On the Distributed Complexity of Large-Scale Graph Computations

Gopal Pandurangan
University of Houston
Houston, TX, USA
gopalpandurangan@gmail.com

Peter Robinson
McMaster University
Hamilton, Canada
peter.robinson@mcmaster.ca

Michele Scquizzato
KTH Royal Institute of Technology
Stockholm, Sweden
mscq@kth.se

ABSTRACT

Motivated by the increasing need to understand the distributed algorithmic foundations of large-scale graph computations, we study some fundamental graph problems in a message-passing model for distributed computing where $k \geqslant 2$ machines jointly perform computations on graphs with n nodes (typically, $n \gg k$). The input graph is assumed to be initially randomly partitioned among the k machines, a common implementation in many real-world systems. Communication is point-to-point, and the goal is to minimize the number of communication *rounds* of the computation.

Our main contribution is the *General Lower Bound Theorem*, a theorem that can be used to show non-trivial lower bounds on the round complexity of distributed large-scale data computations. The General Lower Bound Theorem is established via an information-theoretic approach that relates the round complexity to the minimal amount of information required by machines to solve the problem. Our approach is generic and this theorem can be used in a "cook-book" fashion to show distributed lower bounds in the context of several problems, including non-graph problems. We present two applications by showing (almost) tight lower bounds for the round complexity of two fundamental graph problems, namely *PageRank computation* and *triangle enumeration*. Our approach, as demonstrated in the case of PageRank, can yield tight lower bounds for problems (including, and especially, under a stochastic partition of the input) where communication complexity techniques are not obvious. Our approach, as demonstrated in the case of triangle enumeration, can yield stronger round lower bounds as well as message-round tradeoffs compared to approaches that use communication complexity techniques.

We then present distributed algorithms for PageRank and triangle enumeration with a round complexity that (almost) matches the respective lower bounds; these algorithms exhibit a round complexity which scales superlinearly in k, improving significantly over previous results for these problems [Klauck et al., SODA 2015]. Specifically, we show the following results:

- *PageRank:* We show a lower bound of $\tilde{\Omega}(n/k^2)$ rounds, and present a distributed algorithm that computes the PageRank of all the nodes of a graph in $\tilde{O}(n/k^2)$ rounds.
- *Triangle enumeration:* We show that there exist graphs with m edges where any distributed algorithm requires $\tilde{\Omega}(m/k^{5/3})$ rounds. This result also implies the first non-trivial lower bound of $\tilde{\Omega}(n^{1/3})$ rounds for the *congested clique* model, which is tight up to logarithmic factors. We then present a distributed algorithm that enumerates all the triangles of a graph in $\tilde{O}(m/k^{5/3} + n/k^{4/3})$ rounds.

SPAA '18, July 16–18, 2018, Vienna, Austria

ACM ISBN 978-1-4503-5799-9/18/07…$15.00
https://doi.org/10.1145/3210377.3210409

CCS CONCEPTS

• **Theory of computation → Massively parallel algorithms**; **Distributed algorithms**;

ACM Reference Format:
Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. 2018. On the Distributed Complexity of Large-Scale Graph Computations. In *Proceedings of SPAA '18: 30th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '18).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3210377.3210409

1 INTRODUCTION

The focus of this paper is on the distributed processing of large-scale data, in particular, *graph data*, which is becoming increasingly important with the rise of massive graphs such as the Web graph, social networks, biological networks, and other graph-structured data and the consequent need for fast distributed algorithms to process such graphs. Several large-scale graph processing systems such as Pregel [23] and Giraph [1] have been recently designed based on the *message-passing* distributed computing model [22, 30]. In these systems, the input graph, which is simply too large to fit into a single machine, is distributed across a group of machines that are connected via a communication network and the machines jointly perform computation in a distributed fashion by sending/receiving messages. A key goal in distributed large-scale computation is to minimize the amount of communication across machines, as this typically dominates the overall cost of the computation.

We study fundamental graph problems in a message-passing distributed computing model and present almost tight bounds on the number of communication rounds needed to solve these problems. In the model, called the k-machine model [19] (explained in detail in Section 1.1), the input graph (or more generally, any other type of data) is distributed across a group of k machines that are pairwise interconnected via a communication network. The k machines jointly perform computations on an arbitrary n-vertex input graph (where typically $n \gg k$) distributed among the machines. The communication is point-to-point via message passing. The goal is to minimize the *round complexity*, i.e., the number of *communication rounds*, given some (bandwidth) constraint on the amount of data that each link of the network can deliver in one round. We address

a fundamental issue in distributed computing of large-scale data: What is the distributed (round) complexity of solving problems when each machine can see only *a portion of the input* and there is a *limited bandwidth* for communication? We would like to quantify the round complexity of solving problems as a function of the *size of the input* and the *number of machines* used in the computation.

A main contribution of this paper is a technique that can be used to show non-trivial lower bounds on the distributed complexity (number of communication rounds) of large-scale data computations, and its application to graph problems.

1.1 The Model

We now describe the adopted model of distributed computation, the *k-machine model* (a.k.a. the *Big Data model*), introduced in [19] and further investigated in [3, 7, 27, 29, 32]. The model consists of a set of $k \geqslant 2$ machines $\{M_1, M_2, \ldots, M_k\}$ pairwise interconnected by bidirectional point-to-point communication links. Each machine executes an instance of a distributed algorithm. The computation advances in synchronous rounds where, in each round, machines can exchange messages over their communication links and perform some local computation. Each link is assumed to have a bandwidth of B bits per round; unless otherwise stated, we assume $B = \Theta(\text{polylog}\, n)$. Machines do not share any memory and have no other means of communication. We assume that each machine has access to a private source of true random bits. We say that algorithm $\mathcal{A}$ has ϵ-*error* if, in any run of $\mathcal{A}$, the output of the machines corresponds to a correct solution with probability at least $1 - \epsilon$. The *round complexity of* $\mathcal{A}$ is defined to be the worst-case number of rounds required by any machine when executing $\mathcal{A}$.

Local computation within a machine is considered to happen instantaneously at zero cost, while the exchange of messages between machines is the costly operation. However, we note that in all the algorithms of this paper, every machine in every round performs lightweight computations; in particular, these computations are bounded by a polynomial (typically, even linear) in the size of the input assigned to that machine.

Although the k-machine model is a general model of distributed computation that can be applied to study any (large-scale data) problem, in this paper we focus on investigating graph problems in it. Specifically, we are given an input graph G with n vertices, each associated with a unique integer ID from $[n]$, and m edges. To avoid trivialities, we will assume that $n \geqslant k$ (typically, $n \gg k$). Initially, the entire graph G is not known by any single machine, but rather partitioned among the k machines in a "balanced" fashion, i.e., the nodes and/or edges of G must be partitioned approximately evenly among the machines. We assume a *vertex-partition* model, whereby vertices (and their incident edges) are partitioned across machines. Specifically, the type of partition that we will assume throughout is the *random vertex partition (RVP)*, i.e., vertices (and their incident edges) of the input graph are assigned randomly to machines. This is the typical way used by many real graph processing systems, such as Pregel [23] and Giraph [1, 6], to partition the input graph among the machines; it is easy to accomplish, e.g., via hashing.

More formally, in the *random vertex partition* model each vertex of G is assigned independently and uniformly at random to one of the k machines. If a vertex v is assigned to machine M_i we say that M_i is the *home machine* of v and, with a slight abuse of notation, write $v \in M_i$. When a vertex is assigned to a machine, all its incident edges are known to that machine as well, i.e., the home machine initially knows the IDs of the neighbors of that vertex as well as the identities of their home machines (and the weights of the corresponding edges in case G is weighted). For directed graphs, we assume that out-edges of vertices are known to the assigned machine. (However, we note that our lower bounds hold even if both in- and out-edges are known to the home machine.) An immediate property of the RVP model is that the number of vertices at each machine is *balanced*, i.e., each machine is the home machine of $\tilde{\Theta}(n/k)$ vertices with high probability (see [19]); we shall assume this throughout the paper.

1.2 Our Results

We present a general information-theoretic approach for showing non-trivial round lower bounds for certain graph problems in the k-machine model. This approach can be useful in the context of showing round lower bounds for many other (including non-graph) problems in a distributed setting where the input is partitioned across several machines and the output size is large. Using our approach we show almost tight (up to logarithmic factors) lower bounds for two fundamental, seemingly unrelated, problems, namely PageRank computation and triangle enumeration. These lower bounds apply to distributed computations in essentially all point-to-point communication models, since they apply even to a synchronous complete network (where $k = n$), and *even* when the input is partitioned *randomly*, and thus they apply to worst-case balanced partitions as well (unlike some previous lower bounds, e.g., [39], which apply only under some worst-case partition).

To demonstrate the near-tightness of our lower bounds we present optimal (up to polylog(n) factors) distributed algorithms for such problems. The round complexity of these algorithms scales *superlinearly* in k, improving significantly over previous results.

1. PageRank Computation. In Section 2.3 we show an almost tight lower bound of $\tilde{\Omega}(n/k^2)$ rounds.[1] In Section 3.1 we present an algorithm that computes the PageRank of all nodes of a graph in $\tilde{O}(n/k^2)$ rounds, thus improving over the previously known bound of $\tilde{O}(n/k)$ rounds [19].

2. Triangle Enumeration. In Section 2.4 we show that there exist graphs with m edges where any distributed algorithm requires $\tilde{\Omega}(m/k^{5/3})$ rounds. In Section 3.2 we present an algorithm that enumerates all the triangles of a graph in $\tilde{O}(m/k^{5/3}+n/k^{4/3})$ rounds. This improves over the previously known bound of $\tilde{O}(n^{7/3}/k^2)$ rounds [19].

Our technique can be used to derive lower bounds in other models of distributed computing as well. Specifically, the approach used to show the lower bound for triangle enumeration can be adapted for the popular congested clique model (discussed in Section 1.4), yielding an $\Omega(n^{1/3}/\log n)$ lower bound for the same problem.[2] (Notice that this does not contradict the impossibility result of [12],

[1] Notation $\tilde{\Omega}$ hides a $1/\text{polylog}(n)$ factor, and $\tilde{O}$ hides a $\text{polylog}(n)$ factor and an additive $\text{polylog}(n)$ term.

[2] A preliminary version of this paper, appeared on arXiv [28], contained a slightly worse lower bound of the form $\Omega(n^{1/3}/\log^3 n)$; later, a subsequent work by Izumi and Le Gall [16] showed a lower bound of the form $\Omega(n^{1/3}/\log n)$ using our information-theoretic approach.

which states that any super-constant lower bound for the congested clique would give new lower bounds in circuit complexity: because of the size required by any solution for triangle enumeration, Remark 3 in [12] does not apply.) To the best of our knowledge, this is the first *super-constant* lower bound known for the congested clique model. (Previous bounds were known for weaker versions of the model, e.g., which allowed only broadcast communication, or which applied only to deterministic algorithms [12], or for implementations of specific algorithms [5].)

Our bounds for triangle enumeration also apply to the problem of enumerating all the *open triads*, that is, all the sets of three vertices with exactly two edges. Our techniques and results can be generalized to the enumeration of other small subgraphs such as cycles and cliques.

Due to lack of space, full proofs and additional details are deferred to the full version of the paper.

1.3 Overview of Techniques

Lower Bounds. In Theorem 2.1 we prove a general result, the *General Lower Bound Theorem*, which relates the round complexity in the k-machine model to the minimal amount of information required by machines for correctly solving a problem. While PageRank and triangle enumeration are fundamentally different problems, we derive lower bounds for both problems via the "information to running time" relationship of Theorem 2.1. The General Lower Bound Theorem gives two probabilistic bounds that must be satisfied in order to obtain a lower bound on the round complexity of any problem. The two bounds together capture the decrease in uncertainty (called *surprisal*, see Section 2) that happens to some machine as a result of outputting the solution. We can show that this "surprisal change" represents the maximum expected "Information Cost" over all machines which can be used to lower bound the run time. The proof of the General Lower Bound Theorem makes use of information-theoretic machinery, yet its application requires no use of information theory.

We conjecture that Theorem 2.1 can be used to obtain lower bounds for various problems (including non-graph problems) that have a relatively *large output size* (e.g., shortest paths, sorting, matrix multiplication, etc.) thus complementing the approach based on communication complexity (see, e.g., [9, 12, 13, 19, 24–27, 31] and references therein). In fact, our approach, as demonstrated in the case of triangle enumeration, can yield stronger round lower bounds as well as message-round tradeoffs compared to approaches that use communication complexity techniques (more on this in the next paragraph). Our approach, as demonstrated in the case of PageRank, can yield tight lower bounds for problems (including, and especially, under a stochastic/random partition of the input) where communication complexity techniques are not obvious. In fact, for many problems, applying the General Lower Bound Theorem gives non-trivial lower bounds in a fairly straightforward way that are not (at least easily) obtainable by communication complexity techniques. To give an example, the work of Klauck et al. [19] showed a lower bound of $\tilde{\Omega}(n/k^2)$ for connectivity by appealing to random partition communication complexity—this involved proving the classical set disjointness lower bound *under random input partition*, which involved non-trivial work. On the other hand, the same lower bound of $\tilde{\Omega}(n/k^2)$ for MST can be shown directly[3] via the General Lower Bound Theorem (this bound is tight due to the algorithm of [27]). To give another example, consider the problem of distributed sorting (see, e.g., [26]), whereby n elements are randomly distributed across the k machines and the requirement is that, at the end, the i-th machine must hold the $(i-1)k+1, (i-1)k+2, \ldots, i \cdot k$-th order statistics. One can use the General Lower Bound Theorem to show a $\tilde{\Omega}(n/k^2)$ lower bound for this problem (and this is tight, as there exists an $\tilde{O}(n/k^2)$-round sorting algorithm). Note that the same lower bound (under a random partition) is harder to show using communication complexity techniques.

We also note that tight *round* complexity lower bounds do not always directly follow from exploiting *message (bit)* complexity lower bounds obtained by leveraging communication complexity results. For example, for the problem of triangle enumeration, even assuming the highest possible message lower bound of $\Omega(m)$, this would directly imply a *round* lower bound of $\tilde{\Omega}(m/k^2)$ (since $\Theta(k^2)$ messages can be exchanged in one round) and not the tight $\tilde{\Omega}(m/k^{5/3})$ shown in this paper. Furthermore, our approach can show round-message *tradeoffs* giving stronger message lower bounds for algorithms constrained to run in a prescribed round bound compared to what one can obtain using communication complexity approaches. In particular, for triangle enumeration, we show that any round-optimal algorithm that enumerates all triangles with high probability in the k-machine model needs to exchange a total of $\tilde{\Omega}(mk^{1/3})$ messages in the worst case.

We emphasize that our General Lower Bound theorem gives non-trivial lower bounds only when the output size is large enough, but it still works seamlessly across all output sizes. To illustrate this, we note that the triangle enumeration lower bound of $\tilde{\Omega}(m/k^{5/3})$ is true only for dense graphs, i.e., $m = \Theta(n^2)$. In fact, the real lower bound derived through our theorem is $\tilde{\Omega}((t/k)^{2/3}/k)$, where t is the number of triangles in the input graph; this bound can be shown to apply even for sparse (random) graphs by extending our analysis.

Entropy-based information-theoretic arguments have been used in prior work [19]. However, there is a crucial difference, as explained next. In [19], it was shown that $\tilde{\Omega}(n/k)$ is a lower bound for computing a spanning tree (ST) of a graph. However, this lower bound holds under the criterion that the machine which hosts the vertex (i.e., its home machine) must know at the end of the computation the status of all of its incident edges (whether they belong to a ST or not) and output their respective status. The lower bound proof exploits this criterion to show that any algorithm will require some machine receiving $\Omega(n)$ bits of information, and since any machine has $k-1$ links, this gives a $\tilde{\Omega}(n/k)$ lower bound. This argument fails if we require the final status of each edge to be known by *some* machine (different machines might know the status of different edges); indeed under this output criterion, it can be shown that MST can be solved in $\tilde{O}(n/k^2)$ rounds [27]. On the other hand, the lower bound proof technique of this paper applies to the less restrictive (and more natural) criterion that any machine can output any part of the solution. In [4], a direct sum theorem is shown that yields a communication complexity lower bound for set disjointness. The method of [4] can be applied to obtain lower bounds for functions

[3] The lower bound graph can be a complete graph with random edge weights.

F that can be "decomposed" as $F(\mathbf{x}, \mathbf{y}) = f(g(x_1, y_1), \ldots, g(x_n, y_n))$, by reduction from the information complexity of the function g. These methods do not seem applicable to our setting as we are considering problems where the output size is large.

Upper Bounds. The Conversion Theorem of [19] directly translates algorithms designed for a message passing model for network algorithms to the k-machine model, and almost all the previous algorithms [7, 19, 32] were derived using this result. In contrast, the present paper does not use the Conversion Theorem; instead, it gives direct solutions for the problems at hand, leading to improved algorithms with significantly better round complexity.

While our algorithms use techniques specific to each problem, we point out a simple, but key, unifying technique that proves very useful in designing fast algorithms, called *randomized proxy computation*.[4] Randomized proxy computation is crucially used to distribute communication *and* computation across machines to avoid congestion at any particular machine, which instead is redistributed evenly across all the machines. This is achieved, roughly speaking, by re-assigning the *executions* of individual nodes uniformly at random among the machines. Proxy computation allows one to move away from the communication pattern imposed by the topology of the input graph, which can cause congestion at a particular machine, to a more balanced communication overall.

1.4 Related Work

For a comparison of the k-machine model with other parallel and distributed models proposed for large-scale data processing, including Bulk Synchronous Parallel (BSP) model [37], MapReduce [18], and the congested clique, we refer to [38]. In particular, according to [38], "Among all models with restricted communication the "big data" [k-machine] model is the one most similar to the MapReduce model".

Klauck et al. [19] present lower and upper bounds for several fundamental graph problems in the k-machine model. In particular, they presented weaker upper bounds for PageRank and triangle verification (which also works for triangle enumeration), which are substantially improved in this paper. They do not present any non-trivial lower bound for any of these problems. Also, as pointed out earlier, some lower bounds shown in [19], most notably the $\Omega(n/k^2)$ lower bound of MST (under random input partition and under the requirement that each MST edge has to be output by *some* machine), can be shown in a simpler way using the General Lower Bound Theorem of this paper. Pandurangan et al. [27] showed $\tilde{O}(n/k^2)$-round algorithms in the k-machine model for connectivity, MST, approximate min-cut, and other graph verification problems. The algorithmic techniques used in that paper (except for the randomized proxy computation) cannot be applied for PageRank and triangle enumeration.

The k-machine model is closely related to the BSP model [37]; it can be considered to be a simplified version of BSP, where local computation is ignored and synchronization happens at the end of every round (the synchronization cost is ignored). Unlike BSP which has a lot of different parameters (which typically makes it harder to prove rigorous theoretical bounds [38]), the k-machine model is characterized by one parameter (the number of machines) which allows one to develop and prove clean bounds and serves as a basis for comparing various distributed algorithms.

The k-machine model is also closely related to the classical CONGEST model [30], and in particular to the *congested clique* model, which recently has received considerable attention (see, e.g., [5, 12, 14, 15, 17, 20, 21]). The main difference is that the k-machine model is aimed at the study of large-scale computations, where the size n of the input is significantly bigger than the number of available machines k, and thus many vertices of the input graph are mapped to the same machine, whereas the two aforementioned models are aimed at the study of distributed network algorithms, where $n = k$ and each vertex corresponds to a dedicated machine. More "local knowledge" is available per vertex (since it can access for free information about other vertices in the same machine) in the k-machine model compared to the other two models. On the other hand, all vertices assigned to a machine have to communicate through the links incident on this machine, which can limit the bandwidth (unlike the other two models where each vertex has a dedicated processor). These differences manifest in the design of fast algorithms for these models. In particular, the best distributed algorithm in the congested clique model may not directly yield the fastest algorithm in the k-machine model [27].

2 LOWER BOUNDS

2.1 A General Lower Bound Theorem

In this section we present a result, called *General Lower Bound Theorem*, which provides a general way to obtain round lower bounds in the k-machine model. In Section 2.2 we provide the full of this result. We will then apply it to derive lower bounds for two graph problems, namely, PageRank computation (Section 2.3) and triangle enumeration (Section 2.4).

Consider an n-vertex input graph G partitioned across the machines via the random-vertex partition in the k-machine model. Note that the input graph G is sampled from a probability distribution on a (suitably chosen) set of graphs $\mathcal{G}$. (For example, in the case of PageRank, $\mathcal{G}$ is the set of all possible instantiations of the lower bound graph H shown in Figure 1.) Consider a partition $\mathbf{p} = (p_1, \ldots, p_k)$ of an input graph G. We use boldface $\mathbf{p}$ to denote a vector and p_i to denote the i-th entry of $\mathbf{p}$, which corresponds to the subgraph assigned to machine M_i. In our analysis, we frequently condition on the event that a subgraph $p_i \subseteq G$ is assigned to a certain machine M_i. To simplify the notation, we also use p_i to denote the event that this happens, e.g., $\Pr[E \mid p_i]$ is the probability of event E conditioned on the assignment of p_i to machine M_i.

Let Π_i be the random variable representing the transcript of the messages received by machine M_i across its $k-1$ links when executing a given algorithm $\mathcal{A}$ for (at most) T rounds, and let $\mathcal{GP}$ be the set of all possible partitions of the graphs in $\mathcal{G}$ among the k machines. The execution of algorithm $\mathcal{A}$ is fully determined by the given input partitioning $\mathbf{p} \in \mathcal{GP}$ and the public random bit string $R \in \mathcal{RS}$, where $\mathcal{RS}$ is the set of all possible strings that are used as random bit string by the algorithm. Note that R is itself a random variable. Similarly to above, we write $\Pr[E \mid p_i, r]$ when conditioning event E on the events that the public random string is r and machine M_i obtains subgraph p_i as its input, where

[4] Similar ideas have been used in parallel and distributed computation in different contexts, see, e.g., [35, 36].

$\mathbf{p} = (p_1, \ldots, p_i, \ldots, p_k)$ and $(\mathbf{p}, r) \in \mathcal{GP} \times \mathcal{RS}$. We use $\mathcal{A}_i(\mathbf{p}, r)$ to denote the output of machine M_i, when executing the algorithm for a given $(\mathbf{p}, r)$. For technical reasons, we assume that the output $\mathcal{A}_i(\mathbf{p}, r)$ also includes M_i's initial graph input p_i and the random string r.

Theorem 2.1 (General Lower Bound Theorem). *Let* $\mathrm{IC} = \mathrm{IC}(n, k)$ *be a positive integer-valued function called* information cost, *and let* Z *be a random variable depending only on the input graph. Consider a* T*-round* ϵ*-error algorithm* $\mathcal{A}$*, for some* $\epsilon = o(\mathrm{IC}/\mathbb{H}[Z])$*, where* $\mathbb{H}[Z]$ *is the entropy of* Z*. Let* $Good \subseteq \mathcal{GP} \times \mathcal{RS}$ *be a set of pairs* $(\mathbf{p}, r)$ *where* $\mathbf{p} = (p_1, \ldots, p_k) \in \mathcal{GP}$ *is an input partition and* $r \in \mathcal{RS}$ *is a public random string, and* $|Good| \geqslant (1 - \epsilon - n^{-\Omega(1)})|\mathcal{GP} \times \mathcal{RS}|$*. Suppose that, for every* $(\mathbf{p}, r) \in Good$*, there exists a machine* M_i *receiving input graph* p_i *and outputting* $\mathcal{A}_i(\mathbf{p}, r)$*, such that*

$$Pr[Z = z \mid p_i, r] \leqslant \left(\tfrac{1}{2}\right)^{\mathbb{H}[Z] - o(\mathrm{IC})}, \tag{1}$$

$$Pr[Z = z \mid \mathcal{A}_i(\mathbf{p}, r), p_i, r] \geqslant \left(\tfrac{1}{2}\right)^{\mathbb{H}[Z] - \mathrm{IC}}, \tag{2}$$

for every z *that has nonzero probability conditioned on events* $Out_i = \mathcal{A}_i(\mathbf{p}, r)$*,* $P_i = p_i$*, and* $R = r$*. Then, if* B *denotes the per-round communication link bandwidth, it holds that*

$$T = \Omega\left(\frac{\mathrm{IC}}{Bk}\right). \tag{3}$$

Intuition. We can think of Premise (1) as bounding the initial knowledge of the machines about the random variable Z. On the other hand, Premise (2) says that at least one machine is able to increase its knowledge on the value of Z eventually, which we formalize by conditioning on its output in addition to the initial knowledge. Then, if there is a large set (called *Good*) of inputs where these premises hold, then our theorem says that the worst-case time of the algorithm must be sufficiently large. These insights are formally captured by the *self-information* or *surprisal* of an event E, which is defined as $\log_2(1/\Pr[E])$ [33] and measures the "amount of surprise" or information contained in observing E. Premises (1) and (2) imply that, from some machine M_i's point of view, the occurrence of $\{Z = z\}$ is "$\Omega(\mathrm{IC})$ more surprising" given its initial knowledge, compared to observing this event after computing the output. We can show that this surprisal change IC bounds from below the maximum communication cost over all machines. In this light, (3) says that the run time of the algorithm is roughly a $(1/kB)$-fraction of the maximum expected information cost.

2.2 Proof of the General Lower Bound Theorem

In the proof of Theorem 2.1 we make use of some standard definitions in information theory, such as entropy, denoted with $\mathbb{H}$, and mutual information, denoted with $\mathbb{I}$ (see, e.g., [8]).

Critical Index. For a given input graph partition $\mathbf{p}$ and a random string r, we are interested in identifying the machine that has the maximum expected value of the amount of information that its output reveals about the random variable Z. This motivates us to define the *critical index* function as

$$\ell(\mathbf{p}, r) := \arg\max_{1 \leqslant i \leqslant k} \mathbb{I}[Out_i; Z \mid p_i, r], \tag{4}$$

and define random variables $\Pi_*(\mathbf{p}, r) = \Pi_{\ell(\mathbf{p}, r)}(\mathbf{p}, r)$ and $Out_*(\mathbf{p}, r) = Out_{\ell(\mathbf{p}, r)}(\mathbf{p}, r)$. Intuitively speaking, for each $(\mathbf{p}, r) \in \mathcal{GP} \times \mathcal{RS}$, the random variable Out_* is the output of the machine M_i (where i depends on $\mathbf{p}, r$) that attains the maximum mutual information between its output and the random variable Z. For a given $(\mathbf{p}, r)$, we use $p_* = p_{\ell(\mathbf{p}, r)}$ to denote the input partition of machine $M_{\ell(\mathbf{p}, r)}$. Note that Z depends *only* on the input graph, whereas Π_*, P_*, and Out_* depend on the input graph and, in addition, also on the chosen partition $\mathbf{p}$ and random string r. From (4), we immediately obtain the following property of the critical index.

Observation 1. *For all* $(\mathbf{p}, r) \in \mathcal{GP} \times \mathcal{RS}$*, and for all* $i \in [k]$*, it holds that*

$$\mathbb{I}[Out_*; Z \mid p_*, r] \geqslant \mathbb{I}[Out_i; Z \mid p_i, r],$$

where $p_* = p_{\ell(\mathbf{p}, r)}$ *and* $\mathbf{p} = (p_1, \ldots, p_{\ell(\mathbf{p}, r)}, \ldots, p_k)$*.*

Lemma 2.2. *For every* $(\mathbf{p}, r) \in \mathcal{GP} \times \mathcal{RS}$ *where* $\mathbf{p} = (p_1, \ldots, p_*, \ldots, p_k)$*, it holds that*

$$\mathbb{I}[\Pi_*; Z \mid p_*, r] \geqslant \mathbb{I}[Out_*; Z \mid p_*, r].$$

Lemma 2.3. *For all* $(\mathbf{p}, r) \in Good$ *where* $\mathbf{p} = (p_1, \ldots, p_k)$*, there is an* $i \in [k]$ *(which satisfies (1) and (2) in the premise of the theorem) such that* $\mathbb{I}[Out_i; Z \mid p_i, r] \geqslant \mathrm{IC} - o(\mathrm{IC})$*.*

Proof. For a given $(\mathbf{p}, r) \in Good$, let M_i be a machine satisfying (2) (in addition to (1)). By definition,

$$\mathbb{I}[Out_i; Z \mid p_i, r] = \mathbb{H}[Z \mid p_i, r] - \mathbb{H}[Z \mid Out_i, p_i, r]. \tag{5}$$

We will now bound the terms on the right-hand side. By definition, we obtain

$$\begin{aligned}
\mathbb{H}[Z \mid p_i, r] &= -\sum_z \Pr[Z = z \mid p_i, r] \log_2 \Pr[Z = z \mid p_i, r] \\
&\geqslant (\mathbb{H}[Z] - o(\mathrm{IC})) \sum_z \Pr[Z = z \mid p_i, r] \qquad \text{(by (1))} \\
&= \mathbb{H}[Z] - o(\mathrm{IC}),
\end{aligned} \tag{6}$$

where the last inequality follows from $\sum_z \Pr[Z = z \mid p_i, r] = 1$.

In the remainder of the proof, we derive an upper bound on $\mathbb{H}[Z \mid Out_i, p_i, r]$. Since

$$\mathbb{H}[Z \mid Out_i, p_i, r] \leqslant \mathbb{H}[Z \mid Out_i], \tag{7}$$

we will proceed by proving an upper bound on the latter term. To simplify the notation, we use "$\mathcal{A}_i(\mathbf{p}, r)$" as a shorthand for the event "$Out_i = \mathcal{A}_i(\mathbf{p}, r)$". By definition, we have

$$\begin{aligned}
\mathbb{H}[Z \mid Out_i] &= \sum_{(\mathbf{p}, r)} \Pr[\mathcal{A}_i(\mathbf{p}, r)]\, \mathbb{H}[Z \mid \mathcal{A}_i(\mathbf{p}, r)] \\
&= \sum_{(\mathbf{p}, r) \in Good} \Pr[\mathcal{A}_i(\mathbf{p}, r)]\, \mathbb{H}[Z \mid \mathcal{A}_i(\mathbf{p}, r)] \\
&\quad + \sum_{(\mathbf{p}, r) \notin Good} \Pr[\mathcal{A}_i(\mathbf{p}, r)]\, \mathbb{H}[Z \mid \mathcal{A}_i(\mathbf{p}, r)] \\
&\leqslant \sum_{(\mathbf{p}, r) \in Good} \Pr[\mathcal{A}_i(\mathbf{p}, r)]\, \mathbb{H}[Z \mid \mathcal{A}_i(\mathbf{p}, r)] \\
&\quad + \mathbb{H}[Z] \left(\sum_{(\mathbf{p}, r) \notin Good} \Pr[\mathcal{A}_i(\mathbf{p}, r)] \right),
\end{aligned} \tag{8}$$

where the last inequality follows from $\mathbb{H}[Z] \geqslant \mathbb{H}[Z \mid \mathcal{A}_i(\mathbf{p}, r)]$. Intuitively speaking, the first sum in (8) represents the remaining

uncertainty of Z upon termination, assuming machines start with a hard input assignment (i.e., in *Good*), whereas the second term is weighted by the probability that either the input was easy or the algorithm failed (i.e. $\notin$ *Good*). The following claim bounds the entropy term in the first sum of (8), where $(\mathbf{p}, r)$ is restricted to the set *Good*.

Claim 1. $\mathbb{H}[Z \mid \mathcal{A}_i(\mathbf{p}, r)] \leqslant \mathbb{H}[Z] - \text{IC}$.

We will now derive an upper bound on the second sum in (8).

Claim 2. $\sum_{(\mathbf{p},r)\notin Good} \Pr[\mathcal{A}_i(\mathbf{p}, r)] \leqslant \epsilon + n^{-\Omega(1)}$.

Plugging the bounds in Claims 1 and 2 into (8), we get

$$\mathbb{H}[Z \mid Out_i] \leqslant (\mathbb{H}[Z] - \text{IC}) \sum_{(\mathbf{p},r)\in Good} \Pr[\mathcal{A}_i(\mathbf{p}, r)] + \mathbb{H}[Z]\Big(\epsilon + n^{-\Omega(1)}\Big)$$
$$\leqslant (\mathbb{H}[Z] - \text{IC}) + \mathbb{H}[Z]\Big(\epsilon + n^{-\Omega(1)}\Big).$$

Assuming a sufficiently large constant in the exponent of $n^{-\Omega(1)}$, we observe that $\mathbb{H}[Z] \cdot n^{-\Omega(1)} = o(1)$ since Z depends only on the input graph. By the premise of Theorem 2.1, we have $\epsilon = o(\text{IC}/\mathbb{H}[Z])$ and $\text{IC} \leqslant \mathbb{H}[Z]$, hence $\epsilon \cdot \mathbb{H}[Z] = o(\text{IC})$. From this and (7) we conclude that

$$\mathbb{H}[Z \mid Out_i, p_i, r] \leqslant \mathbb{H}[Z] - \text{IC} + o(\text{IC}).$$

Plugging this upper bound and the lower bound of (6) into the right-hand side of (5), completes the proof of Lemma 2.3. □

Recall that Lemma 2.2 holds for any $(\mathbf{p}, r) \in \mathcal{GP} \times \mathcal{RS}$; in particular, even if we restrict our choice to the set *Good*. Thus, for $(\mathbf{p}, r) \in Good$, where $\mathbf{p} = (p_1, \ldots, p_k)$, let $i \in [k]$ be the index for which Lemma 2.3 holds (which is the index of the machine satisfying Premises (1) and (2)). This yields

$$\begin{aligned}
\mathbb{H}[\Pi_* \mid p_*, r] &\geqslant \mathbb{I}[\Pi_*; Z \mid p_*, r] \\
&\geqslant \mathbb{I}[Out_*; Z \mid p_*, r] && \text{(by Lemma 2.2)} \\
&\geqslant \mathbb{I}[Out_i; Z \mid p_i, r] && \text{(by Obs. 1)} \\
&\geqslant \text{IC} - o(\text{IC}), && (9)
\end{aligned}$$

where the last inequality follows from Lemma 2.3. To complete the proof of Theorem 2.1, we will argue that the worst-case run time needs to be large, as otherwise the entropy of machine $M_{\ell(\mathbf{p},r)}$'s transcript Π_* would be less than IC$-o$(IC). The value of $\mathbb{H}[\Pi_* \mid p_*, r]$ is maximized if the distribution of $(\Pi_* \mid p_*, r)$ is uniform over all possible choices. In the next lemma we show that, during T rounds of the algorithm, the transcript can take at most $2^{(B+1)(k-1)T}$ distinct values, and thus

$$\mathbb{H}[\Pi_* \mid p*, r] \leqslant \log_2\Big(2^{(B+1)(k-1)T}\Big) = O(B\ k\ T). \qquad (10)$$

Lemma 2.4. *Suppose that some machine M_i can receive a message of at most B bits on each of its $k-1$ links in a single round. Let Γ be the bits received by M_i over its $k-1$ links during T rounds. Then, Γ can take at most $2^{(k-1)(B+1)T}$ distinct values.*

Recall that the run time T is the maximum time required by any machine M_i, over all random strings and input assignments, i.e., $T = \max_{(\mathbf{p},r)} T(\mathbf{p}, r)$. Combining (9) and (10), it follows that

$$T = \max_{(\mathbf{p},r)} T(\mathbf{p}, r) = \Omega\left(\frac{IC}{Bk}\right).$$

2.3 A Lower Bound for PageRank Computation

Theorem 2.5. *Let $\mathcal{A}$ be an algorithm that computes a δ-approximation of the* PageRank *vector of an n-node graph for a small constant $\delta > 0$ (depending on the reset probability), and suppose that $\mathcal{A}$ succeeds with probability $\geqslant 1 - o(1/k)$. Then, the run time of $\mathcal{A}$ is $\Omega\Big(\frac{n}{B \cdot k^2}\Big)$, assuming a communication link bandwidth of B bits per round and $k = \Omega(\log^2 n)$ machines. This holds even when the input graph is assigned to the machines via random vertex partitioning.*

We first give a high-level overview of the proof. As input graph G, we construct a weakly connected directed graph where the direction of certain "important" edges is determined by a random bit vector, and assign random IDs to all the vertices. Flipping the direction of an important edge changes the PageRank of connected vertices by a constant factor and hence any (correct) algorithm needs to know about these edge directions. It is crucial that the vertex IDs are chosen randomly, to ensure that knowing just the direction of important edges is not sufficient for computing the PageRank of the adjacent nodes, as these random vertex IDs "obfuscate the position" of a vertex in the graph. This means that a machine needs to know both, the direction of an important edge and the IDs of the connected vertices to be able to output a correct result. By using a Chernoff bound, we can show that the random vertex partitioning of the input graph does not reveal too many edge-directions together with the matching vertex IDs to a single machine. This sets the stage for applying our generic lower bound theorem (Theorem 2.1) to obtain a lower bound on the run time.

The Lower Bound Graph. We consider the following directed graph H (see Figure 1) of n vertices and $m = n-1$ edges; for simplicity, assume that $m/4$ is an integer. Let $X = \{x_1, x_2, \ldots, x_{m/4}\}$, $U = \{u_1, u_2, \ldots, u_{m/4}\}$, $T = \{t_1, t_2, \ldots, t_{m/4}\}$, $V = \{v_1, v_2, \ldots, v_{m/4}\}$, and let $V(G) = \{X \cup U \cup T \cup V \cup \{w\}\}$. The edges between these vertices are given as follows: For $1 \leqslant i \leqslant m/4$, there is a directed edge $u_i \to t_i$, a directed edge $t_i \to v_i$, and a directed edge $v_i \to w$. The edges between u_i and x_i (these are the "important" edges mentioned above) are determined by a bit vector $\mathbf{b}$ of length $m/4$ where each entry b_i of $\mathbf{b}$ is determined by a fair coin flip: If $b_i = 0$ then there is an edge $u_i \to x_i$, otherwise there is an edge $x_i \to u_i$. Lemma 2.6 shows that, for any $1 \leqslant i \leqslant m/4$ and for any $\epsilon < 1$, there is a constant factor separation between the PageRank of any node v_i if we switch the direction of the edge between x_i and u_i.

Lemma 2.6. *The following holds for the* PageRank *value of vertices v_i of G, for $1 \leqslant i \leqslant n/4$: If $b_i = 0$, then* $\text{PageRank}(v_i) = \frac{(2.5-2\epsilon+\epsilon^2/2)\epsilon}{n}$. *Otherwise, if $b_i = 1$, then* $\text{PageRank}(v_i) \geqslant \frac{(3-3\epsilon+\epsilon^2)\epsilon}{n}$. *For any $\epsilon < 1$, there is a constant factor (where the constant depends on ϵ) separation between the two cases.*

The Input Graph Distribution. We now build our input graph G as follows. Let $m = n - 1$, and let ID be the random variable representing a set of n unique integers chosen uniformly at random from $\{S \subset [1, \text{poly}(n)] \colon |S| = n\}$. Assigning each vertex of H a unique integer from ID (in an arbitrary predetermined way) yields a graph G. Let $\mathcal{G}$ denote the set of graphs G determined by all possible (different) ID assignments to all possible instances of H considering all possible edge directions. Let $\mathcal{GP}$ be the set of all input graph partitions (i.e., the set of all graphs in $\mathcal{G}$ and all their possible input partitions) among the k machines, and let $\mathcal{RS}$ be

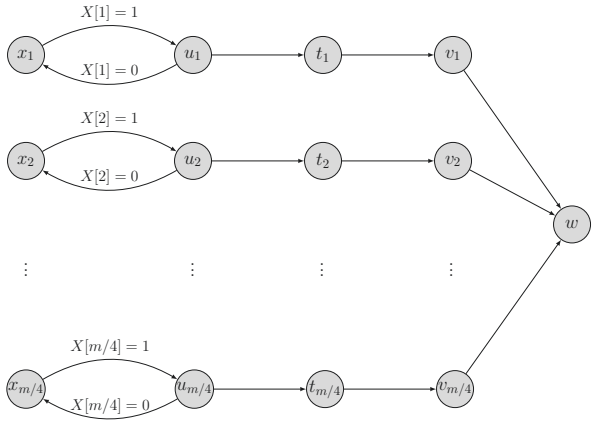

Figure 1: The graph H used to derive a lower bound on the round complexity of PageRank computations.

the set of all random strings used by a given PageRank algorithm $\mathcal{A}$. Let $Bal \subseteq \mathcal{GP}$ be the set of all input partitions where each machine receives $\tilde{\Theta}(n/k)$ vertices of the input graph. Note that $(\mathbf{p}, r) \in \mathcal{GP} \times \mathcal{RS}$ fully determines the run of $\mathcal{A}$. We assume that each machine M_i outputs a set $\{(\pi_1, id_1), \ldots, (\pi_\ell, id_\ell)\}$, where π_j refers to the PageRank value of the vertex with ID id_j. Note that we do not make assumptions neither on which machine being the one that outputs the PageRank of a specific vertex v (which could be a machine that holds no initial knowledge about v and its ID), nor on the individual sizes of these output sets.

Discovering Weakly Connected Paths of Vertices. By the random vertex partitioning, each machine M_i initially holds $\tilde{\Theta}(n/k)$ vertices in total. More specifically, M_i receives random sets $X_i \subseteq X$, $U_i \subseteq U$, $T_i \subseteq T$, and $V_i \subseteq V$, each containing $O(n\log(n)/k)$ vertices. As machine M_i also gets to know the incident edges of these vertices, M_i can locally check if a path induced by some $(x_{j_1}, u_{j_2}, t_{j_3}, v_{j_4}) \in X_i \times U_i \times T_i \times V_i$ is weakly connected, i.e., $j_1 = \cdots = j_4$. Since M_i learns the output pair $(\text{PageRank}(v), id_v)$ at zero cost, we upper bound the number of such paths that the machines learn initially by using a Chernoff bound. That is, we show that each machine learns at most $O\left(\frac{n\log n}{k^{3/2}}\right)$ paths.

Lemma 2.7. *With probability at least $1 - n^{-4}$, the initial graph partition reveals at most $O\left(\frac{n\log n}{k^2}\right)$ weakly connected paths between vertices in X and V to every machine.*

Good Inputs. We define $Good \subseteq Bal \times \mathcal{RS}$ to be the set of all (balanced) inputs and random strings where (1) $\mathcal{A}$ correctly outputs the PageRank of each vertex, (2) partition $\mathbf{p}$ is "balanced", i.e., each machine is assigned $O(n\log n/k)$ vertices (and hence $O(n\log n/k)$ edges since $m = O(n)$), and (3) the partitioning is such that each machine knows at most $O((n\log n)/k^2)$ weakly connected paths initially; we define $Bad = \mathcal{GP} \times \mathcal{RS} \setminus Good$.

Lemma 2.8. *(A) For any $(\mathbf{p}, r) \in Good$, algorithm $\mathcal{A}$ is correct and there must be at least one machine M_i whose output list contains $\Omega(n/k)$ vertices of V. (B) $|Good| \geqslant \left(1 - o(1/k) - n^{-\Omega(1)}\right)|\mathcal{GP}\times\mathcal{RS}|$.*

To instantiate Theorem 2.1, we show in Lemma 2.9 and Lemma 2.10 that we can satisfy the Premises (1) and (2), by setting $\text{IC} = m/4k = \Theta(n/k)$. Plugging the above value of IC in (3) then gives the claimed lower bound.

Lemma 2.9. *Let Z be the random variable representing the set of pairs $\{(b_1, v_1), \ldots, (b_{m/4}, v_{m/4})\}$, where b_j refers to the direction of the edge (x_j, u_j) in the weakly connected path (x_j, u_j, t_j, v_j) of the input graph of Figure 1. Then, for each $(\mathbf{p}, r) \in Good$, where $\mathbf{p} = (p_1, \ldots, p_k)$, and for every possible choice of z, it holds that $Pr[Z = z \mid p_i, r] \leqslant 2^{-(m/4 - O(n\log(n)/k^2))}$.*

Lemma 2.10. *For each $(\mathbf{p}, r) \in Good$, where $\mathbf{p} = (p_1, \ldots, p_k)$, there exists a machine M_i with output $\mathcal{A}_i(\mathbf{p}, r)$ such that, for every choice of z for Z (defined in Lemma 2.9) that has nonzero probability conditioned on $\mathcal{A}_i(\mathbf{p}, r), p_i, r$, it holds that $Pr[Z = z \mid \mathcal{A}_i(\mathbf{p}, r), p_i, r] \geqslant 1/2^{\frac{m}{4} - \frac{m}{4k}}$.*

2.4 A Lower Bound for Triangle Enumeration

We first give a high-level overview of the proof. The input graphs that we use for our lower bounds are sampled according to the $G_{n,1/2}$ Erdös-Renyi random graph model. We will argue that enumerating triangles implies a large reduction of the entropy of the characteristic vector of edges Z, i.e., Z is a bit vector whose entries reflect the presence/absence of an edge in the input graph. We prove that initially the machines do not have significant knowledge of Z, which is equivalent to having a small probability for the event $\{Z = z\}$, for any z. Then, we show that any machine that outputs t/k triangles, for a parameter t, must have reduced its uncertainty about Z by approximately $(t/k)^{2/3}$ bits. In other words, the information obtained by such a machine throughout the course of the algorithm is high. We apply Theorem 2.1 to obtain a lower bound on the run time of any algorithm. This yields the following result.

Theorem 2.11. *There exists a class of graphs $\mathcal{G}$ of n nodes for which every distributed algorithm that solves triangle enumeration in the k-machine model has a time complexity of $\Omega\left(\frac{n^2}{B\cdot k^{5/3}}\right)$, assuming a link bandwidth of B bits per round, $k = \Omega(\log n)$ machines, and an error probability of $\epsilon = o(k^{-2/3})$. This holds even when the input graph is assigned to the machines via random vertex partitioning.*

The Input Graph Distribution. We choose our input graphs according to the Erdös-Renyi random graph model $G_{n,1/2}$, which samples an n-node graph where each possible edge is included independently with probability $1/2$. We use $\mathcal{GP}$ to denote the set of all possible partitions of all possible sampled n-node graphs and, similarly to before, denote the set of all random strings used by the algorithm by $\mathcal{RS}$.

Let Z be the characteristic vector of the edges[5] of the input graph G. Note that the execution of $\mathcal{A}$ is fully determined by the given graph input partition $\mathbf{p} = (p_1, \ldots, p_k) \in \mathcal{GP}$ and the shared (among all machines) random bit string $r \in \mathcal{RS}$, where $\mathcal{RS}$ is the set of all possible strings that are used as random bit string by the algorithm. Hence we have $|\mathcal{GP} \times \mathcal{RS}|$ possible outcomes when running $\mathcal{A}$ on a graph sampled from $\mathcal{G}$.

[5]The characteristic vector specifies the graph G. Order the $\binom{n}{2}$ possible edges in some fixed ordering; if the jth edge in this ordering appears in G, then $Z_j = 1$, otherwise it is 0.

Good Inputs. We define $Good \subseteq \mathcal{GP} \times \mathcal{RS}$ to be the set of input pairs $(\mathbf{p}, r)$ such that (1) $\mathcal{A}$ performs correctly for the graph partition $\mathbf{p}$ of graph G and the random string r, (2) partition $\mathbf{p}$ is "balanced", i.e., each machine is assigned $O(n\log(n)/k)$ vertices (and hence $O(n^2\log(n)/k)$ edges), and (3) G has $\geqslant t$ triangles, for some fixed $t = \Theta(\binom{n}{3})$.

Similarly as for Theorem 2.5, we prove that $\Pr[Z = z \mid p_i, r] \leqslant 1/2^{\binom{n}{2}-O(n^2\log n/k)}$ and $\Pr[Z = z \mid o_i, p_i, r] \geqslant 1/2^{\binom{n}{2}-\frac{1}{3}\left(\frac{t}{k}-O\left(\frac{n^2\log n}{k}\right)\right)^{2/3}}$. Then, setting $\mathrm{IC} = \Theta(n^2/k^{2/3})$, and applying Theorem 2.1, completes the proof of Theorem 2.11.

A tight lower bound in the congested clique. Our analysis extends in a straightforward way to the congested clique model where, in a synchronous complete network of n machines, every machine u receives exactly one input vertex of the input graph and gets to know all its incident edges. Together with the deterministic upper bound of $O(n^{1/3})$ shown in [11], we have the following.

Corollary 2.12. *The round complexity of enumerating all triangles in the congested clique of n nodes with high probability of success is $\Omega\left(\frac{n^{1/3}}{B}\right)$, assuming a link bandwidth of B bits. This bound is tight up to logarithmic factors.*

Message lower bounds. We have the following.

Corollary 2.13. *Let $\mathcal{A}$ by any algorithm that enumerates all triangles with high probability and terminates in $\tilde{O}(\frac{n^2}{k^{5/3}})$ rounds. Then, the total message complexity in the k-machine model of $\mathcal{A}$ is $\tilde{\Omega}(n^2k^{1/3})$. For $\tilde{O}(n^{1/3})$-rounds algorithms in the congested clique, the message complexity is $\tilde{\Omega}(n^{7/3})$.*

3 UPPER BOUNDS

3.1 An Almost Optimal Algorithm for PageRank Computation

In this section we present a simple distributed algorithm to compute the PageRank vector of an input graph in the k-machine model. This algorithm has a round complexity of $\tilde{O}(n/k^2)$, which significantly improves over the previous $\tilde{O}(n/k)$-round solution [19].

We first recall the distributed random walk-based Monte-Carlo algorithm for computing PageRank, for a given reset probability ϵ, as described in [10]. This algorithm is designed and analyzed in the standard CONGEST model, where each vertex of the graph executes the algorithm. The algorithm is as follows. Initially, each vertex creates $c\log n$ random walk tokens, where $c = c(\epsilon)$ is a parameter defined in [10] ($c(\epsilon)$ is inversely proportional to ϵ), which are then forwarded according to the following process: when a node u receives some random walk token ρ, it terminates the token with probability ϵ and, with probability $1-\epsilon$, forwards it to a neighbor of u chosen uniformly at random. Each node keeps a variable ψ_v, for each of its nodes v, which counts the number of random walk tokens that were addressed to v (i.e., the total number of all random walks that visit v). Each node v then estimates its PageRank by computing $\frac{\epsilon\psi_v}{cn\log n}$. It can be shown that this estimate gives a δ-approximation, for any constant $\delta > 0$, to the PageRank value of each node v with high probability, and that this algorithm terminates in $O(\log n/\epsilon)$ rounds with high probability [10]. The key idea to obtain such a fast runtime is to send only the *counts* of the random walks, instead of keeping track of the random walks from different sources. Clearly, only the number (i.e., count) of the random walks visiting a node at any step is required to estimate the PageRank. In the full paper, we describe why a naïve implementation of the above approach only yields a running time of $\tilde{O}(n/k)$.

To avoid the pitfalls of a naïve implementation, we describe an approach that directly exploits the k-machine model. On the one hand, our goal is to reduce the total amount of communication while, on the other hand, we need to ensures that the incurred message complexity is balanced for the available machines. This motivates us to treat vertices differently depending on how many tokens they hold. We say that a vertex u has *low-load in iteration* r if, conceptually, the machine that hosts u considers $\geqslant k$ tokens to be held at u. Otherwise, we say that u has *high-load in iteration* r. Note that, throughout the course of our algorithm, the value of tokens$[v]$ depends on the topology of the input graph and hence a vertex can change its status w.r.t. being a high-load or low-load vertex.

In our algorithm (Algorithm 1), each machine M stores an array tokens$[u]$, which has an entry for each vertex u hosted at M. Initially, we generate $\Theta(\log n)$ tokens for each vertex which we use as the initialization value of tokens. Then, we mimic the (parallel) random walk steps of [10] by performing $\Theta(\log(n)/\epsilon)$ iterations where, in each iteration, each machine M first considers the tokens stored for its low-load vertices. For each such token held at one of its vertices u, M uniformly at random selects a neighboring vertex v and keeps track of how many tokens have chosen v in a separate array $\alpha[v]$. In particular, M also increments the same entry $\alpha[v]$ if v is chosen as the destination for some token of a distinct low-load vertex w at M. Then, M sends a message $\langle\alpha[v], \text{dest:}v\rangle$ for each v where $\alpha[v]$ is nonzero, which is subdelivered to the destination machine using random routing (cf. Lemma 3.2). This ensures that all the messages are delivered in $\tilde{O}(n/k^2)$ rounds.

We now describe how high-load vertices are processed, each of which can hold up to $O(n\log n)$ tokens. To avoid potentially sending a large number of messages for a single high-load vertex u, machine M considers the index set I of machines that host at least one neighbor of u. Then, for each token of u, machine M samples an index from I and keeps track of these counts in an array β, which has an entry for each machine in I. Finally, M generates one message of type $\langle\beta[j], \text{src:}u\rangle$, for each entry j where $\beta[j] > 0$ and sends this count message directly to the respective destination machine. We show that these messages can be delivered in $\tilde{O}(n/k^2)$ rounds by proving that, with high probability, each machine holds $\tilde{O}(n/k^2)$ high-load vertices in any given iteration of the algorithm.

Lemma 3.1. *Every machine M_i sends at most $\tilde{O}(n/k)$ messages in any iteration r with high probability.*

A key ingredient in the analysis of the algorithm is the following simple lemma, which quantifies how fast some specific routing can be done in the k-machine model.

Lemma 3.2. *Consider a complete network of k machines, where each link can carry one message of $O(\text{polylog}\, n)$ bits at each round. If each machine is source of $O(x)$ messages whose destinations are distributed independently and uniformly at random, or each machine is*

Algorithm 1 Computing the PageRank with reset probability $\epsilon > 0$. Code for machine M_i.

Let V_i denote the vertices hosted by machine M_i
Initalize array tokens$[u] \leftarrow \lceil c \log n \rceil$, for $u \in V_i$, where $c > 0$ is a suitable constant ▷ tokens$[u]$ represents the current number of tokens at vertex u
for $\Theta(\log(n)/\epsilon)$ iterations **do**
  for $u \in V_i$ **do**
    sample t from distribution $Binomial(\text{tokens}[u], \epsilon)$
    tokens$[u] \leftarrow$ tokens$[u] - t$ ▷ Terminate each token with probability ϵ

  Initialize array $\alpha[v] \leftarrow 0$, for each $v \in V$ ▷ Process the low-load vertices
  for each vertex $u \in V_i$ where tokens$[u] < k$ **do**
    let $N_u \subseteq V$ be the set of neighbors of vertex u
    while tokens$[u] > 0$ **do**
      sample v uniformly at random from N_u
      $\alpha[v] \leftarrow \alpha[v] + 1$
      tokens$[u] \leftarrow$ tokens$[u] - 1$
  for each $v \in V_i$ where $\alpha[v] > 0$ **do**
    send message $\langle \alpha[v], \text{dest: } v \rangle$ **to** the machine hosting vertex v using random routing

  for each vertex $u \in V_i$ where tokens$[u] \geqslant k$ **do** ▷ Process the high-load vertices
    let $I \subseteq [k]$ be the index set of the machines that host a neighbor of u
    initialize array $\beta[j] \leftarrow 0$, for each $j \in I$
    while tokens$[u] > 0$ **do**
      let $n_{j,u}$ be number of neighbors of u hosted at machine M_j and let d_u be u's degree
      sample index j from distribution $\left(\frac{n_{1,u}}{d_u}, \ldots, \frac{n_{k,u}}{d_u}\right)$
      $\beta[j] \leftarrow \beta[j] + 1$
      tokens$[u] \leftarrow$ tokens$[u] - 1$
    for each $j \in I$ where $\beta[j] > 0$ **do**
      send message $\langle \beta[j], \text{src: } u \rangle$ **to** machine M_j

  for each received message of type $\langle c_w, \text{dest: } w \rangle$ **do**
    tokens$[w] \leftarrow$ tokens$[w] + c_w$
  for each received message of type $\langle c_v, \text{src: } v \rangle$ **do**
    while $c_v > 0$ **do**
      let $N_v \subseteq V$ be the set of neighbors of v hosted at M_i
      sample w uniformly at random from N_v
      tokens$[w] \leftarrow$ tokens$[w] + 1$
      $c_v \leftarrow c_v - 1$

destination of $O(x)$ messages whose sources are distributed independently and uniformly at random, then all the messages can be routed in $O((x \log x)/k)$ rounds w.h.p.

Lemma 3.3. *Consider any iteration r of Algorithm 1. Then, with high probability, all messages generated at iteration r can be delivered in $\tilde{O}(n/k^2)$ rounds.*

From Lemma 3.3 we conclude that all messages generated in a single iteration of Algorithm 1 can be delivered in $\tilde{O}(n/k^2)$ rounds with high probability. A union bound implies the following result.

Theorem 3.4. *Algorithm 1 computes a δ-approximation of the PageRank vector of an n-node graph in the k-machine model with high probability in $\tilde{O}(n/k^2)$ rounds, for any constant $\delta > 0$.*

3.2 An Almost Optimal Algorithm for Triangle Enumeration

In this section we present a randomized algorithm that enumerates all the triangles of an input graph $G = (V, E)$, and that terminates in $\tilde{O}(m/k^{5/3} + n/k^{4/3})$ rounds w.h.p. This bound does not match the (existential) $\tilde{\Omega}(m/k^{5/3})$ lower bound provided in Section 2.4 only for very sparse graphs.

Our algorithm is a generalization of the algorithm TriPartition of Dolev et al. for the congested clique model [11], with some crucial differences explained next. The key idea, which in its generality can be traced back to [2], is to partition the set V of nodes of G in $k^{1/3}$ subsets of $n/k^{1/3}$ nodes each, and to have each of the k machines to examine the edges between pairs of subsets in one of the $(k^{1/3})^3 = k$ possible triplets of subsets (repetitions are allowed).

The algorithm is as follows. Each node picks independently and uniformly at random one color from a set C of $k^{1/3}$ distinct colors through a hash function $h : V \rightarrow C$ initially known by all the machines. This gives rise to a color-based partition of the vertex set V into $k^{1/3}$ subsets of $\tilde{O}(n/k^{1/3})$ nodes each, w.h.p. A deterministic assignment of triplets of colors, hard-coded into the algorithm, logically assigns each of the k possible triplets of such subsets to one distinct machine. Each machine then collects all the edges between pairs of subsets in its triplet. This is accomplished in two steps: (1) For each of the edges it holds, each machine designates one random machine (among the k machines) as the *edge proxy* for that edge, and sends all its edges to the respective edge proxies. The designation of an edge itself is done by the following *proxy assignment rule* (this is necessary to avoid congestion at any one machine): A machine that has a node v whose degree is at least $2k \log n$ requests all other machines to designate the respective edge proxies for each of the incident edges of node v. If two machines request each other to designate the same edge (since their endpoints are hosted by the respective machines), then such a tie is broken randomly. (2) In the second step, all the machines collect their required edges from the respective proxies: since each edge proxy machine knows the hash function h as well as the deterministic assignment of triplets, it can send each edge to the machines where it is needed. Then, each machine simply enumerates all the triangles in its local subgraph.

We now argue that the above algorithm correctly enumerates all the triangles of a graph G, and analyze its round complexity. A key step in the analysis of the complexity is to bound from above the number of edges assigned to each machine. Observe that the number of edges between pairs of subsets of one triplet is no larger than the number of edges in the subgraph of G induced by the nodes of one triplet; in turn, because of the random color-based partition of the vertices made by the algorithm, the latter quantity is asymptotically equivalent to the number of edges in the subgraph

of G induced by a set of (in this case, $\tilde{O}(n/k^{1/3})$) randomly-chosen nodes of a graph. Thus, we shall concentrate on the latter quantity (which is of interest in its own right). To this end, we will use the following concentration result due to Rödl and Ruciński [34].

Proposition 3.5 ([34, Proposition 1]). *Let, for a graph $G = (V, E)$, $m < \eta n^2$, and let R be a random subset of V of size $|R| = t$ such that $t \geqslant 1/3\eta$. Let $e(G[R])$ denote the number of edges in the subgraph induced by R. Then, for some $c > 0$,*

$$\Pr\left[e(G[R]) > 3\eta t^2\right] < t \cdot e^{-ct}$$

We are now ready to analyze the algorithm.

Theorem 3.6. *There is a distributed algorithm for the k-machine model that enumerates all the triangles of an n-node, m-edge graph in $\tilde{O}(m/k^{5/3} + n/k^{4/3})$ rounds with high probability.*

4 CONCLUSIONS

We presented a general technique for proving lower bounds on the round complexity of distributed computations in a general message-passing model for large-scale computation, and showed its application for two prominent graph problems, PageRank and triangle enumeration. We also presented near-optimal algorithms for these problems, which can be efficiently implemented in practice.

Our lower bound technique works by relating the size of the output to the number of communication rounds needed, and could be useful in showing lower bounds for other problems where the output size is large (significantly more than the number of machines), such as sorting, matrix multiplication, shortest paths, matching, clustering, and densest subgraph.

ACKNOWLEDGMENTS

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 715672. G. Pandurangan and M. Scquizzato were also supported, in part, by NSF grants CCF-1527867, CCF-1540512, IIS-1633720, CCF-1717075, and by BSF grants 2008348 and 2016419. P. Robinson acknowledges the support of the Natural Sciences and Engineering Research Council of Canada (NSERC).

REFERENCES

[1] Giraph, http://giraph.apache.org/.
[2] F. N. Afrati and J. D. Ullman. Optimizing multiway joins in a Map-Reduce environment. *IEEE Trans. Knowl. Data Eng.*, 23(9):1282–1298, 2011.
[3] S. Bandyapadhyay, T. Inamdar, S. Pai, and S. V. Pemmaraju. Near-optimal clustering in the k-machine model. In *Proceedings of the 19th International Conference on Distributed Computing and Networking (ICDCN)*, pages 15:1–15:10, 2018.
[4] Z. Bar-Yossef, T. S. Jayram, R. Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. *J. Comput. Syst. Sci.*, 68(4):702–732, 2004.
[5] K. Censor-Hillel, P. Kaski, J. H. Korhonen, C. Lenzen, A. Paz, and J. Suomela. Algebraic methods in the congested clique. In *Proceedings of the 34th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 143–152, 2015.
[6] A. Ching, S. Edunov, M. Kabiljo, D. Logothetis, and S. Muthukrishnan. One trillion edges: Graph processing at facebook-scale. *PVLDB*, 8(12):1804–1815, 2015.
[7] F. Chung and O. Simpson. Distributed algorithms for finding local clusters using heat kernel pagerank. In *Proceedings of the 12th Workshop on Algorithms and Models for the Web-graph (WAW)*, pages 77–189, 2015.
[8] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley, 2006.
[9] A. Das Sarma, S. Holzer, L. Kor, A. Korman, D. Nanongkai, G. Pandurangan, D. Peleg, and R. Wattenhofer. Distributed verification and hardness of distributed approximation. *SIAM J. Comput.*, 41(5):1235–1265, 2012.
[10] A. Das Sarma, A. R. Molla, G. Pandurangan, and E. Upfal. Fast distributed PageRank computation. *Theor. Comput. Sci.*, 561:113–121, 2015.
[11] D. Dolev, C. Lenzen, and S. Peled. "Tri, tri again": Finding triangles and small subgraphs in a distributed setting. In *Proceedings of the 26th International Symposium on Distributed Computing (DISC)*, pages 195–209, 2012.
[12] A. Drucker, F. Kuhn, and R. Oshman. On the power of the congested clique model. In *Proceedings of the 33rd ACM Symposium on Principles of Distributed Computing (PODC)*, pages 367–376, 2014.
[13] M. Elkin, H. Klauck, D. Nanongkai, and G. Pandurangan. Can quantum communication speed up distributed computation? In *Proceedings of the 33rd ACM Symposium on Principles of Distributed Computing (PODC)*, pages 166–175, 2014.
[14] M. Ghaffari and M. Parter. MST in log-star rounds of congested clique. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 19–28, 2016.
[15] J. W. Hegeman, G. Pandurangan, S. V. Pemmaraju, V. B. Sardeshmukh, and M. Scquizzato. Toward optimal bounds in the congested clique: Graph connectivity and MST. In *Proceedings of the 34th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 91–100, 2015.
[16] T. Izumi and F. Le Gall. Triangle finding and listing in CONGEST networks. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 381–389, 2017.
[17] T. Jurdzinski and K. Nowicki. MST in $O(1)$ rounds of congested clique. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2620–2632, 2018.
[18] H. J. Karloff, S. Suri, and S. Vassilvitskii. A model of computation for MapReduce. In *Proceedings of the 21st annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 938–948, 2010.
[19] H. Klauck, D. Nanongkai, G. Pandurangan, and P. Robinson. Distributed computation of large-scale graph problems. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 391–410, 2015.
[20] C. Lenzen. Optimal deterministic routing and sorting on the congested clique. In *Proceedings of the 32nd ACM Symposium on Principles of Distributed Computing (PODC)*, pages 42–50, 2013.
[21] Z. Lotker, B. Patt-Shamir, E. Pavlov, and D. Peleg. Minimum-weight spanning tree construction in $O(\log \log n)$ communication rounds. *SIAM J. Comput.*, 35(1):120–131, 2005.
[22] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., 1996.
[23] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM International Conference on Management of Data (SIGMOD)*, pages 135–146, 2010.
[24] D. Nanongkai, A. D. Sarma, and G. Pandurangan. A tight unconditional lower bound on distributed randomwalk computation. In *Proceedings of the 30th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 257–266, 2011.
[25] R. Oshman. Communication complexity lower bounds in distributed message-passing. In *Proceedings of the 21st International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 14–17, 2014.
[26] G. Pandurangan, D. Peleg, and M. Scquizzato. Message lower bounds via efficient network synchronization. In *Proceedings of the 23rd International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 75–91, 2016.
[27] G. Pandurangan, P. Robinson, and M. Scquizzato. Fast distributed algorithms for connectivity and MST in large graphs. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 429–438, 2016.
[28] G. Pandurangan, P. Robinson, and M. Scquizzato. Tight bounds for distributed graph computations. *CoRR*, abs/1602.08481, 2016.
[29] G. Pandurangan, P. Robinson, and M. Scquizzato. Fast distributed algorithms for connectivity and MST in large graphs. *ACM Trans. Parallel Comput.*, 2018.
[30] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial and Applied Mathematics, 2000.
[31] D. Peleg and V. Rubinovich. A near-tight lower bound on the time complexity of distributed minimum-weight spanning tree construction. *SIAM J. Comput.*, 30(5):1427–1442, 2000.
[32] J. Qiu, S. Jha, A. Luckow, and G. C. Fox. Towards HPC-ABDS: An initial high-performance big data stack, in building robust big data ecosystem. 2014.
[33] F. M. Reza. *An introduction to information theory*. Courier Corporation, 1961.
[34] V. Rödl and A. Ruciński. Random graphs with monochromatic triangles in every edge coloring. *Random Struct. Algorithms*, 5(2):253–270, 1994.
[35] C. Scheideler. *Universal Routing Strategies for Interconnection Networks*, volume 1390 of *Lecture Notes in Computer Science*. Springer, 1998.
[36] L. G. Valiant. A scheme for fast parallel communication. *SIAM J. Comput.*, 11(2):350–361, 1982.
[37] L. G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, 1990.
[38] S. Vassilvitskii. Models for parallel computation (a hitchhikers' guide to massively parallel universes), http://grigory.us/blog/massively-parallel-universes/, 2015.
[39] D. P. Woodruff and Q. Zhang. When distributed computation is communication expensive. *Distrib. Comput.*, 30(5):309–323, 2017.