corollary 4.1.] Given a graph $G = (V, E)$, and a skeleton graph $S_x = (M, E_S)$, for some constant $0 < x < 1$, it is possible to simulate one round of the* Congested Clique *model over S_x in $\tilde{O}(n^{2x-1} + n^{\frac{x}{2}})$ rounds of the* Hybrid *model on G, w.h.p. That is, within $\tilde{O}(n^{2x-1} + n^{\frac{x}{2}})$ rounds of the* Hybrid *model on G, w.h.p., every node $v \in M$ can, for each node $u \in M$, each send a unique $O(\log n)$ bit message to u.*

4.2.2 Simulating the Oracle Model

Here, we define the Oracle model and then show how to efficiently simulate it over a skeleton graph in the Hybrid model.

Definition 4.2.3 (ORACLE Model)**.** In the ORACLE model over a network G, there exists one *oracle* node ℓ, which in every round can send to and receive from every node v a number of $O(\log n)$-bit messages that is equal to the degree of v in G.

Theorem 1.6 (ORACLE Simulation)**.** *Given a graph $G = (V, E)$, for every constant $0 < x < 1$, there is an algorithm which simulates one round of the* ORACLE *model, on a skeleton graph $S_x = (M, E_S)$, in $\tilde{O}(n^{1-x} + n^{2x-1})$ rounds of the* HYBRID *model on G, w.h.p.*

Proof. We prove the claim by showing how to simulate a round of the ORACLE model in $O(1)$ rounds of the CONGESTED CLIQUE model and 1 round of the LOCAL model. Then, invoking the simulations of Lemmas 4.2.1 and 4.2.2, gives the desired round complexity in the HYBRID model.

We show how to send messages to the oracle, and the receiving part is symmetric. The pseudocode is given by Algorithm 4.1. First, each $v \in M$ broadcasts its degree in S_x using one round of the CONGESTED CLIQUE model (Line 1) and selects as an oracle ℓ the node with largest degree in S_x, breaking ties by identifier (Line 2). Then, the identifiers of the neighbors of ℓ are broadcast using one round of the CONGESTED CLIQUE model (Line 3). The actual messages are sent to these neighbors instead of to ℓ itself (Line 4) and ℓ learns all these messages in 1 round of the LOCAL model in Line 5.

Clearly, all the nodes select the same oracle ℓ (Line 2). Due to the definition of the ORACLE model, each node $v \in M$ has $\deg_{S_x}(v)$ messages to send, and since $\deg_{S_x}(\ell) \geq \deg_{S_x}(v)$, there are enough neighbors of ℓ to receive one message from v per neighbor, which is why Line 4 can work. ■

Algorithm 4.1 Simulating the ORACLE model in the CONGESTED CLIQUE with LOCAL.

```
CONGESTED CLIQUE model: $v \in M$ broadcasts $\deg_{S_x}(v)$.
Select an oracle $\ell \leftarrow \arg\max_{v \in M} \{ (\deg_{S_x}(v), v) \}$.
CONGESTED CLIQUE model: $v \in M$ broadcasts if it is a neighbor of $\ell$.
CONGESTED CLIQUE model: $v \in M$ sends $i$-th message to $i$-th neighbor of $\ell$ for each $i$.
LOCAL model: $\ell$ collects the messages from its neighbors.
```

4.2.3 Simulating the Tiered Oracles Model

We further enhance our ORACLE model and define the TIERED ORACLES model, where, roughly speaking, all nodes, in parallel, can learn all the edges adjacent to nodes with degrees in lower degree buckets. To simulate the stronger TIERED ORACLES model over a skeleton graph in the HYBRID model, we need additional insights. Here, we use the fact that when we scatter messages independently at random, denser neighborhoods are more likely to receive a given message than sparse neighborhoods. In other words, while for simulating the ORACLE model, we used the LOCAL round only to concentrate information in a single node ℓ, here we exploit the information that *each* node can gather from its neighborhood.

Definition 4.2.4 (TIERED ORACLES Model)**.** In the TIERED ORACLES model over a network G, in every round, suppose each node v has a set of $O(\log n)$-bit messages M_v of size $|M_v| = \deg(v)$, then each node u can receive all messages in M_v for every v such that $\deg(u) \geq \deg(v)/2$.

To simulate the TIERED ORACLES model, we first prove the following model-independent tool.

Lemma 4.2.5 (Sampled neighbors)**.** *Given is a graph $G = (V, E)$. For a value $c \leq n$, there is a value $x = \tilde{O}(n/c)$ such that the following holds w.h.p.: Let $V' \subseteq V$ be a subset of $|V'| = x$ nodes sampled uniformly at random from M. Then each node $u \in V$ with* $\deg(u) \geq c$ *has a neighbor in V'.*

Proof. For some node $u \in V$, the probability of not having a neighbor sampled to the set V' is $(1 - \deg(u)/n)^x \leq e^{-x \cdot \det(u)/n} \leq e^{x \cdot c/n}$. Thus there exists $x = \tilde{O}(n/c)$ such that node u has a neighbor in the set V', w.h.p. ■

Theorem 1.7 (TIERED ORACLES Simulation)**.** *Given a graph $G = (V, E)$, for every constant $0 < x < 1$, there is an algorithm which simulates one round of the* TIERED ORACLES *model, on a skeleton graph $S_x = (M, E_S)$, in $\tilde{O}(n^{1-x} + n^{2x-1})$ rounds of the* HYBRID *model on G, w.h.p.*

Proof. We prove the claim by reducing one round of the TIERED ORACLES model to $\tilde{O}(1)$ rounds of the CONGESTED CLIQUE model followed by a round of the LOCAL model on the skeleton graph S_x. By Lemmas 4.2.1 and 4.2.2, we obtain that the resulting round complexity is $\tilde{O}(n^{1-x} + n^{2x-1})$.

For each $v \in M$, let M_v be the set of messages, of size $|M_v| = \deg_{S_x}(v)$, which v desires to broadcast. For each message in M_v node $v \in M$ samples uniformly at random $x = \tilde{O}(2 \cdot |M| / \deg_{S_x}(v))$ nodes of M and sends the message to those nodes. As each node sends and receives $\tilde{O}(|M|)$ messages this can be done using with the well known routing theorem of Lenzen [Len13, Theorem 3.7] by simulating $\tilde{O}(1)$ rounds of the CONGESTED CLIQUE model. Alternatively, this can be done in the same round complexity by applying the algorithm for *token routing* [KS20, Theorem 2.2]. Afterwards, we simulate one round of the LOCAL model over S_x for each node to learn tokens received by its neighbors in S. Due to Lemma 4.2.5 *(Simulating the* TIERED ORACLES *Model)*, each node $u \in V$ learns messages from each v such that $\deg_{S_x}(u) \geq \deg_{S_x}(v)/2$ w.h.p.

4.3 Shortest Paths Algorithms

4.3.1 Warm-Up: Exact SSSP

As a warm-up, we show how to compute exact SSSP in the Oracle model, and then we simulate this on a skeleton graph in the Hybrid model in order to get exact SSSP in the Hybrid model within $\tilde{O}(n^{1/3})$ rounds. We note that later, in Section 4.3.3, we obtain this complexity for exact distances from a much larger set, of $O(n^{1/3})$ sources.

Lemma 4.3.1 (Exact SSSP in the Oracle Model)**.** *There is a* deterministic *algorithm in the* Oracle *model that given a weighted graph $G = (V, E)$ and source $s \in V$ solves exact SSSP in $O(1)$ rounds.*

Proof. Let $s \in V$ be the source node. We solve the problem in two communication rounds. In the first round, oracle ℓ learns all of E by receiving from each node v its adjacent edges. Afterwards, oracle ℓ, given all the edges in the graph G, locally computes the distance from s to every other node. In the second round, oracle ℓ sends for each $v \in V$ the value $d(s, v)$. It is clear that the algorithm computes SSSP from $s \in V$, and that it takes two rounds in the Oracle model.

Theorem 1.8 (Exact SSSP)**.** *Given a weighted graph $G = (V, E)$, there is an algorithm in the* Hybrid *model that computes an exact weighted SSSP in $\tilde{O}(n^{1/3})$ rounds w.h.p.*

Proof. Let s be the source node, and let $x = 2/3$. We start by constructing a skeleton graph $S_x = (M, E_S)$, by sampling nodes with probability $n^{-1/3}$ and using Claim 4.1.9 *(Skeleton Graphs)*. Then, we simulate the algorithm given in Lemma 4.3.1 in the Oracle model, which computes the distance $d_S(s, v)$ from s to each node $v \in M$. By Property 5 of the skeleton graph, for every $v \in M$, it holds that $d_S(s, v) = d_G(s, v)$. To extend this and compute the distance from s for each node $v \in V$, we apply Claim 4.1.10 *(Skeleton Graphs)*.

Constructing the skeleton graph takes $O(h) = \tilde{O}(n^{1/3})$ rounds w.h.p., by Claim 4.1.9 *(Skeleton Graphs)*. Simulating the algorithm from Lemma 4.3.1 completes in $\tilde{O}(n^{1/3})$ rounds w.h.p. by Theorem 1.6 *(Simulating powerful models)*. Applying Claim 4.1.10 *(Skeleton Graphs)* takes $\tilde{O}(n^{1/3})$ rounds. Therefore, overall, the execution of the algorithm completes in $\tilde{O}(n^{1/3})$ rounds w.h.p.

4.3.2 Exact n^x-RSSP

Recall that in Definition 4.1.3 *(Notations and Problem Definitions)*, we are given set of roughly n^x sources sampled independently with probability n^{x-1}, and we need for each node to compute its distance to each source. We do so by constructing a skeleton graph S_x from the random sources. We show that using one round of the Tiered Oracles model, and $O(\log n)$ rounds of the Congested Clique model, one can solve APSP over S_x. To do so, we split the nodes of the graph into $\lceil \log n \rceil$ tiers by degree and compute APSP by proceeding tier after tier and computing distances from current tier to all the tiers below.

Lemma 4.3.2 (APSP in Congested Clique with Tiered Oracles)**.** *There is a* deterministic *algorithm which, given a weighted graph $G = (V, E)$, solves exact APSP on G using $O(\log |V|)$ rounds of the* Congested Clique *model and one round of the* Tiered Oracles *model.*

Proof. The pseudocode for the algorithm appears in Algorithm 4.2. We partition the nodes V by their degrees into $\lceil \log |V| \rceil$ tiers, $T_j = \{ v \in V : 2^j \leq \deg(v) < 2^{j+1} \}$ for $0 \leq j < \lceil \log |V| \rceil$. Denote by $T_{>i} = \bigcup_{k>i} T_k$ the nodes in all tiers $k > i$ and by $T_{\leq i} = \bigcup_{k \leq i} T_k$ the nodes in all tiers $k \leq i$. Similarly, define $T_{\geq i}$ and $T_{<i}$. Denote by $d_{\leq i}(u, v)$ the weight of the shortest path between u and v that uses only edges adjacent to at least one node in $T_{\leq i}$.

Algorithm 4.2 Exact-APSP: Computes exact APSP using the Congested Clique and Tiered Oracles models

```
Tiered Oracles model: each $v \in T_i$ broadcasts to $u \in T_{\geq i}$ its incident edges.
for $i = \lceil \log|V| \rceil - 1$ downto 0 do
    For each node $u \in T_{\leq i}$, each node $v \in T_i$ computes $\tilde{d}(v,u) \leftarrow \min \{ d_{\leq i}(v,u), \min_{w \in T_{>i}} \{ \tilde{d}(v,w) + d_{\leq i}(w,u) \} \}$.
    Congested Clique model: $v \in T_i$ sends to $u \in T_{\leq i}$, the value $\tilde{d}(v,u)$.
end for
```

The outline of our algorithm is as follows. We start by having each node $v \in T_i$ broadcast its incident edges to all the nodes in tiers greater than or equal to its own, that is, to all $u \in T_{\geq i}$, using one round of the Tiered Oracles model (Line 1). Afterwards, in the loop in Line 2, we compute the solution tier by tier, starting from the topmost tier, which contains nodes knowing all the edges in the graph. While processing the i-th tier, every node $v \in T_i$ already knows its distance to every node in $T_{>i}$, and so computes its distances to every node $u \in T_{\leq i}$. A shortest path between such v and u can either pass through edges which are all known to v, or be broken into a subpath from v to some node $w \in T_{>i}$ and then a path from w to u which is known to v. Thus, we compute the distance from $v \in T_i$ to the nodes $T_{\leq i}$ (Line 3). On Line 4, node $v \in T_i$, which knows for each node $u \in T_{\leq i}$ the distance to u, sends it to u.

For each $u, v \in V$, Algorithm 4.2 outputs a value $\tilde{d}(u, v)$. We show that it is the correct distance in G, that is $\tilde{d}(u, v) = d(u, v)$.

One round of the Tiered Oracles model suffices for ensuring that for each tier, T_i, every node $v \in T_i$ knows all the edges incident to all the nodes $u \in T_{\leq i}$. Let $v \in T_i$, and $u \in T_j$ such that $i \geq j$, observe that it holds that $\deg(v) \geq 2^i \geq \frac{1}{2} 2^j = \frac{1}{2} \deg(u)$, and therefore after Line 1 node v knows the edges incident to u. Thus, each node $v \in T_i$ knows enough information to compute the function $d_{\leq i}$, which is the distance function in G limited to edges incident to nodes in $T_{\leq i}$.

By induction on tier index i, we show that after iteration i of the loop in Line 2 all the nodes in V know the exact distances to all nodes in tiers $T_{\geq i}$.

Base case: In iteration $i = \lceil \log |V| \rceil - 1$, node $v \in T_{\lceil \log |V| \rceil - 1}$ (if exists) in the topmost tier knows about all the edges in E since it knows about all edges incident to nodes $T_{\leq \lceil \log |V| \rceil - 1} = V$. Thus, v can compute the solution to the entire APSP on G, since $d_{\leq \lceil \log |V| \rceil - 1} = d$. Since the set

$$\{\, d(v,w) + d_{\leq \lceil \log |V| \rceil - 1}(w,u) \,\}_{w \in T_{> \lceil \log |V| \rceil - 1}}$$

is empty, we get that $\tilde{d}(v,u) = d_{\leq \lceil \log |V| \rceil - 1}(v,u) = d(v,u)$. That is, node $v \in T_{\lceil \log |V| \rceil - 1}$ computes for each other node $u \in V$ its weighted distance $d(v,u)$ and sends it to u on Line 4.

Induction Step: In iteration $i < \lceil \log |V| \rceil - 1$, consider $v \in T_i$ and $u \in T_{\leq i}$, and let P be a shortest path between them. Recall that node v can locally compute $d_{\leq i}$, and thus knows the value $d_{\leq i}(v,u)$ and for each $w \in T_{>i}$, it knows the value $d_{\leq i}(w,u)$. Further, for each $w \in T_{>i}$, the value $\tilde{d}(v,w) = d(v,w)$ is known to v from one of the previous iterations of the loop in Line 2, by the induction assumption. All values in the set $\{\, \tilde{d}(v,w) + d_{\leq i}(w,u) \,\}_{w \in T_{>i}} \cup \{\, d_{\leq i}(v,u) \,\}$ are either infinite or correspond to some (not necessary simple) path from v to u, thus $\tilde{d}(v,u) \geq d(v,u)$. To show that $\tilde{d}(v,u) \leq d(v,u)$, we consider two cases. If P does not contain nodes from $T_{>i}$, then $d_{\leq i}(v,u) = d(v,u)$ is the length of P. Otherwise, let $w' \in T_{>i}$ be the last node on P (closest to u) which belongs to $T_{>i}$. By the induction hypothesis, v knows $\tilde{d}(v,w') = d(v,w')$. Moreover, the subpath from w' to u only contains edges with at least one endpoint incident to node in $T_{\leq i}$, thus $d_{\leq i}(w',u) = d(w',u)$. For this node w' the value $\{\, \tilde{d}(v,w') + d_{\leq i}(w',u) \,\}$ belongs to $\{\, \tilde{d}(v,w) + d_{\leq i}(w,u) \,\}_{w \in T_{>i}}$. Thus, in both cases the computed $\tilde{d}(v,u)$ is at most the weighted length of P. Hence, $\tilde{d}(v,u) = d(v,u)$. On Line 4, node v informs u about the correct $d(v,u)$, which completes the induction proof.

Lines 1 and 4 each take a single round of the Tiered Oracles model and the Congested Clique model, respectively, and thus the execution of the entire algorithm takes $O(\log |V|)$ rounds of the Congested Clique model and one round of the Tiered Oracles model. ■

By simulating the algorithm given in Lemma 4.3.2 *(Exact n^x-RSSP)* using Theorem 1.7 *(Simulating powerful models)* and Lemma 4.2.2 *(Model Simulation Preliminaries)*, we get exact APSP over the skeleton graph, as follows.

Corollary 4.1 (Exact APSP on Skeleton Graph). *For any constant $0 < x < 1$, there is an algorithm in the* Hybrid *model that computes an exact weighted APSP on a skeleton graph $S_x = (M, E_S)$, in $\tilde{O}(n^{1-x} + n^{2x-1})$ rounds w.h.p.*

Finally, we extend the result to n^x-RSSP on G, by having each node in the graph learn the information stored in the skeletons in its $\tilde{O}(n^{1-x})$ neighborhood.

Theorem 1.9 (Exact n^x-RSSP)**.** *Given a graph $G = (V, E)$, $0 < x < 1$, and a set of nodes M sampled independently with probability n^{x-1}, there is an algorithm in the* Hybrid *model that ensures that every $v \in V$ knows the exact, weighted distance from itself to every node in M within $\tilde{O}(n^{1/3} + n^{2x-1})$ rounds w.h.p.*

Proof. Primarily, assume that $x \geq \frac{2}{3}$. Otherwise, we add each node outside of M with probability $(n^{-1/3} - n^{x-1})/(1 - n^{x-1})$ into the set M. Thus, each node has probability exactly $(n^{x-1} \cdot 1) + (1 - n^{x-1}) \cdot (n^{-1/3} - n^{x-1})/(1 - n^{x-1}) = n^{-1/3}$ to be sampled into M, ensuring $x = 2/3$. We use Claim 4.1.9 *(Skeleton Graphs)* to build a skeleton graph $S_x = (M, E_S)$ in $\tilde{O}(n^{1/3})$ rounds w.h.p. Then, we compute exact APSP on the skeleton graph using Corollary 4.1 *(Exact n^x-RSSP)* in $\tilde{O}(n^{1/3} + n^{2x-1})$ rounds w.h.p. By Property 5 of the skeleton graph, for each $v, u \in M$ it holds that $d_S(v, u) = d(v, u)$, where $d_S(v, u)$ is the distance in the skeleton graph. So, we apply Claim 4.1.10 *(Skeleton Graphs)* with $\alpha = 1, \beta = 0$ and set of sources $V' = M$, to compute an exact weighted shortest paths distances, from M to all of V, in additional $\tilde{O}(n^{1/3})$ rounds w.h.p.

Instantiating Theorem 1.9 with $x = 2/3$ gives $n^{2/3}$-RSSP in $\tilde{\Theta}(n^{1/3})$ rounds w.h.p., which is tight due to our lower bound given in Theorem 1.10 *(Exact $n^{2/3}$-RSSP)*. We extensively use our $n^{2/3}$-RSSP algorithm for our following results.

4.3.3 Exact $n^{1/3}$-SSP

We now present an improvement over the warm-up exact SSSP algorithm which we showed previously, by providing an algorithm for exact shortest paths from a *given* set of $n^{1/3}$ nodes ($n^{1/3}$-SSP) in $\tilde{O}(n^{1/3})$ rounds. To do so, we create a skeleton graph and use our algorithm for $n^{2/3}$-RSSP algorithm to compute exact distances from the skeleton nodes to the entire graph. Then, we adapt the behavior of the source nodes depending on the number of skeleton nodes in their neighborhood (which is proportional to the density of the neighborhoods). That is, nodes in sparse neighborhoods can broadcast the distances from themselves to all the skeleton nodes which they see surrounding them, while a node in dense neighborhoods can take over a skeleton node surrounding it and use it as a proxy to communicate efficiently with the other skeleton nodes in the graph. We formalize this in this section, as well as Lemma 4.3.3 *(Exact $n^{1/3}$-SSP)* which is a generic tool which performs this action of *taking over* skeleton nodes as proxies.

We show the following fundamental algorithm, which allows *assigning* skeletons to help other skeletons. That is, given a set of nodes A where each node in A sees many skeleton nodes in its neighborhood, it is possible to assign skeleton nodes to service the nodes of A. We use this to increase sending and receiving *capacity* of the nodes of A. This is a key tool which we use in the proof of Theorem 1.11 *(Exact $n^{1/3}$-SSP)* and we believe it may be useful for additional tasks.

Lemma 4.3.3 (Reassign Skeletons)**.** *Given graph $G = (V, E)$, a skeleton graph $S_x = (M, E_S)$, a value k which is known to all the nodes, and nodes $A \subseteq V$ such that each $u \in A$ has at least $\tilde{\Theta}(k \cdot |A|)$ nodes $M_u \subseteq M$ in its $\tilde{\Theta}(n^{1-x})$ neighborhood, there is an algorithm that assigns $K_u \subseteq M_u$ nodes to u, where $|K_u| = \tilde{\Omega}(k)$, such that each node in M is assigned to at most $\tilde{O}(1)$ nodes in A. With respect to the set A, it is only required that every node in G must know whether or not it itself is in A – that is, the entire contents of A do not have to be globally known. The algorithm runs in $\tilde{O}(n^{1-x})$ rounds in the* Hybrid *model, w.h.p.*

Proof. The pseudocode is provided by Algorithm 4.3.

Algorithm 4.3 Reassign-Skeletons(A, k)

```
Compute $|A|$ by running Aggregate-And-Broadcast.
Skeleton node $v \in M$ learns its $\tilde{O}(n^{1-x})$-hop neighborhood.
Skeleton node $v \in M$ samples each $u \in A \cap N_G^{\tilde{\Theta}(n^{1-x})}(v)$ with probability $\frac{1}{|A|}$.
Skeleton node $v \in M$ informs each sampled node $u$ about $v \in K_u$.                ■
```

First, each node w learns the size of the set A by invoking Claim 4.1.5 *(Communication)* with value 1 if $w \in A$ and 0 otherwise, and the summation function (Line 1). Then, each skeleton node $v \in M$, learns its $\tilde{\Theta}(n^{1-x})$-hop neighborhood (Line 2), and in particular it learns the nodes $A_v = A \cap N_G^{\tilde{\Theta}(n^{1-x})}(v)$. Then, v samples each $u \in A_v$ independently with probability $\frac{1}{|A|}$ (Line 3). Afterwards, v informs each node u it sampled on the previous stage that $v \in K_u$ (Line 4).

For every $v \in M$, since $|A_v| \leq |A|$, and since v samples nodes from there A_v independently with probability $\frac{1}{|A|}$, by Chernoff Bounds each v assigns itself to at most $\tilde{O}(1)$ nodes $a \in A_v$ w.h.p. Hence, by a union bound over all skeleton nodes, each skeleton node is assigned to $\tilde{O}(1)$ nodes w.h.p.

For every $u \in A$, since it is sampled by at least $\tilde{\Omega}(k \cdot |A|)$ skeleton nodes independently with probability $\frac{1}{|A|}$, by Chernoff Bounds it is sampled by $|K_u| = \tilde{\Omega}(1)$ skeleton nodes w.h.p. Thus, by union bound over all skeleton nodes, each $u \in A$ has $|K_u| = \tilde{\Omega}(1)$ assigned nodes w.h.p.

By Claim 4.1.5 *(Communication)*, Line 1 takes $\tilde{O}(1)$ rounds w.h.p., and Lines 2 and 4 take $\tilde{O}(n^{1-x})$ rounds, and thus the entire execution completes in $\tilde{O}(n^{1-x})$ rounds w.h.p.

Now we apply Claim 4.1.7 *(Communication)* or Lemma 4.3.3 *(Exact $n^{1/3}$-SSP)* depending on density of each source's neighborhood and show how to compute exact $n^{1/3}$-SSP in $\tilde{O}(n^{1/3})$ rounds. For sources in "sparse" neighborhoods, in which there is a small number of skeleton nodes, we use Claim 4.1.7 *(Communication)* to inform all nodes about their distances to those skeletons. For source v with "dense" neighborhood, in which there are many skeleton nodes, we use Lemma 4.3.3 *(Exact $n^{1/3}$-SSP)* to get at least one skeleton node u which participates in the round of the Congested Clique model on behalf of that source and sends each other skeleton node v' the distance $d(v, v')$.

Theorem 1.11 (Exact $n^{1/3}$ Sources Shortest Paths)**.** *Given a weighted graph $G = (V, E)$, and a set of sources U, such that $|U| = O(n^{1/3})$, there exists an algorithm, at the end of which each $v \in V$ knows its distance from every $s \in U$, which runs in $\tilde{O}(n^{1/3})$ rounds w.h.p.*

Proof. The pseudocode for the algorithm appears in Algorithm 4.4.

Algorithm 4.4 Exact-$n^{1/3}$-SSP: Computes an exact weighted $n^{1/3}$-SSP. Routine for node $u \in V$

```
Join $M$ independently with probability $n^{-1/3}$.
Compute $n^{2/3}$-RSSP from $M$.
Construct skeleton graph $S_{2/3} = (M, E_S)$.
Learn $h = \tilde{\Theta}(n^{1/3})$-hop neighborhood.
if $u \in U$ then
    if $|N_G^h(u) \cap M| = \tilde{O}(n^{1/3})$ then
        Participate in Token-Disseminationwith a token $\langle u, v', d(v', u)\rangle$ for each $v' \in M \cap N_G^h(s)$.
    else
        $K_u \leftarrow$Reassign-Skeletons$\Big(\Big\{\ u\colon |N_G^h(u) \cap M| = \tilde{\Omega}(n^{1/3})\ \Big\}, \tilde{O}(1)\Big)$.
        Send each $v \in K_u$ the values $d(u, v')$ for each $v' \in M$.
    end if
end if
if $u \in M$ then
    In the CONGESTED CLIQUE model: for each $v' \in M$ and each $v \in U$ such that $u \in K_v$ send $d(v, v')$ to $v'$.
    For each $s \in U$, compute $\tilde{d}(u, s)$ by Eq. (4.2) and output it.
end if
Apply Claim 4.1.10, which given distances from skeleton to sources $\tilde{d}\colon M \times U \mapsto \mathbb{N}$ extends it to distances from each nodes to sources $\tilde{d}\colon V \times U \mapsto \mathbb{N}$.
```

Without loss of generality the set of nodes U is globally known (it can be disseminated in $\tilde{O}(n^{1/6})$ rounds w.h.p. using Claim 4.1.7 *(Communication)*). We build $M \subseteq V$ by marking nodes independently with probability $n^{-1/3}$ (Line 1). Then we run the algorithm from Theorem 1.9 *(Exact $n^{2/3}$-RSSP)* with $x = 2/3$ to obtain w.h.p. $n^{2/3}$-RSSP from the set of nodes M (Line 2), such that w.h.p. every $u \in V$ knows its distance to every node in M. Afterwards, we apply Claim 4.1.9 *(Skeleton Graphs)* to construct a skeleton graph $S_{2/3} = (M, E_S)$ w.h.p. Then, each source learns the information in its h-hop neighborhood (Line 4), for $h \in \tilde{\Theta}(n^{1/3})$. In particular, it counts the skeleton nodes in its h-hop neighborhood.

If a source finds that the number of skeleton nodes in its h-hop neighborhood is $\tilde{O}(n^{1/3})$, then it participates in a token dissemination protocol (Claim 4.1.7 *(Communication)*) and w.h.p. informs all the graph about its distance to these skeleton nodes (Line 7).

Otherwise, each source $u \in U$ which finds that there are at least $\tilde{\Omega}(n^{1/3})$ skeleton nodes in its h-hop neighborhood, applies Lemma 4.3.3 *(Exact $n^{1/3}$-SSP)* with $k = \tilde{O}(1)$ and $A = \{ u \colon |N_G^h(u) \cap M| = \tilde{\Omega}(n^{1/3}) \}$ and receives $K_u \subseteq N_G^h(u) \cap M$, a set of $\tilde{\Omega}(1)$ skeletons. Such a source $u \in U$ sends by local edges to $v \in K_u$ the distance $d(u, v')$ to each $v' \in M$ (Line 10). Each skeleton node $u \in M$ sends the distances $d(s, v')$ to each $v' \in M$, for each source node $s \in U$ that it is assigned to, by simulating the Congested Clique model. If skeleton node $u \in M$ for source $s \in U$ did not receive the distance from s, it computes it using Eq. (4.2) (Line 15) based on the information it received in Line 7.

$$\tilde{d}(u,s) = \min \{ d^h(u,s), \min_{v' \in M} \{ d(u,v') + d^h(v',s) \} \}^4 \tag{4.2}$$

After each skeleton knows the distance to each source, we apply Claim 4.1.10 to compute distances from sources to all the nodes.

Lemma 4.3.4. *After Line 15, each $u \in M$ knows $d(v,s)$ for each $s \in U$ w.h.p.*

Proof. Lemma 4.3.4 Let $u \in M, s \in U$. To show the claim we split into the two cases, by the number of sources in h-hop neighborhood of s. If the neighborhood is dense, meaning $|N_G^h(s) \cap M| = \tilde{\Omega}(n^{1/3})$, then s knows $d(s,u)$ w.h.p. after Line 2 by Theorem 1.9 *(Exact $n^{2/3}$-RSSP)* and sends it to u in Line 14 by Lemma 4.3.3 *(Exact $n^{1/3}$-SSP)* and Lemma 4.2.2 *(Model Simulation Preliminaries)* In this case Eq. (4.2) gives the exact answer for $v' = u$ w.h.p. Otherwise, the distance is computed by Eq. (4.2).

First, we show that node $u \in M$ on Line 15 can compute Eq. (4.2). From the local exploration in Line 4 u learns $d^h(u,s)$. After Line 2 u, w.h.p., knows for each $v' \in M$ $d(u,v')$ and due to Claim 4.1.7 *(Communication)*. In Line 7 it receives $d(v',s)$ w.h.p.

Now we prove that the value $\tilde{d}(u,s)$, computed by Eq. (4.2), w.h.p. equals $d(v,s)$. Each element in $\{ d^h(u,s) \} \cup \{ d(u,v') + d^h(v',s) \}_{v' \in M}$ is either infinite, or corresponds to some (not necessary simple) path, thus, since the graph has non-negative weights, it holds that $\tilde{d}(u,s) \geq d(u,s)$. We show that $\tilde{d}(u,s) \leq d(u,s)$ by considering two cases. If shortest path between u and s has less than h hops, then $d(u,s) = d^h(u,s)$ appears as an argument to the set over which we take the minimum and thus $\tilde{d}(u,s) \leq d^h(u,s) = d(u,s)$. Otherwise, by Property 6 w.h.p. there exists a shortest path between u and s such that there is a node $v'' \in M$ in one of its h last (closest to s) nodes.Thus, the corresponding element $d(u,v'') + d^h(v'',s) = d(u,s)$ appears in $\{ d(u,v') + d^h(v',s) \}_{v' \in M}$ for $v' = v''$. Therefore, in both cases $\tilde{d}(u,s) \leq d(u,s)$. ■

We continue the proof of Theorem 1.11. By Lemma 4.3.4, each node in M knows the

[4]Note that difference from Eq. (4.1). There node u does not know precise distance to skeleton, but only h-limited distance, while here it knows the precise distance after Line 2. Similarly with the $d^h(v',s)$ term.

distance to each node in U, thus by Claim 4.1.10 *(Skeleton Graphs)* with $\alpha = 1, \beta = 0$ there is an algorithm to compute shortest paths distance from U.

By Theorem 1.9 *(Exact $n^{2/3}$-RSSP)* with $x = \frac{2}{3}$, Line 2 completes in $\tilde{O}(n^{1/3})$ rounds w.h.p. For Line 3, by Claim 4.1.9 *(Skeleton Graphs)*, the round complexity, w.h.p., is $\tilde{O}(n^{1/3})$ as well. Line 4 completes in $\tilde{O}(n^{1/3})$ rounds. Since there are at most $\ell = \tilde{O}(n^{1/3})$ tokens per source and $k = \Omega(n^{2/3})$ tokens overall, Line 7 takes $\tilde{O}(n^{1/3})$ rounds w.h.p. by Claim 4.1.7 *(Communication)*. By Lemma 4.3.3 *(Exact $n^{1/3}$-SSP)*, Line 9 takes $\tilde{O}(n^{1/3})$ rounds w.h.p. All skeleton nodes are assigned to some helpers in their $\tilde{O}(n^{1/3})$-hop neighborhood by Line 9, so Line 4 takes $\tilde{O}(n^{1/3})$ rounds. Since each skeleton selects $\tilde{O}(1)$ sources w.h.p. in Line 9 by Lemma 4.3.3 *(Exact $n^{1/3}$-SSP)*, Line 14 simulates $\tilde{O}(1)$ rounds of the Congested Clique model and takes $\tilde{O}(n^{1/3})$ rounds by Lemma 4.2.2 *(Model Simulation Preliminaries)* w.h.p. Finally by Claim 4.1.10 *(Skeleton Graphs)*, Line 17 for $x = \frac{2}{3}$ takes $\tilde{O}(n^{1/3})$ rounds as well. Thus, the overall execution of the algorithm takes $\tilde{O}(n^{1/3})$ rounds. ■

4.3.4 Approximating n^x-SSP from a Given Set of Sources

We previously showed how to approximate distances to a set of independently, uniformly, randomly chosen sources, and here we leverage this to the case of a *given* set of sources, which is the case we care more about. We show an algorithm for approximating distances from a given set of sources, which is *tight*, matching the lower bound of [KS20], when the number of sources is at least $\Omega(n^{2/3})$: given $O(n^x)$ sources, our algorithm takes $\tilde{O}(n^{1/3}/\epsilon + n^{x/2})$ rounds for a $(1+\epsilon)$-approximation in unweighted graphs, and a 3-approximation in weighted graphs.

The proof of the following, appears as a part of [KS20, Theorem 4.2], which shows how to obtain distance approximation algorithms in the Hybrid model from distance approximation algorithms in the Congested Clique model. A similar approach is also used in the proof of [AHK+20, Theorem 2.3]. We find this tool useful in general in order to obtain, given some algorithm which approximates distances on a skeleton graph, an algorithm for approximating distances from a set of nodes to the entire graph, and hence we extract it here from the proof of [KS20, Theorem 4.2].

Claim 4.3.5 (n^x-SSP from APSP on Skeleton Graph)**.** *[KS20, Theorem 4.2] Consider a graph $G = (V, E)$, its skeleton graph $S_x = (M, E_x)$, for some constant $0 < x < 1$ and a set of sources U, where $|U| = \tilde{\Theta}(n^y)$, for some constant $0 < y < 1$. Let A_x be a T-round algorithm in the* Hybrid *model, such that given a skeleton graph S_x, A_x ensures that for every pair $v, v' \in M$, both v and v' know an (α, β)-approximation for the distance between them in G.*

Then, for any value $\eta \geq 1$, there is an algorithm B for the Hybrid *model which takes $\tilde{O}(T + n^{\frac{y}{2}} + \eta \cdot n^{1-x})$ rounds over G, and ensures that for every $v \in V, s \in U$, node v knows an approximation for the distance from v to s in G, where the approximation factor is $(2\alpha + 1, \beta)$ if G is weighted, and $\left(\alpha + \frac{2+\alpha}{\eta}, \beta\right)$ if G is unweighted.*

Proof. We follow the proof of [KS20, Theorem 2.2]. We apply A_x on S, and denote by $\tilde{d}^S$ the computed (α, β)-approximation for the distances amongst the set of nodes M. Afterwards, nodes in V learn their $(\eta \cdot h)$-hop neighborhoods, where $h = \tilde{O}(n^{1-x})$. As in [KS20, Theorem 4.2], each source $s \in U$ selects its *representative* – the closest node $r_s \in M$ found in its h-hop neighborhood – and participates in a token dissemination protocol with the token $\left\langle r_s, d^h(r_s, s)\right\rangle$. Each node $u \in V$, for each source $s \in U$, outputs $\tilde{d}(u, s) = \min \{ d^{\eta h}(u, s), \min_{v' \in M \cap N_G^h(u)} \{ d^h(u, v') + \tilde{d}(v', r_s) \} + d^h(r_s, s) \}$.

Each node $u \in V$ knows $\tilde{d}(v', r_s)$ since it is an output of the algorithm A_x for nodes $v' \in M \cap N_G^h(u)$ and $u \in V$ learns this information from each v' in its h-hop neighborhood. Node u also knows $d^h(r_s, s)$ w.h.p., due to Claim 4.1.7 *(Communication)*.

There are at most $\tilde{O}(n^y)$ source nodes w.h.p. due to Property 7, so the token dissemination protocol with $k = \tilde{O}(n^y)$ and $\ell = 1$ takes $\tilde{O}(\sqrt{n^y}) = \tilde{O}(n^{\frac{y}{2}})$ rounds w.h.p., by Claim 4.1.7 *(Communication)*. Overall, the number of rounds is $\tilde{O}(T + n^{\frac{y}{2}} + \eta \cdot n^{1-x})$, as needed.

To show the approximation, first notice that the algorithm does not underestimate the distances, since each value in the set over which the minimum is taken is either infinite or corresponds to the approximate weight of a not necessarily simple path. This also implies that in case shortest path between u and s has less than ηh hops, it holds that $\tilde{d}(u, s) = d^{\eta h}(u, s) = d(u, v)$, so we assume without loss of generality that shortest path between u and v at at least h hops and also $d(u, s) \geq hop(u, s) > \eta \cdot h$.

Take two nodes $u \in V, s \in U$, by Property 6, there is a shortest path between u and s such that there is a node $u' \in M$ in the first h nodes (from the u-side) and a node $v'' \in M$ in the last h nodes (from the s side). Let $v' = \arg\min_{v' \in M} \{ d^h(u, v') + \tilde{d}(v', r_s) \}$.

We upper bound $\tilde{d}$ for the weighted case:

$$\begin{aligned}
\tilde{d}(u, s) &\leq \left(d^h(u, v') + \tilde{d}(v', r_s)\right) + d_h(r_s, s) \\
&\leq \left(d(u, u') + \tilde{d}(u', r_s)\right) + d(v'', s) \\
&\leq (d(u, u') + d(v'', s)) + \alpha \cdot d(u', r_s) + \beta \\
&\leq d(u, s) + \alpha \cdot (d(u', s) + d(s, r_s)) + \beta \\
&\leq d(u, s) + \alpha \cdot (d(u, s) + d(v, v'')) + \beta \\
&\leq (2\alpha + 1) \cdot d(u, s) + \beta,
\end{aligned}$$

where the first transition is due to the computation of $\tilde{d}(u, s)$, the second is implied by the choice of v' and the representative r_s, the third follows from the definition of an (α, β)-approximation of the distances and Property 5 of the skeleton graph, the forth is due to the non-negative weights and the triangle inequality, the fifth is due to the non-negative weights and the choice of the representative r_s, and the last one is due to the non-negative weights. Also, we may use the fact that $d(u, v) \geq \eta \cdot h$ and obtain a purely multiplicative approximation factor of $2\alpha + 1 + \frac{\beta}{\eta \cdot h}$.

However, for the unweighted case we can upper bound $\tilde{d}$ slightly better:

$$\begin{aligned}\tilde{d}(u,s) &\leq \Big(d(u,v') + \tilde{d}^S(v',r_s)\Big) + d(r_s,s)\\ &\leq \Big(d(u,u') + \tilde{d}^S(u',r_s)\Big) + d(r_s,s)\\ &\leq 2\cdot h + \alpha\cdot(d(u',s)+h) + \beta\\ &< (2+\alpha)\cdot\frac{d(u,s)}{\eta} + \alpha\cdot d(u,s) + \beta,\end{aligned}$$

where the first transition is due to the computation of $\tilde{d}(u,s)$, the second is implied by the choice of v', the third follows from u' and r_s being in h-hop neighborhood of u and s, respectively, and the definition of a (α,β)-approximation, and the forth is due to the assumption that $d(u,s) > \eta\cdot h$. Again, using the fact that $d(u,v)\geq\eta\cdot h$, we obtain a purely multiplicative approximation factor of $\alpha + \frac{2+\alpha}{\eta} + \frac{\beta}{\eta\cdot h}$. ■

We use the claim above to compute our tight approximation of the n^y-SSP.

Theorem 1.12 (Approximate Multiple Source Shortest Paths)**.** *Given a graph $G = (V,E)$, a set of sources U, where $|U| = \tilde{\Theta}(n^y)$ for some constant $0 < y < 1$, and a value $0 < \epsilon$, there is an algorithm in the* Hybrid *model which ensures that every node $v \in V$ knows an approximation to its distance from every $s \in U$, where the approximation factor is $(1+\epsilon)$ if G is unweighted and 3 if G is weighted. The complexity of the algorithm is $\tilde{O}(n^{1/3}/\epsilon + n^{y/2})$ rounds, w.h.p.*

Proof. We use $x = 2/3$ and let $A_{2/3}$ be the algorithm from Corollary 4.1 *(Exact n^x-RSSP)*. Notice that Corollary 4.1 *(Exact n^x-RSSP)* expects a skeleton graph and not a random set M, yet it is possible to convert M to a skeleton graph using Claim 4.1.9 *(Skeleton Graphs)* in $\tilde{O}(n^{1/3})$ rounds w.h.p. Further, Corollary 4.1 *(Exact n^x-RSSP)* computes the distances between the nodes of M over a skeleton graph and not over G, as required of $A_{2/3}$ in Claim 4.3.5 *(Approximating n^x-SSP from a Given Set of Sources)*, yet, this is equivalent due to Property 5 in the definition of a skeleton graph. Using $A_{2/3}$, the rest of the proof follows directly from Claim 4.3.5 *(Approximating n^x-SSP from a Given Set of Sources)*, with $\alpha = 1, \beta = 0, \eta = (2+\alpha)/\epsilon, x = 2/3, T = \tilde{O}(n^{1/3})$ in $\tilde{O}(n^{1/3} + n^{y/2} + n^{1/3}/\epsilon) = \tilde{O}(n^{1/3}/\epsilon + n^{y/2})$ rounds leading to the approximation factor of $(2\cdot 1 + 1 = 3, 0) = (3,0)$ for weighted graphs and $(1 + 2/(2/\epsilon), 0) = (1+\epsilon, 0)$. ■

Notice, that for $y \geq \frac{2}{3}$, the complexity is $\tilde{\Theta}(n^{\frac{y}{2}})$, which is tight due to the lower bound of [KS20, Theorem 1.5].

4.3.5 Approximations of Eccentricities

Lemma 4.3.6 ($(1+\epsilon)$-Approx. Unweighted Eccentricities)**.** *Let $G = (V,E)$ be an unweighted graph, and let $\epsilon > 0$. Then there exists an algorithm in the* Hybrid *model which, for each v, computes a $(1+\epsilon)$-approximation of unweighted $ecc(v)$ in $\tilde{O}(n^{1/3}/\epsilon)$ rounds w.h.p.*

Proof. We run the algorithm from Theorem 1.9 *(Exact $n^{2/3}$-RSSP)* with $x = 2/3$ to obtain $n^{2/3}$-RSSP from a set of nodes M selected independently randomly with probability $n^{-1/3}$ from V, such that every $v \in V$ knows its distance to every node in M. This takes $\tilde{O}(n^{1/3})$ rounds. Then, each node v learns its $\left(1 + \frac{1}{\epsilon}\right) h$-hop neighborhood, where $h = \tilde{\Theta}(n^{1/3})$, in $\tilde{O}(n^{1/3}/\epsilon)$ rounds. The approximate eccentricity $\widetilde{ecc}(v)$ of node v is the maximum between the distance to the farthest node in M from v, and distance to the farthest node in the $\left(1 + \frac{1}{\epsilon}\right) h$-hop neighborhood of v.

We now prove the approximation factors. Notice, that there exists a node $w \in V$ such that $\widetilde{ecc}(v) = d(v, w) \leq ecc(v)$, so we do not overestimate the eccentricity. If $ecc(v) \leq \left(1 + \frac{1}{\epsilon}\right) h$, the farthest node is in the explored neighborhood and the returned $\widetilde{ecc}(v)$ is exact in this case. Let u be farthest node from v. By Claim 4.1.9 and Property 6 there exists a node $w \in M$ on some shortest path between v and u within hop-distance at most h from u w.h.p. By definition, $\widetilde{ecc}(v) \geq d(v, w) \geq ecc(v) - h \geq ecc(v) - \frac{ecc(v)}{1+\frac{1}{\epsilon}} = \left(1 - \frac{1}{\frac{\epsilon+1}{\epsilon}}\right) ecc(v) = \left(1 - \frac{\epsilon}{1+\epsilon}\right) ecc(v) = \left(\frac{1+\epsilon-\epsilon}{1+\epsilon}\right) ecc(v) = \frac{1}{1+\epsilon} ecc(v)$. ■

Lemma 4.3.7 (3-Approx. Weighted Eccentricities)**.** *Let $G = (V, E)$ be a weighted graph. There is an algorithm in the* HYBRID *model that computes a 3-approximation of weighted eccentricities in $\tilde{O}(n^{1/3})$ rounds, w.h.p.*

Proof. We run the algorithm from Theorem 1.9 *(Exact $n^{2/3}$-RSSP)* with $x = 2/3$ to obtain $n^{2/3}$-RSSP from a set of nodes M selected independently randomly with probability $n^{-1/3}$ from V, such that every $u \in V$ knows its distance to every node in M. Then, each $v \in M$ learns its h-hop neighborhood, where $h = \tilde{\Theta}(n^{1/3})$, and we use token dissemination, Claim 4.1.7 *(Communication)*, to ensure that all the nodes in the graph know $ecc_h(v) = \max_{w \in N_G^h(v)} \{ d(v, w) \}$ for each $v \in M$. The approximate eccentricity $\widetilde{ecc}(u)$ of node $u \in V$, is the value of Eq. (4.3).

$$\widetilde{ecc}(u) = \frac{1}{3} \max_{v \in M} \{ d(u, v) + ecc_h(v) \} \tag{4.3}$$

To show the approximation factor for some $u \in V$, let $v' = \arg\max_{v \in M} \{ d(u, v) + ecc_h(v) \}$ and $v'' \in N_G^h(v)$ be the farthest node from v' in its h-hop neighborhood, that is $ecc_h(v') = d(v', v'')$. Further, let $p \in V$ be the farthest node from $u \in V$, in other words, $ecc(u) = d(u, p)$, and $p' \in M$ be some marked node in the h-hop neighborhood of p, which exists w.h.p. by Chernoff bound. Finally, let $p'' \in V$ be the farthest node from p' in its h-hop neighborhood, that is, $ecc_h(p') = d(p', p'')$. Notice that $\widetilde{ecc}(u) = \frac{d(u,v')+d(v',v'')}{3} \geq \frac{d(u,p')+d(p',p'')}{3} \geq \frac{d(u,p')+d(p',p)}{3} \geq \frac{d(u,p)}{3} = \frac{ecc(u)}{3}$, where the first inequality is due to the choice of v', the second inequality follows from the choice of p'' and the third is implied by triangle inequality. Also, we do not overestimate the eccentricity, since $\widetilde{ecc}(u) = \frac{d(u,v')+d(v',v'')}{3} \leq \frac{d(u,v')+d(v',u)+d(u,v'')}{3} \leq ecc(u)$, where the first inequality is due to triangle inequality and the second is due to the graph being undirected and due to the definition of eccentricity.

By Theorem 1.9 *(Exact $n^{2/3}$-RSSP)* the round complexity of the $n^{2/3}$-RSSP is $\tilde{O}(n^{1/3})$ w.h.p. and token dissemination with a total of $k = |M| = \tilde{O}(n^{2/3})$ tokens w.h.p. (by Chernoff bound) and $\ell = 1$ tokens per node takes $\tilde{O}(n^{1/3})$ rounds w.h.p. due to Claim 4.1.7 *(Communication)*. ■

Combining Lemma 4.3.6 *(Approximations of Eccentricities)* and Lemma 4.3.7 *(Approximations of Eccentricities)* gives Theorem 1.13 *(Approximate eccentricities and diameter)*.

4.3.6 Diameter Approximations

Corollary 4.2 ($(1+\epsilon)$-Approx. Unweighted Diameter)**.** *Let $G = (V, E)$ be an unweighted graph, and let $\epsilon > 0$. There exists an algorithm in the* Hybrid *model which computes a $(1+\epsilon)$-approximation of the diameter in $\tilde{O}(n^{1/3}/\epsilon)$ rounds, w.h.p.*

Proof. To achieve this result we first use the algorithm from Lemma 4.3.6 *(Approximations of Eccentricities)* to compute $(1+\epsilon)$-approximate eccentricities, for each $v \in V$ a value $\widetilde{ecc}(v)$, in $\tilde{O}(n^{1/3}/\epsilon)$ rounds, and then use Claim 4.1.5 *(Communication)* to compute the maximum between the approximate eccentricities $\widetilde{ecc}(v)$ in an additional number of $\tilde{O}(1)$ rounds. Notice that the maximum of $(1+\epsilon)$-approximate eccentricities is indeed a good approximation of the diameter, as

$$\widetilde{D} = \max\left\{\, \widetilde{ecc}(v) \,\right\} \geq \max\left\{ \frac{ecc(v)}{1+\epsilon} \right\} = \frac{\max\left\{\, ecc(v) \,\right\}}{1+\epsilon} = \frac{D}{1+\epsilon}.$$

On the other hand, since the approximate eccentricities $\widetilde{ecc}(v)$ do not overestimate the real eccentricities, their maximum does not over overestimate the true diameter. Thus, $\widetilde{D} = \max\left\{\, \widetilde{ecc}(v) \,\right\} \leq \max\left\{\, ecc(v) \,\right\} = D$, which completes the proof. ■

Our oracle-based techniques allow us to solve weighted SSSP fast. After we do it, the following well-known simple reduction allows us to compute 2 approximate weighted diameter.

Claim 4.3.8 (Diameter From SSSP)**.** *Given a graph $G = (V, E)$, a value $\alpha > 0$ and an algorithm which computes an α approximation of weighted SSSP in T rounds of the* Hybrid *model, there is an algorithm which computes a 2α-approximation of the weighted diameter in $T + \tilde{O}(1)$ rounds of the* Hybrid *model.*

Proof. We compute SSSP from an arbitrary node s. Then, each node v proposes its candidate for the diameter $\frac{1}{\alpha}\tilde{d}(v, s)$, which it computes after SSSP. The approximation is the maximum of all the proposals $\tilde{D} = \max_{u \in V} \frac{1}{\alpha}\tilde{d}(v, s)$, and us computed using Claim 4.1.5 *(Communication)*.

Since for each $v \in V$, it holds that $\tilde{d}(v, s) \leq \alpha d(v, s)$ is a shortest path between v and s, $D \geq \tilde{D} = \max_{u \in U} \frac{1}{\alpha} d(v, s)$. Let u and v be the endpoints of some path whose length is the diameter (i.e., $D = d(u, v)$), by the triangle inequality, $d(u, s) + d(s, v) \geq$

$d(u,v)$. Thus, at least one of the terms is greater then $\frac{d(u,v)}{2}$, assume without loss of generality that $d(u,s) \geq \frac{d(u,v)}{2}$. Since $\tilde{d}(u,s) \geq d(u,s)$ and the proposal of u is at least $\frac{1}{\alpha}\tilde{d}(u,s) \geq \frac{1}{2\alpha}d(u,s)$, it holds that $\tilde{D} \geq d(u,s) \geq \frac{d(u,v)}{2} = \frac{D}{2\alpha}$.

The round complexity for computing the SSSP approximation is T, and the round complexity for the aggregation operation is $\tilde{O}(1)$ by Claim 4.1.5 *(Communication)*. ■

Corollary 4.3 (2-Approx. Weighted Diameter)**.** *There is an algorithm in the* Hybrid *model that computes a* 2*-approximation of weighted diameter in* $\tilde{O}(n^{1/3})$ *rounds w.h.p.*

Proof. The claim follows immediately from Theorem 1.8 *(Exact SSSP)* and Claim 4.3.8 *(Diameter Approximations)* with $\alpha = 1$. ■

4.4 A Lower Bound for n^x-RSSP

We use the following Claim 4.4.1 *(A Lower Bound for n^x-RSSP)* to obtain our lower bound.

Claim 4.4.1 (Lower Bound Random Variable)**.** *[AHK+20, Lemma 4.12] Let $G = (V,E)$ be an n-node graph that consists of a subgraph $G' = (V',E')$ and a path of length L (edges) from some node $a \in V \setminus V'$ to $b \in V'$ and that except for node b is node-disjoint from V'. Assume further that the nodes in V are collectively given the state of some random variable X and that node a needs to learn the state of X. Every randomized algorithm that solves this problem in the* Hybrid *network model requires $\Omega(\min\{L, \frac{H(X)}{L\cdot\log^2 n}\})$ rounds, where $H(X)$ denotes the Shannon entropy of X.*

Theorem 1.10 (Lower Bound RMSSP)**.** *Let $p = \Omega(\log n/n)$ and $\alpha < \sqrt{n/p}\cdot\log(n)/2$. Any α-approximate algorithm for computing unweighted shortest paths from random sources sampled independently with probability p in the* Hybrid *network model takes $\Omega(\sqrt{p\cdot n}/\log n)$ rounds w.h.p.*

Proof. We use the same base construction as in [AHK+20, Theorem 2.5] and in [KS20, Theorem 1.5] with only slight modifications. However, for the sake of self-containment, we recall here the entire construction.

We construct an unweighted graph $G = (V,E)$ which contains a path, with end-points at nodes a and c, and two *fooling* sets of nodes S_b, S_c, both of size $y = \lfloor \frac{n}{4} \rfloor$. Every node in S_c has an edge to c. At hop-distance $L = \left\lfloor \frac{\sqrt{p\cdot n}}{\log n} \right\rfloor \geq 1$ from a there is a node b. Each node in S_b has an edge to b. The segment of path between b and c contains all other nodes, thus the hop distance between b and c is $z = n - 2y - L - 1$, and the hop distance between a and c is $x = z + L = n - 2y - 1$. See Figure 4.1.

We follow the scheme of [AHK+20, Theorem 2.5] and reserve $2y$ identifiers from the set $[2y]$ for nodes in $S = S_b \cup S_c$ and assign additional $n - 2y$ identifiers to other nodes $V \setminus S$ in a globally known manner. The nodes from $S = S_b \cup S_c$ are assigned identifiers

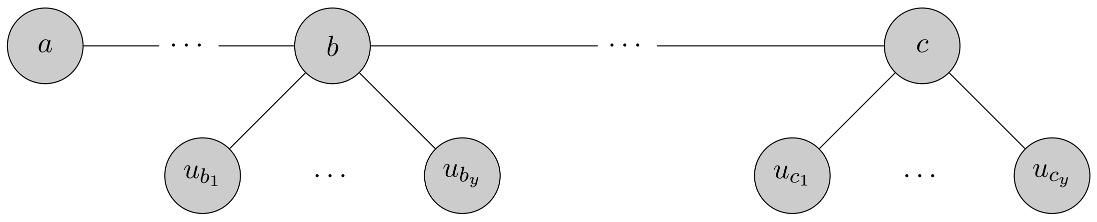

Figure 4.1: Lower bound construction.

randomly from the reserved set $[2y]$, as follows. Each identifier from $[y]$ (the first half of identifiers) is assigned with probability $1/2$ to the next currently unlabeled node in S_b and with probability $1/2$ to the next currently unlabeled node in S_c. The other y reserved identifiers are arbitrarily assigned to nodes left unlabeled in S.

Let S' be the set of nodes which are sampled to be sources. Consider the set $S'' = [y] \cap S'$ of sources with identifiers in $[y]$. For $1 \leq i \leq y$ we define X_i to be an indicator of whether identifier i is one of the sampled source identifiers. Notice that in expectation there are $E\left[\sum_{i=1}^{y} X_i\right] = p \cdot y = p\lfloor \frac{n}{4} \rfloor$ sampled nodes with identifiers in $[y]$. Moreover, for a large enough n, since $p = \Omega(\frac{\log n}{n})$, by Chernoff Bounds, there exists a constant $\beta > 0$ such that w.h.p. $\sum_{i=1}^{y} X_i \geq \beta p \cdot n$, which means that there are at least $\beta \cdot p \cdot n = \Omega(p \cdot n)$ source nodes in S''. These may be in S_b or S_c.

By definition, an α-approximation of distances from sources S' is a function $\widetilde{hop}\colon (V, S') \mapsto \mathbb{Z}$ which satisfies for every $v \in V, u \in S'$: $hop(v,u) \leq \widetilde{hop}(v,u) \leq \alpha hop(v,u)$, where $hop(u,v)$ denotes the hop distance between u and v. In particular, the node $a \in V$ should compute the values $\widetilde{hop}\,(a,\cdot)$, which satisfy for every $u \in S''$: $hop(a,u) \leq \widetilde{hop}(a,u) \leq \alpha hop(a,u)$. If $u \in S_c$ then $hop(a,u) = x+1$, otherwise, if $u \in S_b$ then $hop(a,u) = L+1$. Thus, if a does not know whether $u \in S_c$ or $u \in S_b$, it has to be on the safe side and return $\widetilde{\delta}\,(a,u) = x+1 \geq L+1$. The approximation factor for $u \in S_b$ is

$$\alpha = \frac{x+1}{L+1} = \frac{n-2y}{L+1} = \frac{n - 2\lfloor \frac{n}{4} \rfloor}{\frac{\sqrt{pn}}{\log n}} \geq \frac{1}{2}\sqrt{\frac{n}{p}}\log n.$$

Given S', let $Y \in \{\, b, c \,\}^{|S''|}$ be a random variable indicating for each sampled node S'' whether it belongs to S_b or S_c. As in the proof of [AHK+20, Theorem 2.5], to get any better approximation, node a has to learn for each node $s \in S''$ whether it is in S_b or S_c. For this a has to learn the random variable Y whose entropy is w.h.p.

$$H\,(Y) = \sum_{q \in \{\, b,c \,\}^{|S''|}} 2^{-|S''|} \log\left(2^{-|S''|}\right) = \Omega(|S''|) = \Omega(p \cdot n).$$

By Claim 4.4.1 *(A Lower Bound for n^x-RSSP)*, $\Omega(\frac{\sqrt{p \cdot n}}{\log n})$ rounds are required for a to learn the variable Y w.h.p.

Chapter 5

Conclusion and open questions

In this thesis, we focused on the impact of global edges on distributed computations. We worked with well-known Congested Clique [LPPP05] and novel Hybrid [AHK$^+$20] models. Overall, it is important to notice, that global edges significantly reduce the round complexity for global problems. For example, the optimal algorithm for SSSP in the Local model requires $\Omega(D)$ rounds, where diameter of the graph D could be as large as n. Adding global edges allows to solve $n^{1/3}$-SSSP in only $\tilde{O}(n^{1/3})$ rounds w.h.p. For the Congest model, solving scheduling problem requires $\Omega(\mathsf{congestion} + \mathsf{dilation})$ rounds for any set of problems. Meanwhile, in the Congested Clique model, we obtain algorithms which solve the scheduling problem in time close to $O(\mathsf{GlobalCongestion} + \mathsf{dilation})$, where $\mathsf{GlobalCongestion}$ for a set of jobs could be up to n^2 times lower than a $\mathsf{congestion}$.

5.1 Congested Clique open questions

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 Congested Clique model. Our interest in obtaining protocols with fast amortized complexities stems from the growing number of problems which admit $O(1)$-round Congested 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 1.4 shows that running $t = \operatorname{poly} n$ 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.

Question 5.1.1. What is the *amortized* round complexity of various problems in the Congested Clique model?

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 1.5 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 1.1 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 1.1 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.

Question 5.1.2. How many instances of the MST, Coloring and Min Cut could be solved in $O(1)$ rounds in the Congested Clique model?

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.

5.2 Hybrid open questions

We show an algorithm to compute $1+\epsilon$ approximate unweighted diameter and 2 approximate weighted diameter in $\tilde{O}(n^{1/3}/\epsilon)$ rounds w.h.p. Previous work [KS20] showed that computing the exact diameter and $2-\epsilon$ approximate diameter requires $\Omega(n^{1/3})$ rounds. Hence, there is an intriguing question on the round complexity of those problems.

Question 5.2.1. What is the complexity of computing the exact unweighted diameter and 2-approximate weighted diameter in the Congest model?

There is a lack of known algorithms which run in $o(n^{1/3})$ rounds. There is an n^ϵ algorithm for SSSP approximation [AHK+20], however its approximation factor

$(1/\epsilon)^{O(1/\epsilon)}$ is rather large. The $o(n^{1/3})$ barrier is rather natural. It follows from the fact that it is possible to simulate one round of the Congested Clique model on the skeleton graph with nodes sampled independently with probability n^{x-1} in $\tilde{O}(n^{2x-1} + n^{x/2})$ rounds w.h.p. This round complexity has its minima $\tilde{O}(n^{1/3})$ for $x = 2/3$. There also is an unpublished result which computes the approximate SSSP in $o(n^{1/3})$ rounds. This makes us wonder, if there are algorithms for *exact* distances which run faster than $o(n^{1/3})$, or what are the lower bound technique which allows us to obtain better lower bounds for exact distances.

Question 5.2.2. Can we compute the exact distance from some given set of nodes faster than $o(n^{1/3})$ rounds?

Finally, our algorithms might require up to $\Omega(n)$ messages to travel via some edge in one round. We wonder is it possible to obtain similar results with a lower number of messages. Currently, the only work [FHS20] which considers the finite value of local bandwidth λ, more precisely $\lambda = O(\log n)$ shows algorithm on very specific graph families which are sparse.

Question 5.2.3. What is the dependency of round complexity of distance computation on local bandwidth λ of the Hybrid network model?

Current research in the Hybrid network model mostly studies distance related problems. For this kind of problems, the combination of the Local and the Node-Capacitated Clique [AGG$^+$19] models allows to obtain algorithms faster than in each one of the models [KS20]. Since Hybrid at least as powerful as the Node-Capacitated Clique model, the results from [AGG$^+$19] hold in Hybrid model. Some of the limitations of those results can be removed using local edges. For example, [AGG$^+$19, Theorem 3.2] an MST algorithm, where for each edge *at least one* of its endpoints knows that it belongs to MST. Using one round of the Local models, another endpoint can be informed. We wonder, if using global edges can improve problems which are known to be NP-complete in sequential settings. For example, if we can compute a coloring with less than $o(n^{1/2+\epsilon} \cdot \chi)$ colors efficiently [Bar12].

Question 5.2.4. What is the round complexity of NP-hard problems in the Hybrid model.

Bibliography

[ACD$^+$20] Bertie Ancona, Keren Censor-Hillel, Mina Dalirrooyfard, Yuval Efron, and Virginia Vassilevska Williams. Distributed distance approximation. *CoRR*, abs/2011.05066, 2020.

[ACK16] Amir Abboud, Keren Censor-Hillel, and Seri Khoury. Near-linear lower bounds for distributed distance computations, even in sparse networks. In Cyril Gavoille and David Ilcinkas, editors, *Distributed Computing - 30th International Symposium, DISC 2016, Paris, France, September 27-29, 2016. Proceedings*, volume 9888 of *Lecture Notes in Computer Science*, pages 29–42. Springer, 2016.

[AGG$^+$19] John Augustine, Mohsen Ghaffari, Robert Gmyr, Kristian Hinnenthal, Christian Scheideler, Fabian Kuhn, and Jason Li. Distributed computation in node-capacitated networks. In Christian Scheideler and Petra Berenbrink, editors, *The 31st ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2019, Phoenix, AZ, USA, June 22-24, 2019*, pages 69–79. ACM, 2019.

[AHK$^+$20] John Augustine, Kristian Hinnenthal, Fabian Kuhn, Christian Scheideler, and Philipp Schneider. Shortest paths in a hybrid network model. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1280–1299. SIAM, 2020.

[ALSY90] Yehuda Afek, Gad M. Landau, Baruch Schieber, and Moti Yung. The power of multimedia: Combining point-to-point and multiaccess networks. *Inf. Comput.*, 84(1):97–118, 1990.

[AR19] Udit Agarwal and Vijaya Ramachandran. Distributed weighted all pairs shortest paths through pipelining. In *2019 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2019, Rio de Janeiro, Brazil, May 20-24, 2019*, pages 23–32. IEEE, 2019.

[AR20] Udit Agarwal and Vijaya Ramachandran. Faster deterministic all pairs shortest paths in congest model. In Christian Scheideler and Michael Spear, editors, *SPAA '20: 32nd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, July 15-17, 2020*, pages 11–21. ACM, 2020.

[ARKP18] Udit Agarwal, Vijaya Ramachandran, Valerie King, and Matteo Pontecorvi. A deterministic distributed algorithm for exact weighted all-pairs shortest paths in õ(n 3/2) rounds. In Calvin Newport and Idit Keidar, editors, *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, PODC 2018, Egham, United Kingdom, July 23-27, 2018*, pages 199–205. ACM, 2018.

[Bar12] Leonid Barenboim. On the locality of some np-complete problems. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II*, volume 7392 of *Lecture Notes in Computer Science*, pages 403–415. Springer, 2012.

[BKKL17] Ruben Becker, Andreas Karrenbauer, Sebastian Krinninger, and Christoph Lenzen. Near-optimal approximate shortest paths and transshipment in distributed and streaming models. In Andréa W. Richa, editor, *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, volume 91 of *LIPIcs*, pages 7:1–7:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[BN19] Aaron Bernstein and Danupon Nanongkai. Distributed exact weighted all-pairs shortest paths in near-linear time. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 334–342. ACM, 2019.

[CDKL19] Keren Censor-Hillel, Michal Dory, Janne H. Korhonen, and Dean Leitersdorf. Fast approximate shortest paths in the congested clique. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 74–83. ACM, 2019.

[CDP20] Artur Czumaj, Peter Davies, and Merav Parter. Simple, deterministic, constant-round coloring in the congested clique. In Yuval Emek and Christian Cachin, editors, *PODC '20: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, August 3-7, 2020*, pages 309–318. ACM, 2020.

[CFG$^+$19] Yi-Jun Chang, Manuela Fischer, Mohsen Ghaffari, Jara Uitto, and Yufan Zheng. The complexity of $(\Delta+1)$ coloring in congested clique, massively parallel computation, and centralized local computation. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 471–480. ACM, 2019.

[CGL20] Keren Censor-Hillel, François Le Gall, and Dean Leitersdorf. On distributed listing of cliques. In Yuval Emek and Christian Cachin, editors, *PODC '20: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, August 3-7, 2020*, 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.

[CLT20] Keren Censor-Hillel, Dean Leitersdorf, and Elia Turner. Sparse matrix multiplication and triangle listing in the congested clique model. *Theor. Comput. Sci.*, 809:45–60, 2020.

[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 Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 821–840. SIAM, 2019.

[CWC11] Yong Cui, Hongyi Wang, and Xiuzhen Cheng. Channel allocation in wireless data center networks. In *INFOCOM 2011. 30th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 10-15 April 2011, Shanghai, China*, pages 1395–1403. IEEE, 2011.

[DLP12] Danny Dolev, Christoph Lenzen, and Shir Peled. "tri, tri again": Finding triangles and small subgraphs in a distributed setting - (extended abstract). In Marcos K. Aguilera, editor, *Distributed Computing - 26th International Symposium, DISC 2012, Salvador, Brazil, October 16-18, 2012. Proceedings*, volume 7611 of *Lecture Notes in Computer Science*, pages 195–209. Springer, 2012.

[DP20] Michal Dory and Merav Parter. Exponentially faster shortest paths in the congested clique. In Yuval Emek and Christian Cachin, editors, *PODC '20: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, August 3-7, 2020*, pages 59–68. ACM, 2020.

[Elk20] Michael Elkin. Distributed exact shortest paths in sublinear time. *J. ACM*, 67(3):15:1–15:36, 2020.

[FHS20] Michael Feldmann, Kristian Hinnenthal, and Christian Scheideler. Fast hybrid network algorithms for shortest paths in sparse graphs. In *Proceedings of the 24th International Conference on Principles of Distributed Systems (OPODIS)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[FHW12] Silvio Frischknecht, Stephan Holzer, and Roger Wattenhofer. Networks cannot compute their diameter in sublinear time. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1150–1162. SIAM, 2012.

[FN18] Sebastian Forster and Danupon Nanongkai. A faster distributed single-source shortest paths algorithm. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 686–697. IEEE Computer Society, 2018.

[Gal16] François Le Gall. Further algebraic algorithms in the congested clique model and applications to graph-theoretic problems. In Cyril Gavoille and David Ilcinkas, editors, *Distributed Computing - 30th International Symposium, DISC 2016, Paris, France, September 27-29, 2016. Proceedings*, volume 9888 of *Lecture Notes in Computer Science*, pages 57–70. Springer, 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 Calvin Newport and Idit Keidar, editors, *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, PODC 2018, Egham, United Kingdom, July 23-27, 2018*, pages 129–138. ACM, 2018.

[Gha15] Mohsen Ghaffari. Near-optimal scheduling of distributed algorithms. In Chryssis Georgiou and Paul G. Spirakis, editors, *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 3–12. ACM, 2015.

[Gha16] Mohsen Ghaffari. An improved distributed algorithm for maximal independent set. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 270–277. SIAM, 2016.

[Gha17] Mohsen Ghaffari. Distributed MIS via all-to-all communication. In Elad Michael Schiller and Alexander A. Schwarzmann, editors, *Proceedings* *of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, pages 141–149. ACM, 2017.

[GHSS17] Robert Gmyr, Kristian Hinnenthal, Christian Scheideler, and Christian Sohler. Distributed monitoring of network properties: The power of hybrid networks. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 137:1–137:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[GL18] Mohsen Ghaffari and Jason Li. Improved distributed algorithms for exact shortest paths. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 431–444. ACM, 2018.

[GN18] Mohsen Ghaffari and Krzysztof Nowicki. Congested clique algorithms for the minimum cut problem. In Calvin Newport and Idit Keidar, editors, *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, PODC 2018, Egham, United Kingdom, July 23-27, 2018*, pages 357–366. ACM, 2018.

[GN20] Mohsen Ghaffari and Krzysztof Nowicki. Massively parallel algorithms for minimum cut. In Yuval Emek and Christian Cachin, editors, *PODC '20: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, August 3-7, 2020*, pages 119–128. ACM, 2020.

[GNT20] Mohsen Ghaffari, Krzysztof Nowicki, and Mikkel Thorup. Faster algorithms for edge connectivity via random 2-out contractions. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1260–1279. SIAM, 2020.

[GP16] Mohsen Ghaffari and Merav Parter. MST in log-star rounds of congested clique. In George Giakkoupis, editor, *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25-28, 2016*, pages 19–28. ACM, 2016.

[HHLX15] Kai Han, Zhiming Hu, Jun Luo, and Liu Xiang. RUSH: routing and scheduling for hybrid data center networks. In *2015 IEEE Conference on Computer Communications, INFOCOM 2015, Kowloon, Hong Kong, April 26 - May 1, 2015*, pages 415–423. IEEE, 2015.

[Hir76] Daniel S. Hirschberg. Parallel algorithms for the transitive closure and the connected component problems. In Ashok K. Chandra, Detlef Wotschke, Emily P. Friedman, and Michael A. Harrison, editors, *Proceedings of the 8th Annual ACM Symposium on Theory of Computing, May 3-5, 1976, Hershey, Pennsylvania, USA*, pages 55–57. ACM, 1976.

[HKN16] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. A deterministic almost-tight distributed algorithm for approximating single-source shortest paths. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 489–498. ACM, 2016.

[HLL$^+$13] He Huang, Xiangke Liao, Shanshan Li, Shaoliang Peng, Xiaodong Liu, and Bin Lin. The architecture and traffic management of wireless collaborated hybrid data center network. In Dah Ming Chiu, Jia Wang, Paul Barford, and Srinivasan Seshan, editors, *ACM SIGCOMM 2013 Conference, SIGCOMM'13, Hong Kong, China, August 12-16, 2013*, pages 511–512. ACM, 2013.

[Hoe63] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.

[HP15a] James W. Hegeman and Sriram V. Pemmaraju. Lessons from the congested clique applied to mapreduce. *Theor. Comput. Sci.*, 608:268–281, 2015.

[HP15b] Stephan Holzer and Nathan Pinsker. Approximation of distances and shortest paths in the broadcast congest clique. In Emmanuelle Anceaume, Christian Cachin, and Maria Gradinariu Potop-Butucaru, editors, *19th International Conference on Principles of Distributed Systems, OPODIS 2015, December 14-17, 2015, Rennes, France*, volume 46 of *LIPIcs*, pages 6:1–6:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015.

[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 Chryssis Georgiou and Paul G. Spirakis, editors, *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 91–100. ACM, 2015.

[IG17] Taisuke Izumi and François Le Gall. Triangle finding and listing in CONGEST networks. In Elad Michael Schiller and Alexander A. Schwarzmann, editors, *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, pages 381–389. ACM, 2017.

[JN18] Tomasz Jurdzinski and Krzysztof Nowicki. MST in $O(1)$ rounds of congested clique. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 2620–2632. SIAM, 2018.

[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 Yuval Emek and Christian Cachin, editors, *PODC '20: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, August 3-7, 2020*, pages 109–118. ACM, 2020.

[KSV10] Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 938–948. SIAM, 2010.

[Len13] Christoph Lenzen. Optimal deterministic routing and sorting on the congested clique. In Panagiota Fatourou and Gadi Taubenfeld, editors, *ACM Symposium on Principles of Distributed Computing, PODC '13, Montreal, QC, Canada, July 22-24, 2013*, pages 42–50. ACM, 2013.

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

[LP13] Christoph Lenzen and David Peleg. Efficient distributed source detection with limited bandwidth. In Panagiota Fatourou and Gadi Taubenfeld, editors, *ACM Symposium on Principles of Distributed Computing, PODC '13, Montreal, QC, Canada, July 22-24, 2013*, pages 375–382. ACM, 2013.

[LP15] Christoph Lenzen and Boaz Patt-Shamir. Fast partial distance estimation and applications. In Chryssis Georgiou and Paul G. Spirakis, editors, *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 153–162. ACM, 2015.

[LPP19] Christoph Lenzen, Boaz Patt-Shamir, and David Peleg. Distributed distance computation and routing with small messages. *Distributed Comput.*, 32(2):133–157, 2019.

[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.

[Nan14] Danupon Nanongkai. Distributed approximation algorithms for weighted shortest paths. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 565–573. ACM, 2014.

[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 Christian Scheideler and Jeremy T. Fineman, editors, *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures, SPAA 2018, Vienna, Austria, July 16-18, 2018*, pages 405–414. ACM, 2018.

[PRT12] David Peleg, Liam Roditty, and Elad Tal. Distributed algorithms for network diameter and girth. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II*, volume 7392 of *Lecture Notes in Computer Science*, pages 660–672. Springer, 2012.

[RZ11] Liam Roditty and Uri Zwick. On dynamic shortest paths problems. *Algorithmica*, 61(2):389–401, 2011.

[SHK$^+$12] Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed verification and hardness of distributed approximation. *SIAM J. Comput.*, 41(5):1235–1265, 2012.

[UY91] Jeffrey D. Ullman and Mihalis Yannakakis. High-probability parallel transitive-closure algorithms. *SIAM J. Comput.*, 20(1):100–125, 1991.

[VVB14] Stefano Vissicchio, Laurent Vanbever, and Olivier Bonaventure. Opportunities and research challenges of hybrid software defined networks. *Comput. Commun. Rev.*, 44(2):70–75, 2014.

[WAK$^+$10] Guohui Wang, David G. Andersen, Michael Kaminsky, Konstantina Papagiannaki, T. S. Eugene Ng, Michael Kozuch, and Michael P. Ryan. c-through: part-time optics in data centers. In Shivkumar Kalyanaraman, Venkata N. Padmanabhan, K. K. Ramakrishnan, Rajeev Shorey, and Geoffrey M. Voelker, editors, *Proceedings of the ACM SIGCOMM 2010 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, New Delhi, India, August 30 -September 3, 2010*, pages 327–338. ACM, 2010.

לא מאפשר פיתרון יעיל לבעיות הללו. לעומת זאת שילוב שלהם נותן אפשרות לפתור בעיות חישוב מרחקים בזמן תת-ליניארי.

אנחנו משפרים את התוצאות הטובות ביותר במודל הזה בעזרת שימוש בפרוטוקולי תקשורת מודעי צפיפות. ההבחנה החשובה שלנו היא שאזורים בגרף בעלי יותר צמתים לומדים יותר מידע באמצעות הקשתות הגלובליות.

אנו מציגים את הפרימיטיבים התקשורתיים שאנחנו מפתחים בסימלוץ של מודלים חזקים על Skeleton Graph של הגרף המקורי, הגרף בו כל צומת נדגם בהסתברות $n^{-1/3}$ וצמתים נדגמים מחוברים בקשת אם ורק אם הם במרחק של $\tilde{O}(n^{1/3})$ קפיצות בגרף המקורי ($\tilde{O}(), \tilde{\Omega}()$ מסתירים פקטורים פולי-לוגריתמיים). הגרף הזה בהסתברות גבוהה הוא בעל תכונות חשובות בחישוב מרחקים וגם ניתן לסמלץ עליו סיבוב של Congested Clique ב-$\tilde{O}(n^{1/3})$ סיבובים של המודל ההיברידי.

אנחנו מראים איך לסמלץ ב-$\tilde{O}(n^{1/3})$ סיבובים של Hybrid בהסתברות גבוהה סיבוב אחד של מודל Oracle על Skeleton Graph. במודל הזה קיים צומת (בעל דרגה מקסימלית) שבכל סיבוב יכול לקבל ולשלוח $\deg_S(v)$ הודעות לכל צומת v. במודל הזה אנחנו משתמשים בשביל למצוא מרחק מצומת נתון לכל הצמתים בגרף.

כמו כן אנחנו מראים איך לסמלץ ב-$\tilde{O}(n^{1/3})$ סיבובים של Hybrid בהסתברות גבוהה סיבוב אחד של מודל Tiered Oracles על Skeleton Graph. במודל הזה בסיבוב אחד כל צומת u לומד $\deg_S(v)$ הודעות מכל צומת v כך ש-$\deg_S(v) < \deg_S(u)/2$. זה מאפשר לחלק את הצמתים ל-$O(\log n)$ שכבות לפי הדרגה שלהם מעוגלת לחזקה של 2 וללמוד ע"י צמתים משכבות עליונות על קשתות צמודות לצמתים משכבות תחתונות. אנחנו פותרים את $n^{2/3}$-RSSP, שהיא בעיית מציאת משקל מסלול קל ביותר מקבוצה בה כל צומת נדגם בהסתברות $n^{-1/3}$ ב-$\tilde{O}(n^{1/3})$ סיבובים בהסתברות גבוהה. אנחנו גם מראים חסם תחתון שמראה שהסיבוכיות הזאת היא אופטימלית עד כדי פקטור פולי-לוגריתמי. בעזרת פתרון לבעיה הזאת ושימוש בכלים נוספים אנחנו מראים אלגוריתם שמוצא בהסתברות גבוהה משקל מסלולים קלים ביותר מכל הצמתים בגרף לכל אחד מ- $n^{1/3}$ הצמתים הנתונים.

לבנוסף, אנחנו גם משתמשים בשיטה ידועה בשביל לחשב קירוב ל- n^x-SSP, כלומר, למשקל מסלולים קלים ביותר מקבוצה נתונה של n^x צמתים, בעזרת האלגוריתם שלנו ל-$n^{2/3}$-RSSP, ב-$\tilde{O}(n^{1/3} + n^{x/2})$ סיבובים בהסתברות גבוהה. לפי חסם תחתון מעבודות קודמות, התוצאה הזאת היא אופטימלית מבחינת זמן ריצה עד כדי פקטור פולי-לוגריתמי. היא גם משפרת תוצאות מעבודות קודמות בזמן ריצה או פקטור קירוב.

אנחנו משתמשים בתוצאות חישוב מרחקים ממספר צמתים בשביל לקרב אקסצנטריות של הצמתים וקוטר של הגרף. אקסצנטריות של צומת הוא מרחק הכבד ביותר מאותו צומת. קוטר של הגרף הוא האקסצנטריות מקסימלית בגרף, כלומר מרחק הכי כבד בגרף. אנחנו מראים קירוב $1+\epsilon$ לקוטר בגרף לא ממושקל ו-2 קירוב לקוטר בגרף ממושקל ב-$\tilde{O}(n^{1/3}/\epsilon)$ בהסתברות גבוהה. התוצאה הזאת משפרת תוצאות קודמות. היא גם מעניינת לאור חסם תחתון $\tilde{\Omega}(n^{1/3})$ ידוע עבור קוטר מדויק ועבור $2-\epsilon$ קירוב לקוטר בגרף ממושקל.

במכונות שונות, האלגוריתם עושה את ההקצאה מחדש בכל סיבוב. בשביל לחשב את ההקצאה וליישם אותה בצורה יעילה - אנחנו מגדירים את המחלקה של משימות שהן יעילות בזיכרון ומפתחים אלגוריתם שפועל עבורן.

האלגוריתם הבא שפיתחנו הוא אלגוריתם אקראי שנועד להסיר את הנחת יעילות בזיכרון. הוא פועל עבור משימות שהן יעילות בקלט-פלט. רוב הבעיות הנחקרות ב-Congested Clique הן כאלו. אנחנו מראים בעזרת חסמי הופדינג שאחרי ערבוב אקראי של הקלט, Lenzen's routing scheme (תכנית ניתוב ידועה) מעבירה את ההודעות של אותו השלב במספר קרוב לאופטימלי של סיבובים בהסתברות גבוהה.

האלגוריתם האחרון שאנחנו מציגים הוא אלגוריתם אקראי המבוסס על שיטת העיכובים האקראיים שפותחה בשביל בעיות ניתוב חבילות, ושומשה גם בשביל שיבוץ משימות במודל החישוב המבוזר Congest.

אנחנו משווים את האלגוריתמים הללו בינם לבין עצמם ומדגימים אותם על בעיית מציאת קבוצה בלתי תלויה מקסימלית, כלומר בעיה בה צריכים לחשב תת קבוצה של צמתי הגרף כך שאין קשת ביניהם ואי אפשר להרחיב אותה על ידי הוספת עוד צומת. הפרוטוקול הטוב ביותר הידוע למציאת קבוצה בלתי תלויה מקסימלית ב-Congested Clique רץ ב-$O(\log\log\Delta)$ סיבובים בהסתברות גבוהה ואנחנו מראים שהוא צורך $O(n^2)$ הודעות. בעזרת האלגוריתם מבוסס הערבוב שלנו, אנחנו משבצים t משימות של מציאת קבוצה בלתי תלויה מקסימלית ב-$O(t+\log\log\Delta\log n)$ סיבובים בהסתברות גבוהה. הסיבוכיות המשוערת עבור ביצוע מספר משימות מוגדרת להיות מספר הסיבובים הכולל הנדרש לביצוע כל המשימות, מחולק במספר המשימות. כלומר אנחנו מראים איך להריץ $t=\Omega(\log\log\Delta\log n)$ משימות כאלו במספר קבוע של סיבובים משוערכים.

בעיה נוספת שאנחנו מדגימים את האלגוריתמים שלנו עליה היא Pointer Jumping, בה לכל צומת יש תמורה, ולאחד מהצמתים יש מספר והוא נדרש לפלוט את התוצאה של הפעלת הרכבת התמורות על המספר הזה. אנחנו מראים פרוטוקול שהוא יעיל גם בזיכרון וגם בקלט/פלט עבור תמורות שאורכן לכל היותר כמספר מכונות. האלגוריתם מבוסס הערבוב מבצע את t המשימות הללו ב-$O(t+\log^2 n)$ סיבובים בהסתברות גבוהה. האלגוריתם הדטרמיניסטי מבצע $\sqrt{n}$ משימות של Pointer Jumping עם תמורות באורך $\sqrt{n}$ ב-$O(\log n)$ סיבובים. במילים אחרות אנחנו מראים אלגוריתם לביצוע המשימות הללו ב- $o(1)$ סיבובים משוערכים.

לסיכום, אנו מציעים סיבוכיות משוערכת כמדד מעניין למחקר עתידי ב-Congested Clique ומשאירים כשאלה פתוחה למחקר עתידי את הסיבוכיות המשוערכת של בעיות עבורן ידועים אלגוריתמים הרצים במספר קבוע של סיבובים.

מודל ה- Hybrid הוא מודל תאורטי המתאר תקשורות ברשתות היברידיות. הוא מושפע בפרט ממרכזי נתונים עם תיקשורת היברידית. המודל הזה הוא שילוב של מודל חישוב מבוזר ידוע Local, בו בכל סיבוב ניתן לשלוח מספר לא מוגבל של הודעות בין כל זוג צמתים שהם שכנים בגרף הקלט בעזרת קשתות לוקליות ומודל חישוב מבוזר שהוגדר לאחרונה, Node-Capacitated Clique, בו ניתן בכל סיבוב לכל צומת לשלוח ולקבל $O(\log n)$ הודעות בעזרת קשתות גלובליות.

במודל הזה אנחנו חוקרים בעיות חישוב מרחקים. שימוש בקשתות גלובליות או לוקאליות בלבד

תקציר

בעידן המודרני בו כמות הנתונים ודרישות העיבוד הולכות וגדלות, אנשים משתמשים בחישוב מבוזר על מנת לעבד את הנתונים הללו. בפרט, פעמים רבות אנשים משתמשים ברשתות אשר מאפשרות תקשורת קצה לקצה כגון overlay network או ברשתות היברידיות אשר מאפשרות גם תקשורת מהירה בין חלק מזוגות המכונות וגם תקשורת איטית יותר בין כל הזוגות.

המחקר שלנו עוסק בהשפעה של הקשתות הגלובליות על חישוב מבוזר. אנו חוקרים מודלים שעובדים בסיבובים סינכרוניים. מדד הסיבוכיות העיקרי בהם הוא מספר הסיבובים עד שהמכונה האחרונה מסיימת את חישובה. מדד מעניין נוסף הוא מספר ההודעות שנשלחות על ידי האלגוריתם. אנחנו חוקרים תקשורת קצה לקצה במודל ידוע שנקרא Congested Clique ומודל נוסף שהוצג לאחרונה שנקרא Hybrid.

Congested Clique הוא מודל תאורטי לתיאור תקשורת ב-overlay network. במודל הזה יש n מכונות, כאשר בכל סיבוב כל מכונה יכולה לשלוח ולקבל הודעה אחת. להרבה מהבעיות במודל הזה ידועים פתרונות ב-$O(1)$ סיבובים. במודל הזה אנחנו חוקרים את הבעיה המשלימה. אנחנו מעוניינים בהרצה של הרבה משימות מבוזרות במקביל. אנחנו מציגים 3 אלגוריתמים מבוזרים שפותרים בעיית שיבוץ משימות מבוזרות בזמן קרוב לאופטימלי.

האלגוריתמים שלנו לא מניחים ידע מקדים על דפוס התקשורת, ורובם גם לא מניחים ידע מקדים על תכונות לא טריוויאליות שלו. מטעמי נוחות, אנחנו קוראים צומת לכל אחד מצמתים ווירטואלים בכל אחת ממשימות ומכוגה לכל אחת ממכונות פיזיות שמבצעות את אלגוריתם השיבוץ. עבור קבוצה של משימות, כל מכונה מקבלת את הקלט באופן מבוזר. כלומר כל מכונה מקבלת קלט של אחד מהצמתים בכל אחת מהמשימות. המטרה של כל מכונה לפלוט פלט גם כן בצורה מבוזרת, כלומר כל מכונה צריכה לפלוט עבור כל משימה את הפלט של הצומת שהיא קיבלה את הקלט שלו. אנחנו מניחים שהפרוטוקולים ידועים לכל המכונות.

כל האלגוריתמים שלנו מריצים את המשימות בשלבים. בכל שלב אלגוריתם השיבוץ מקדם את כל המשימות לסיבוב הבא. לכן, תפקידו של אלגוריתם השיבוץ הוא להעביר את ההודעות הנשלחות על-ידי המשימות בצורה כמה שיותר יעילה. האתגר העיקרי של האלגוריתם הוא המקרה בו הרבה מההודעות נשלחות על ידי משימות שונות מאותה מכונה ונוצר צוואר בקבוק בתקשורת.

האלגוריתם הראשון שאנחנו מציגים הוא אלגוריתם דטרמיניסטי. הדרך שהוא מתמודד עם האתגר המתואר, הוא להקצות מחדש את הצמתים הווירטואליים המסומלצים למכונות פיזיות אשר מסמלצות אותם בצורה שפותרת את צוואר בקבוק התקשורת. משום שהצוואר הבקבוק בכל שלב עלול להיות

המחקר בוצע בהנחייתה של פרופסור קרן צנזור-הלל, בפקולטה למדעי המחשב.

חלק מן התוצאות בחיבור זה פורסמו כמאמרים מאת המחבר ושותפיו למחקר בכנסים ובכתבי-עת במהלך תקופת מחקר מגיסטר של המחבר, אשר גרסאותיהם העדכניות ביותר הינן:

Keren Censor-Hillel, Dean Leitersdorf, and Volodymyr Polosukhin. Distance computations in the hybrid network model via oracle simulations. *CoRR (To appear in STACS 2021)*, abs/2010.13831, 2020.

Keren Censor-Hillel, Yannic Maus, and Volodymyr Polosukhin. Near-optimal scheduling in the congested clique. *CoRR (To appear in SIROCCO 2021)*, abs/2102.07221, 2021.

תודות

אני מודה למנחה שלי פרופסור קרן צנזור-הלל על הנחייתה בסבלנות ועם תשומת לב. אני רוצה להודות לשותפים שלי דין לייטרסדורף ויאניק מאוס על תרומתם וזמנם. כמו כן אני רוצה להודות להוריי, משפחתי וחברי על תמיכתם האישית והעזרה התחבירית מקרוב ומרחוק.

חישוב מבוזר בעזרת קשתות גלובליות בעלות רוחב פס מוגבל

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת התואר
מגיסטר למדעים במדעי המחשב

וולודימיר פולוסוחין

הוגש לסנט הטכניון — מכון טכנולוגי לישראל
טבת תשפ"א   חיפה   ינואר 2020

חישוב מבוזר בעזרת קשתות גלובליות בעלות רוחב פס מוגבל

וולודימיר פולוסוחין