On the Distributed Complexity of Large-Scale Graph Computations

GOPAL PANDURANGAN, University of Houston
PETER ROBINSON, City University of Hong Kong
MICHELE SCQUIZZATO, University of Padova

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 $k \geq 2$ 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. This result 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 "cookbook" fashion to show distributed lower bounds for several problems, including non-graph problems. We present two applications by showing (almost) tight lower bounds on the round complexity of two fundamental graph problems, namely, *PageRank computation* and *triangle enumeration*. These applications show that our approach can yield lower bounds for problems where the application of communication complexity techniques seems not obvious or gives weak bounds, including and especially under a stochastic partition of the input.

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 that scales superlinearly in k, improving significantly over previous results [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 an approximation of 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

A preliminary version of this work [56] appeared in the *Proceedings of the 30th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'18)*. This work was supported, in part, by NSF grants CCF-1527867, CCF-1540512, IIS-1633720, CCF-1717075, by BSF grants 2008348 and 2016419, by University of Padova grant BIRD197859/19, by a grant from the City University of Hong Kong [Project No. 7200639/CS], and by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU11213620].

Authors' addresses: G. Pandurangan, Department of Computer Science, University of Houston, 3551 Cullen Blvd, Houston, TX 77204, USA; email: gopalpandurangan@gmail.com; P. Robinson, Department of Computer Science, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon, Hong Kong; email: peter.robinson@cityu.edu.hk; M. Scquizzato, Department of Mathematics, University of Padova, Via Trieste 63, 35121 Padova, Italy; email: scquizza@math.unipd.it.

1539-9087/2021/06-ART7 $15.00
https://doi.org/10.1145/3460900

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.

CCS Concepts: • **Theory of computation → Massively parallel algorithms**; **Distributed algorithms;**

Additional Key Words and Phrases: Distributed graph algorithms; PageRank; triangle enumeration; lower bounds

ACM Reference format:
Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. 2021. On the Distributed Complexity of Large-Scale Graph Computations. *ACM Trans. Parallel Comput.* 8, 2, Article 7 (June 2021), 28 pages.
https://doi.org/10.1145/3460900

1 INTRODUCTION

Distributed processing of large-scale data, in particular *graph data*,is becoming increasingly important with the rise of massive graphs such as the Web graph, social networks, biological networks, and other graph-structured data. Several large-scale graph processing systems such as Pregel [45] and Giraph [1] have been recently designed based on the *message-passing* distributed computing model [44, 58]. In these systems, the input graph, which is simply too large to fit into a single machine, is distributed across a group of machines connected via a communication network, and the machines jointly perform computation in a distributed fashion by exchanging messages. A key goal in distributed Big Data computing is to minimize the amount of communication across machines, as this typically dominates the overall cost of the computation [60].

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 adopted model, called the k-machine model [34], the input 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. 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. In particular, we would like to quantify how the round complexity scales with the number of machines used: More precisely, does the number of rounds scale linearly (or even super-linearly) in k? And what is the best possible round complexity for various problems?

The main contribution of this article 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*, introduced in Reference [34], and further investigated, e.g., in References [6, 26, 29, 35, 55]. The model consists of a set of $k \geq 2$ machines $\{M_1, M_2, \ldots, M_k\}$ that are 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, i.e., B bits can be transmitted over each link in each round; unless otherwise stated, we assume $B = \Theta(\text{polylog}\, n)$.[1] 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 an algorithm $\mathcal{A}$ is the maximum number of communication 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. This assumption is standard in the context of large-scale data processing. In fact, even assuming communication links with a bandwidth of order of gigabytes per second, the amount of data that typically has to be exchanged can be in the order of tera- or peta-bytes, and this generally dominates the overall computation cost [42]. However, we note that in all the algorithms of this article, 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.

In this article, we focus on investigating graph problems in this model. 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 $n \geq k$ (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 [45] and Giraph [1, 13], 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.[2] If a vertex v is assigned to machine M_i, then 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 Reference [34]); we shall assume this throughout the article. A convenient way to

[1] There is an alternative (but equivalent) way to view this communication restriction: Instead of putting a bandwidth restriction on the links, we can put a restriction on the amount of information that each *machine* can communicate (i.e., send/receive) in each round. The results that we obtain in the bandwidth-restricted model will also apply to the latter model [34]. Also, our bounds can be easily rewritten in terms of the B parameter.

[2] An alternative partitioning model is the so-called ***random edge partition (REP)*** model [55, 72]: Here, each edge of G is assigned independently and randomly to one of the k machines. One can extend our results to get bounds for the REP model, since it is easy to show that one can transform the input partition from one model to the other in $\tilde{O}(m/k^2 + n/k)$ rounds.

implement the RVP model is through hashing: each vertex (ID) is hashed to one of the k machines. Hence, if a machine knows a vertex ID, then it also knows where it is hashed to.

Eventually, in a computation each machine M_i, for each $1 \leq i \leq k$, must set a designated local output variable o_i (which need not depend on the set of vertices assigned to machine M_i), and the *output configuration* $o = \langle o_1, \ldots, o_k \rangle$ must satisfy certain feasibility conditions w.r.t. problem $\mathcal{P}$. For example, when considering the PageRank problem, each o_i corresponds to PageRank values (of one or more nodes), such that the PageRank value of each node of the graph should be output by at least one machine.

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 for 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, complementing the approach based on communication complexity (see, e.g., References [18, 21, 22, 34, 47, 50, 52, 53, 55, 59] and references therein). Using our approach, we show almost tight (up to logarithmic factors) lower bounds for two fundamental, seemingly unrelated, graph 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 model (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., Reference [72], which apply only under some worst-case partition).

To demonstrate the near-tightness of our lower bounds, we present optimal (up to a polylog(n) factor) distributed algorithms for these problems. All these algorithms exhibit a round complexity that 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.[3] 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 [34].

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 [34].

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.[4] (Notice that this does not contradict the result of Reference [21], which states that proving any super-constant lower bound for the congested clique would give new lower bounds in circuit complexity: In particular, because of the size required by any solution for triangle enumeration, Remark 3 in Reference [21] 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, which, e.g., allowed only broadcast communication, or which applied only to deterministic algorithms [21], or for implementations of specific algorithms [12].)

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

[4]A preliminary version of this article appeared on arXiv [54], 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 [30] showed a lower bound of the form $\Omega(n^{1/3}/\log n)$ using our information-theoretic approach.

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.

1.3 Overview of Techniques

Lower Bounds. In Theorem 2.1, we give 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. This theorem gives two probabilistic bounds that must be satisfied 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 that can be used to lower bound the runtime. The proof of the General Lower Bound Theorem makes use of information-theoretic machinery, yet its application requires no knowledge of information theory.

The General Lower Bound Theorem gives, in a fairly straightforward way, non-trivial lower bounds for problems where the application of communication complexity techniques seems not obvious, including and especially under a stochastic/random partition of the input. As an example, the work of Klauck et al. [34] showed a (tight) lower bound of $\tilde{\Omega}(n/k^2)$ for connectivity by appealing to *random-partition* communication complexity. This involved proving a lower bound for the classical set disjointness function assuming the inputs are randomly—rather than adversarially—distributed to the players, and this required non-trivial work. However, a lower bound of the same form for MST can be shown directly via the General Lower Bound Theorem—with a possible lower bound graph being the complete graph with random edge weights.

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 article. 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, such as in Reference [34], where 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 that hosts the vertex (i.e., its home machine) must know at the end of the computation the status of all of its incident edges, that is, whether they belong to the ST or not. 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 [55]. However, the lower bound proof technique of this article applies to the less restrictive (and more natural) criterion that any machine can output any part of the solution. In Reference [7], a direct sum theorem is shown that yields a communication complexity lower bound for set disjointness. The method of Reference [7] can be applied to obtain lower bounds for functions 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 Reference [34] directly translates algorithms designed for a message passing model for network algorithms to the k-machine model, and almost all the previous algorithms [15, 34] were derived using this result. In contrast, the present article does not use the Conversion Theorem; instead, it gives direct solutions for the problems at hand in the k-machine model, 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*. 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 reassigning the *executions* of individual nodes uniformly at random among the machines. (Similar ideas have been used in parallel and distributed computation in different contexts; see, e.g., References [65, 66].) 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. For example, a simple use of this strategy in the triangle enumeration algorithm (see Section 3.2) is as follows: Each edge in the graph is assigned a random machine as its proxy; the proxy does computation "associated" with the edge. This alleviates the congestion associated with machines having high-degree nodes. A slightly more sophisticated use of randomized proxy computation is made in our PageRank algorithm (see Section 3.1).

1.4 Related Work

Klauck et al. [34] 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 article. They do not present any non-trivial lower bound for any of these problems. Also, as pointed out earlier, some lower bounds shown in Reference [34], 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 article. Pandurangan et al. [55] showed $\tilde{O}(n/k^2)$-round algorithms in the k-machine model for connectivity, MST, approximate min-cut, and other graph verification problems. Except for the randomized proxy computation, the algorithmic techniques used in Reference [55] do not apply for PageRank computation and triangle enumeration. This model has been further investigated in, e.g., References [6, 26, 29, 35].

Another popular model serving as an abstraction of many modern large-scale data processing frameworks is the **Massively Parallel Computation (MPC)** model [33]. The distinguishing feature of this model is that a single machine of a large cluster cannot store the entirety of the input, but just a sublinear fraction of it. In particular, if N denotes the size of the input, then the memory of each machine is assumed to have size $s = O(N^{1-\epsilon})$ for some constant $\epsilon > 0$. Observe that

limiting the amount of local memory also implicitly limits the communication bandwidth—you can only send what you have in your memory; and vice versa, if you have limited communication bandwidth as in the k-machine model and a fast algorithm, then you do not have time to accumulate a large amount of data in your local memory. Hence, these two models often lead to similar algorithm design challenges. However, because of the relatively large available communication bandwidth, there are barriers to achieving super-constant lower bounds in the MPC model, and thus only *conditional* lower bounds are known—see Reference [48] and references therein.

The k-machine model is also closely related to the classical CONGEST model [58], and in particular to the *congested clique* model, which recently has received considerable attention (see, e.g., References [12, 21, 25, 28, 32, 40, 41, 43, 46]). 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. However, all vertices assigned to a machine have to communicate through the links incident on this machine, which can limit the bandwidth—in contrast with 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 for the congested clique may not directly yield the fastest algorithm in the k-machine model [55].

For a more detailed comparison of the k-machine model with other parallel and distributed models for large-scale data processing, such as the MPC model [33], the **Bulk Synchronous Parallel (BSP)** model [67], the message-passing model [72], and the congested clique, we refer to References [55, 68].

PageRank and triangle enumeration have received considerable attention in other models of distributed computing—see, e.g., References [27, 36–38, 57] and references therein. However, none of these results and techniques therein can be translated to yield the bounds shown in this article.

1.5 Preliminaries

PageRank. PageRank is one of the most important measures to rank the importance of nodes in a graph and was first proposed to rank Web pages [11]. The PageRank of a graph $G = (V, E)$ is defined as follows: Let ϵ be a small fixed constant. The PageRank (vector) of a graph (e.g., see References [3, 5, 8, 17]) is the *stationary distribution* vector π of the following special type of random walk: At each step of the random walk, with probability ϵ the random walk restarts from a node chosen uniformly at random among all nodes in the graph, and with probability $1 - \epsilon$ the walk follows a randomly chosen outgoing (neighbor) edge from the current node and moves to that neighbor. ϵ is called the *reset* probability. The computation of PageRank and its variants has been of tremendous research interest in both academia and industry. For a detailed survey of PageRank see, e.g., References [8, 39].

There are mainly two broad approaches to the PageRank computation (see, e.g., Reference [4]). One is the use of linear algebraic techniques (e.g., the Power Iteration [51]), and the other is Monte Carlo method [3]. In the Monte Carlo method, the basic idea is to approximate PageRank by directly simulating the corresponding random walk and then estimating the stationary distribution with the performed walk's distribution [3, 19].

Triangle enumeration. The triangle enumeration problem is to enumerate all the triangles in a graph, where a triangle is a set of three vertices all adjacent to each other.[5] This problem has attracted much interest because of its numerous practical applications, including the analysis of social processes in networks [23, 71], community detection [10], dense subgraph mining [69], joins in databases [49], and the solution of systems of geometric constraints [24]. The interested reader may refer to References [9, 14] for additional applications.

Triangle detection and triangle counting are also well-studied problems, and potentially significantly easier than triangle enumeration; however, we emphasize that for many applications, including all the aforementioned ones, triangle detection or triangle counting is not enough, and a complete enumeration of all the triangles is required.

In general, the enumeration of small subgraphs, cliques, or triplets of vertices that consist of exactly two edges (usually called *open triads*), also has numerous applications [9, 14, 69, 70].

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 proof 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 ith entry of $\mathbf{p}$. 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 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. We use R to denote the random variable of the sampled public random string. Similarly as 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 $\mathbf{p} = (p_1, \ldots, p_i, \ldots, p_k)$ and $(\mathbf{p}, r) \in \mathcal{GP} \times \mathcal{RS}$. We use the random variable Out_i to denote the output of machine M_i when executing the given algorithm. To simplify notation, we simply write "x" to denote the event $\{X = x\}$, for random variable X. For technical reasons, we assume that the output also includes M_i's initial graph input p_i and the public random string r.[6]

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, where* $\mathrm{IC} \leq \mathbb{H}[Z]$ *and where* $\mathbb{H}[Z]$ *is the entropy of* Z. *Consider a* T*-round* ϵ*-error algorithm* $\mathcal{A}$, *for some* $\epsilon = o(\mathrm{IC}/\mathbb{H}[Z])$. *Let Good* $\subseteq \mathcal{GP} \times \mathcal{RS}$ *be a set of pairs* $(\mathbf{p}, r)$ *where* $\mathbf{p} = (p_1, \ldots, p_k)$ *is an input*

[5] Sometimes this problem is also referred to as *triangle listing*, although there is a small difference: In triangle listing the output must be generated and stored in memory, whereas in triangle enumeration the output is not required to be stored. This distinction is relevant in bounded-memory models.

[6] Any given algorithm can be modified to achieve this behavior without increasing the complexity.

partition and r is a public random string, and $|Good| \geq (1-\epsilon-n^{-\Omega(1)})|\mathcal{GP}\times\mathcal{RS}|$. 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

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

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

for every z that has nonzero probability conditioned on events $Out_i=\mathcal{A}_i(\mathbf{p},r)$, $P_i=p_i$, and $R=r$. Moreover, assume that, for all $(\mathbf{p},r)\in\mathcal{GP}\times\mathcal{RS}$, and every $i\in[k]$, it holds that $\mathbb{H}[Z]\geq\mathbb{H}[Z\mid Out_i=\mathcal{A}_i(\mathbf{p},r)]$. Then, if B denotes the per-round communication link bandwidth, then 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, which will usually be some function of the input graph. For instance, when considering triangle enumeration in Section 2.4, Z will be the list of all edges. However, Premise (2) shows that at least one machine is able to increase its knowledge about the value of Z eventually, which we formalize by conditioning on its output in addition to the initial knowledge. In the context of triangle enumeration, this means that some machine must learn about many edges in the graph. 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 $\log_2(1/\Pr[E])$ [61] and measures the "amount of surprise" or information contained in observing an event 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. Consequently, (3) tells us that the running time of the algorithm is roughly a $(1/kB)$-fraction of the maximum expected information cost.

We point out that the premise $\mathbb{H}[Z]\leq\mathbb{H}[Z\mid Out_i=\mathcal{A}_i(\mathbf{p},r)]$ turns out to be a very minor restriction: For the choices of Z in our applications of this theorem, it is immediate that conditioning on the output of one machine does not *increase* the (expected) uncertainty of Z.

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, which we now recall (and which can be found, e.g., in Reference [16]). Consider random variables X, Y, and W. The *entropy of X* is defined as $\mathbb{H}[X]=-\sum_x \Pr[X=x]\log_2\Pr[X=x]$, and the *conditional entropy* is defined as

$$\mathbb{H}[X\mid Y]=\sum_y \Pr[Y=y]\,\mathbb{H}[X\mid Y=y]. \tag{4}$$

The *mutual information between X and Y given some event $\{W=w\}$* is denoted by $\mathbb{I}[X;Y\mid W=w]$, and given by

$$\mathbb{I}[X;Y\mid W=w]=\mathbb{H}[X\mid W=w]-\mathbb{H}[X\mid Y,W=w]. \tag{5}$$

From this it immediately follows that

$$\mathbb{H}[X\mid W=w]\geq\mathbb{I}[X;Y\mid W=w]. \tag{6}$$

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 transcript reveals about the random variable Z. This motivates us to define the *critical index* function as

$$\ell(\mathbf{p}, r) := \arg\max_{1 \le i \le k} \mathbb{I}[\Pi_i; Z \mid p_i, r], \tag{7}$$

and define random variables

$$\Pi_*(\mathbf{p}, r) = \Pi_{\ell(\mathbf{p},r)}(\mathbf{p}, r) \text{ and } Out_*(\mathbf{p}, r) = Out_{\ell(\mathbf{p},r)}(\mathbf{p}, r). \tag{8}$$

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

$$p_* = p_{\ell(\mathbf{p},r)} \tag{9}$$

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.

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

$$\mathbb{I}[\Pi_*; Z \mid p_*, r] \ge \max_{1 \le i \le k} \mathbb{I}[Out_i; Z \mid p_i, r].$$

Proof. Consider a $(\mathbf{p}, r) \in \mathcal{GP} \times \mathcal{RS}$ as described in the premise of the lemma. It holds that

$$\begin{aligned} \mathbb{I}[\Pi_*; Z \mid p_*, r] &\ge \max_{1 \le i \le k} \mathbb{I}[\Pi_i; Z \mid p_i, r] && \text{(by (7))} \\ &= \max_{1 \le i \le k} (\mathbb{H}[Z \mid p_i, r] - \mathbb{H}[Z \mid \Pi_i, p_i, r]). && \text{(by (5))} \end{aligned}$$

The random variable Out_i, which represents the output of machine M_i, is fully determined by the transcript Π_i, M_i's input graph assignment (i.e., the random variable P_i), and the random bits. Therefore, by the data processing inequality (see Reference [16]), we can use the bound $\mathbb{H}[Z \mid \Pi_i, p_i, r] \le \mathbb{H}[Z \mid Out_i, p_i, r]$ in the right-hand side of the above inequality to obtain

$$\mathbb{I}[\Pi_*; Z \mid p_*, r] \ge \max_{1 \le i \le k} (\mathbb{H}[Z \mid p_i, r] - \mathbb{H}[Z \mid Out_i, p_i, r]) = \max_{1 \le i \le k} \mathbb{I}[Out_i; Z \mid p_i, r]$$

and the lemma follows. □

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 $\mathbb{I}[Out_i; Z \mid p_i, r] \ge \text{IC} - o(\text{IC})$.*

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

$$\mathbb{I}[Out_i; Z \mid p_i, r] = \mathbb{H}[Z \mid p_i, r] - \mathbb{H}[Z \mid Out_i, p_i, r]. \tag{10}$$

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

$$\begin{aligned} \mathbb{H}[Z \mid p_i, r] &= -\sum_z \Pr[Z = z \mid p_i, r] \log_2 \Pr[Z = z \mid p_i, r] \\ &\ge (\mathbb{H}[Z] - o(\text{IC})) \sum_z \Pr[Z = z \mid p_i, r] && \text{(by (1))} \\ &= \mathbb{H}[Z] - o(\text{IC}), && (11) \end{aligned}$$

where the last equality 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 machine M_i includes its input p_i and the public random string r in its output, we have

$$\mathbb{H}[Z \mid Out_i, p_i, r] = \mathbb{H}[Z \mid Out_i], \tag{12}$$

and thus, we will proceed by proving an upper bound on the latter term. To simplify 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

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

Recalling that the premise of Theorem 2.1 states $\mathbb{H}[Z \mid \mathcal{A}_i(\mathbf{p}, r)] \leq \mathbb{H}[Z]$, we can plug this bound into the second term of the sum on the right-hand side to obtain

$$\mathbb{H}[Z \mid Out_i] \leq \sum_{(\mathbf{p},r)\in Good} \Pr[\mathcal{A}_i(\mathbf{p}, r)]\, \mathbb{H}[Z \mid \mathcal{A}_i(\mathbf{p}, r)] + \mathbb{H}[Z]\left(\sum_{(\mathbf{p},r)\notin Good} \Pr[\mathcal{A}_i(\mathbf{p}, r)]\right). \tag{13}$$

Intuitively speaking, the first sum in Equation (13) 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 Equation (13), where $(\mathbf{p}, r)$ is restricted to the set *Good*.

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

Proof of Claim 1. From the definition of entropy, we obtain

$$\mathbb{H}[Z \mid \mathcal{A}_i(\mathbf{p}, r)] = -\sum_{z} \Pr[Z = z \mid \mathcal{A}_i(\mathbf{p}, r)] \log_2 \Pr[Z = z \mid \mathcal{A}_i(\mathbf{p}, r)]. \tag{14}$$

Since we assume that machine M_i also outputs its initial graph assignment (i.e., p_i) and the public random string r, it holds that

$$\mathbb{H}[Z \mid \mathcal{A}_i(\mathbf{p}, r)] = \mathbb{H}[Z \mid \mathcal{A}_i(\mathbf{p}, r), p_i, r],$$

which allows us to rewrite Equation (14) as

$$\mathbb{H}[Z \mid \mathcal{A}_i(\mathbf{p}, r)] = -\sum_{z} \Pr[Z = z \mid \mathcal{A}_i(\mathbf{p}, r), p_i, r] \cdot \log_2 \Pr[Z = z \mid \mathcal{A}_i(\mathbf{p}, r), p_i, r].$$

Recalling that M_i satisfies Equation (2), we get

$$\mathbb{H}[Z \mid \mathcal{A}_i(\mathbf{p}, r)] \leq (\mathbb{H}[Z] - \mathrm{IC}) \sum_{z} \Pr[Z = z \mid \mathcal{A}_i(\mathbf{p}, r), p_i, r] = \mathbb{H}[Z] - \mathrm{IC},$$

since $\sum_z \Pr[Z = z \mid \mathcal{A}_i(\mathbf{p}, r), p_i, r] = 1$. □

We will now derive an upper bound on the second sum in Equation (13).

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

Proof of Claim 2. Consider the set $(\mathcal{GP} \times \mathcal{RS}) \setminus Good$. According to our model, the input graph and its partitioning among the machines correspond to choosing, uniformly at random, an

element from $\mathcal{GP}$, whereas the random string r is uniformly selected from $\mathcal{RS}$. Since the output of machine M_i is fully determined by $(\mathbf{p}, r)$, we have

$$\sum_{(\mathbf{p},r)\notin Good} \Pr[\mathcal{A}_i(\mathbf{p},r)] = \sum_{(\mathbf{p},r)\notin Good} \Pr[(\mathbf{p},r)] = \Pr[(\mathcal{GP}\times\mathcal{RS})\setminus Good].$$

From the lower bound on the size of *Good* in the theorem premise, we obtain an upper bound such that

$$\sum_{(\mathbf{p},r)\notin Good} \Pr[\mathcal{A}_i(\mathbf{p},r)] = \Pr[(\mathcal{GP}\times\mathcal{RS})\setminus Good] \leq \epsilon + n^{-\Omega(1)},$$

thus proving the claim. □

Plugging the bounds in Claims 1 and 2 into Equation (13), we get

$$\begin{aligned}\mathbb{H}[Z \mid Out_i] &\leq (\mathbb{H}[Z] - \mathrm{IC}) \sum_{(\mathbf{p},r)\in Good} \Pr[\mathcal{A}_i(\mathbf{p},r)] + \mathbb{H}[Z]\left(\epsilon + n^{-\Omega(1)}\right)\\ &\leq (\mathbb{H}[Z] - \mathrm{IC}) + \mathbb{H}[Z]\left(\epsilon + n^{-\Omega(1)}\right).\end{aligned}$$

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 $\epsilon = o(\mathrm{IC}/\mathbb{H}[Z])$ and $\mathrm{IC} \leq \mathbb{H}[Z]$, hence $\epsilon\cdot\mathbb{H}[Z] = o(\mathrm{IC})$. From this and Equation (12), we conclude that

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

Plugging this upper bound and the lower bound of Equation (11) into the right-hand side of Equation (10) 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

$$\begin{aligned}\mathbb{H}[\Pi_* \mid p_*, r] &\geq \mathbb{I}[\Pi_*; Z \mid p_*, r] && \text{(by (6))}\\ &\geq \mathbb{I}[Out_i; Z \mid p_i, r] && \text{(by Lemma 2.2)}\\ &\geq \mathrm{IC} - o(\mathrm{IC}), && (15)\end{aligned}$$

where the last inequality follows from Lemma 2.3. To complete the proof of Theorem 2.1, we will argue that the worst-case running time needs to be large, as otherwise the entropy of machine $M_{\ell(\mathbf{p},r)}$'s transcript Π_* would be less than $\mathrm{IC} - o(\mathrm{IC})$. 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

$$\mathbb{H}[\Pi_* \mid p*, r] \leq \log_2\left(2^{(B+1)(k-1)T}\right) = O(B\,k\,T). \quad (16)$$

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.*

Proof. Since in a synchronous model one can convey information even by *not* sending any bits in a given round, there are at most $2^B + 1 < 2^{B+1}$ distinct possibilities for the communication received over a single link of bandwidth B in any given round. Thus, we can view the communication received over M_i's $k-1$ links as a word ω_1 of length $k-1$, where each character of ω_1 is chosen from

an alphabet of size (at most) 2^{B+1}, resulting in $2^{(B+1)(k-1)}$ possible choices for ω_1. Finally, we view Γ, i.e., the communication received over the T rounds, as a word of length T, where the alphabet size of each character is at most $2^{(B+1)(k-1)}$, yielding $2^{(B+1)(k-1)T}$ many choices in total. □

Recall that the running 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 Equation (15) and Equation (16), it follows that

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

This completes the proof of Theorem 2.1.

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 at least $1 - o(1/k)$. Then, the running time of $\mathcal{A}$ is $\Omega(\frac{n}{B \cdot k^2})$, 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 running 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 $1 \leq i \leq m/4$, 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 $1 \leq i \leq m/4$ 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 $1 \leq i \leq n/4$: 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* $\text{PageRank}(v_i) \geq \frac{(3-3\epsilon+\epsilon^2)\epsilon}{n}$. *For any $\epsilon < 1$, there is a constant factor separation between the two cases.*

Proof. We will determine an estimate of $\text{PageRank}(v_i)$ using the distributed random walk approach described at the beginning of Section 3.1, in which the expected number of random walk

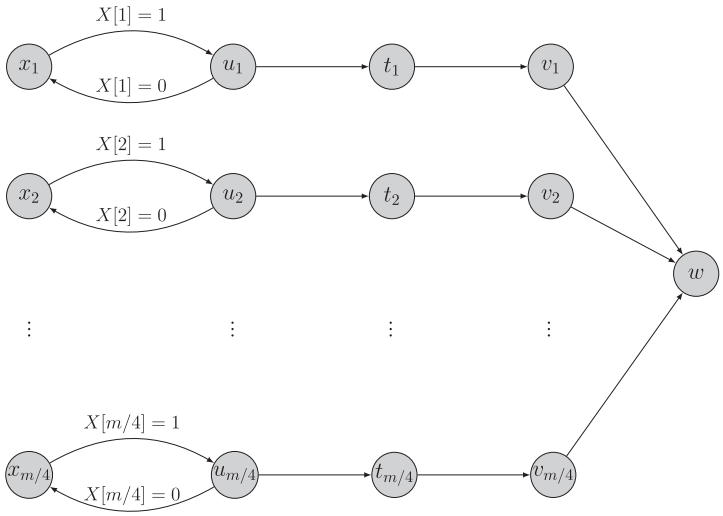

Fig. 1. The graph H used to derive a lower bound on the round complexity of PageRank computations.

tokens addressed to one node, multiplied by $\epsilon/cn \log n$, gives a high-probability estimate of the PageRank value of the node. If ψ_{v_i} denotes the number of random walk tokens addressed to node v_i, then

$$\mathbb{E}[\psi_{v_i} | b_i = 0] = c \log n \left(1 + (1-\epsilon) + \frac{(1-\epsilon)^2}{2}\right)$$

and

$$\mathbb{E}[\psi_{v_i} | b_i = 1] = c \log n \left(1 + (1-\epsilon) + (1-\epsilon)^2 + (1-\epsilon)^3\right).$$

Therefore,

$$\mathsf{PageRank}(v_i) = \frac{(2.5 - 2\epsilon + \epsilon^2/2)\epsilon}{n}$$

if $b_i = 0$, and

$$\mathsf{PageRank}(v_i) \geq \frac{(3 - 3\epsilon + \epsilon^2)\epsilon}{n}$$

if $b_i = 1$. □

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 $\{S \subset [1, \text{poly}(n)] : |S| = n\}$. Assigning to each vertex of H a unique integer drawn uniformly and without repetitions from ID 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 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 make assumptions neither on which machine outputs the PageRank of a specific vertex v (which therefore could be a machine that has no initial knowledge about v), 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.

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

Proof. Fix one machine M_i. If a vertex is assigned to M_i, then machine M_i knows its incident edges and the IDs of their endpoints. Therefore, M_i can discover a weakly connected path (between X and V) in one of the following ways: (1) M_i obtains $x_j \in X$ and $t_j \in T$; (2) M_i obtains $u_j \in U$ and $v_j \in V$. The argument is similar in both cases, and hence, we focus on (1) for the rest of this proof. By the random vertex partition process, the probability that x_j and t_j both are assigned to machine M_i is $\frac{1}{k^2}$. Since all vertices are assigned independently at random, a standard Chernoff bound shows that with high probability $O(n\log n/k^2)$ matching vertex pairs (x_j, t_j) are assigned to machine M_i. Applying the union bound over the k machines completes the proof. □

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) $|Good| \geq (1 - o(1/k) - n^{-\Omega(1)})|\mathcal{GP} \times \mathcal{RS}|$.*

Proof. Part (A) follows directly from the definition of set *Good*. For (B), note that $\mathcal{A}$ succeeds with probability at least $1 - o(1/k)$. Moreover, the random vertex partitioning ensures that each machine receives $\tilde{\Theta}(n\log(n)/k)$ vertices with probability at least $1 - n^{-4}$. Hence, the above is true for at least a $(1 - o(1/k) - n^{-4})$-fraction of the possible graph partition and random string pairs in $\mathcal{GP} \times \mathcal{RS}$. □

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 $\{(b_1, v_1), \ldots, (b_{m/4}, v_{m/4})\}$, 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,*

$$Pr[Z = z \mid p_i, r] \leq 2^{-\left(m/4 - O(n\log(n)/k^2)\right)}.$$

Proof. Consider a $(\mathbf{p}, r) \in Good$ where $\mathbf{p} = (p_1, \ldots, p_i, \ldots, p_k)$. By Lemma 2.8, part (A), algorithm $\mathcal{A}$ correctly computes the PageRank, and some machine (without loss of generality) M_i outputs at least $\Omega(n/k)$ PageRank values.

By Lemma 2.6, we know that algorithm $\mathcal{A}$ can only correctly output $\text{PageRank}(v_j)$ at machine M_i if M_i knows the direction of the edge between u_j and x_j (from Lemma 2.6, since the direction

of the corresponding edge can be derived from the PageRank value). This means that if machine M_i outputs the PageRank for v_j as a pair (π_j, v_j), then it can reconstruct the pair (b_j, v_j), for any $1 \leq j \leq m/4$.

Since $(\mathbf{p}, r) \in \mathit{Good}$, it follows by Lemma 2.7 that each machine M_i learns at most $\eta = O(n \log(n)/k^2)$ output entries of V for free by inspecting its assigned input. In addition to these η entries, M_i might know partial information about the remaining $\Omega(n) - \eta$ pairs.

It follows that, for each of the other weakly connected paths that are not concerned with its η already known PageRank values, M_i either has initial knowledge of the index ℓ of the respective vertex $v_\ell \in V_i$ or it knows the edge direction b_ℓ between x_ℓ and u_ℓ, but not both. Notice that knowledge of the vertex ID of v_ℓ reveals no additional information about the index ℓ, since we choose vertex IDs uniformly at random. We refer to these paths as being *partially known to* M_i.

It follows that, for each index j for which the path is partially known to M_i, there are two possibilities $(0, v_j)$ and $(1, v_j)$, each of which is equally likely, according to the input distribution.

Therefore, taking into account the initial input assignment, we still have at least $2^{m/4 - O(n \log(n)/k^2)}$ possible choices for z, i.e., the output of M_i concerning vertices in V, each of which is equally likely without conditioning on further knowledge. Thus,

$$\Pr[Z = z \mid p_i, r] \leq 2^{-(m/4 - O(n \log(n)/k^2))},$$

completing the proof of the lemma. □

Lemma 2.10. *For each $(\mathbf{p}, r) \in \mathit{Good}$, 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 $Pr[Z = z \mid \mathcal{A}_i(\mathbf{p}, r), p_i, r] \geq 1/2^{\frac{m}{4} - \frac{m}{4k}}$.*

Proof. By Lemma 2.8, we know that there is a machine M_i that outputs at least $m/4k$ PageRank values of vertices in V. Let λ be the total number of pairs (b_j, v_j), where b_j is the direction of the edge (x_j, u_j) in the weakly connected path (x_j, u_j, t_j, v_j) (cf. Lemma 2.9) that remain unknown to machine M_i conditioned on its input p_i, random string r, and its output o_i.

Observing that the size of its output o_i is at least $m/4k$, and from the fact that we can recover the pair (b_j, v_j) if M_i outputs the PageRank of v_j (see proof of Lemma 2.9), it follows that $\lambda \leq m/4 - m/4k$, and thus there are $2^{\frac{m}{4} - \frac{m}{4k}}$ distinct choices for z. The probability bound is minimized if each of the remaining possible choices of z are equally likely. This implies that $\Pr[Z \mid o_i, p_i, r] \geq 1/2^{\frac{m}{4} - \frac{m}{4k}}$, as desired. □

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 running 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 $\Omega(\frac{n^2}{B \cdot k^{5/3}})$, 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[7] 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 $\mathcal{G}$.

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 at least t triangles, for some fixed $t = \Theta(\binom{n}{3})$.

Lemma 2.12 (Good Inputs). *(A) For every $(\mathbf{p}, r) \in$ Good, at least one machine outputs at least t/k triangles when executing algorithm $\mathcal{A}$ with $(\mathbf{p}, r)$, and (B) $|Good| \geq (1 - \epsilon')|\mathcal{GP} \times \mathcal{RS}|$, where $\epsilon' = \epsilon + n^{-\Omega(1)}$.*

Proof. Part (A) is immediate from the definition of *Good*. For (B), note that $\mathcal{A}$ succeeds with probability at least $1 - \epsilon$ and the random vertex partitioning guarantees a balanced partition with probability at least $1 - n^{-4}$. From Reference [31, Equation 4.10], we know that the number of triangles in a input graph G sampled from $G_{n,1/2}$ is $\Theta(\binom{n}{3})$ with probability at least $1 - e^{-\Omega(1)}$, and hence the set *Good* contains all except at most a $(1-\epsilon-n^{-3})$-fraction of the graphs in $\mathcal{GP}\times\mathcal{RS}$. □

Lemma 2.13. *Let random variable Z denote the characteristic vector of the edges of the sampled input graph G. For every $(\mathbf{p}, r) \in$ Good where $\mathbf{p} = (p_1, \ldots, p_k)$ and every characteristic edge vector z, it holds that $Pr[Z = z \mid p_i, r] \leq 1/2^{\binom{n}{2} - O(n^2 \log(n)/k)}$, for every $i \in [1, k]$.*

Proof. For any $(\mathbf{p}, r) \in$ *Good*, each machine has initial knowledge of $O(n^2 \log n/k)$ edges. Consider any machine M_i. Since the random vertex partitioning and the sampling of the input graph are independent, there are at least $2^{\binom{n}{2} - O(n^2 \log(n)/k)}$ choices for the remaining edges, all of which are equally likely according to the random graph model, giving the claim. □

Lemma 2.14. *Let $(\mathbf{p}, r) \in$ 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 edge vector z that has non-zero probability conditioned on $\mathcal{A}_i(\mathbf{p}, r)$, p_i, r,*

$$Pr[Z = z \mid \mathcal{A}_i(\mathbf{p}, r), p_i, r] \geq 1/2^{\binom{n}{2} - O(n^2 \log(n)/k) - \Omega((t/k)^{2/3})}.$$

Proof. By assumption $(\mathbf{p}, r) \in$ *Good*, which means that the machines output all $t = \Theta(\binom{n}{3})$ triangles. Thus, there is some machine M_i that outputs at least t/k triangles. We will bound from below the number of edges known by machine M_i conditioned on its output and its input assignment.

[7] 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.

Initially, M_i discovers $t_3 = t_3(P_i)$ "local" triangles (for which it knows all threes edges) by inspecting its assigned portion of the input graph given by P_i. Since we are restricting the inputs to be in $Good_i$, we know that the edges known to M_i are bounded by $O(n^2 \log n/k)$ and hence the number of triangles formed using these edges is

$$t_3 = O((n^2 \log n/k)^{3/2}) = O(n^3 \log^{3/2}(n)/k^{3/2}).$$

We call a triangle λ *undetermined w.r.t.* M_i if M_i is unaware of at least one edge of λ initially. Formally, λ is undetermined if there are two input graphs G and G' where λ exists in G but not in G' and both graphs are compatible with the input p_i assigned to machine M_i.

By the above, we have at least $(t/k) - t_3$ undetermined triangles that are output by M_i. From Equation (10) in Reference [62], we know that the number of distinct edges necessary for representing ℓ triangles is $\Omega(\ell^{2/3})$. This means that at least $((t/k) - t_3)^{2/3}$ edges are required for representing the undetermined triangles of M_i. We can divide the undetermined triangles into two sets: One set T_1 contains triangles that have a vertex allocated to M_i, and the other set T_2 contains triangles that have no vertex allocated to M_i. Set T_1 contributes $|T_1|/(n \log n/k)$ unknown edges, since the number of vertices allocated to this machine is $O(n \log n/k)$, whereas T_2 contributes $1/3 \cdot (|T_2|)^{2/3}$ unknown edges. These two sets of unknown edges might overlap, hence, we need to consider the maximum over them, which can be shown to be $\Omega(((t/k) - t_3)^{2/3})$. Hence, it is possible to recover $\Omega(((t/k) - t_3)^{2/3})$ edges from M_i's output that were unknown to M_i initially. Let η denote the number of unknown edges of Z when M_i outputs its solution. Taking into account the initially known edges, we have

$$\eta \leq \binom{n}{2} - \Omega\left(\frac{t}{k} - t_3\right)^{2/3} - O\left(\frac{n^2 \log n}{k}\right) = \binom{n}{2} - O\left(\frac{n^2 \log n}{k}\right) - \Omega\left(\frac{t}{k}\right)^{2/3} \tag{17}$$

possible edges that are unknown to M_i, since $t_3 = o(t/k)$. Since we have sampled the edges of the input graph following the $G_{n,1/2}$ random graph model, it follows that, for any z that has nonzero probability given M_i's output and initial assignment, $\Pr[Z = z \mid o_i, p_i, r] = 2^{-\eta}$. The lemma follows by applying Equation (17). □

Proof of Theorem 2.11. We are now ready to instantiate Theorem 2.1 where Z is the characteristic vector of edges as defined above. Note that Lemma 2.13 and Lemma 2.14 satisfy Premises (1) and (2). Note that $\Omega(t/k)^{2/3} = \Omega(n^2/k^{2/3})$. Setting IC $= \Theta(n^2/k^{2/3})$ 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 Reference [20], this implies the following:

Corollary 2.15. *The round complexity of enumerating all triangles in the congested clique of n nodes with high probability of success is* $\Omega(\frac{n^{1/3}}{B})$*, assuming a link bandwidth of B bits. This bound is tight up to logarithmic factors.*

Message lower bounds. We point out that it is possible to extend Theorem 2.1 to yield new message lower bounds for algorithms that attain an efficient time complexity. We outline the high-level argument for triangle enumeration. Consider an algorithm matching the time bound of Theorem 2.11, i.e., $T = \tilde{O}(n^2/k^{5/3})$ assuming a bandwidth of $B = O(\log n)$ bits. In the k-machine model, in T rounds each machine can receive at most $\mu = \tilde{O}(n^2/k^{2/3})$ bits in total. Lemma 2.13 tells us that every machine has very little initial knowledge about the t triangles in the graph given its initial graph assignment when considering inputs chosen from *Good*. However,

inspecting the proof of Lemma 2.14, we can observe that a machine M_j who outputs t_j triangles needs to receive $\tilde{\Omega}(t_j^{2/3})$ bits of information. If we restrict the algorithm to terminate within T rounds, then this means that each machine can output at most $O(n^3/k)$ triangles, as this requires $\mu = O((n^3/k)^{2/3})$ bits of information. This implies that the output per machine must be roughly balanced and every machine needs to receive $\Omega(\mu)$ bits of information, yielding a message complexity of $\tilde{\Omega}(k \cdot n^2/k^{2/3}) = \tilde{\Omega}(n^2k^{1/3})$. In particular, this rules out algorithms that aggregate all input information at a single machine (which would only require $O(m)$ messages in total). From the above, we have the following:

Corollary 2.16. *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 Approximation

In this section, we present a simple distributed algorithm to approximate 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 [34].

We first recall the distributed random-walk-based Monte Carlo algorithm for computing PageRank, for a given reset probability ϵ, as described in Reference [19]. 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 Reference [19], which are then forwarded according to the following process: When a node u receives some random walk token ρ, with probability ϵ it terminates the token, and with probability $1 - \epsilon$ it forwards the token to a neighbor node chosen uniformly at random. Each machine 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 [19]. 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.

Note that a straightforward implementation of the above random walk-based algorithm might yield a suboptimal running time in the k-machine model. (In fact, applying the Conversion Theorem of Reference [34] to implement the above algorithm gives only $\tilde{O}(n/k)$ time.) The main issue is that some machine might receive too many random walks destined for the nodes in that machine. For example, during some step of the random walk it might happen that n different walks are destined to different nodes in the same machine, causing $\Omega(n)$ congestion at some machine leading to a $\Omega(n/k)$ bound. For example, in a star-like topology, the center vertex c that resides at some machine M_1 might need to receive n random walks from its neighbors, hence causing a round complexity of $\tilde{\Omega}(n/k)$. In the above example, since there is only one high-degree vertex, we can get around this problem by sending only the counts. However, the situation is less clear if $\Omega(n)$ tokens are destined for different nodes in the same machine.

To avoid the above pitfalls, we describe an approach tailored to 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 ensure 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 is *light in iteration r* if, conceptually, the machine that hosts u considers less than k tokens to be held at u. Otherwise, we say that u is *heavy in iteration r*.

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 that we use as the initialization value of tokens. Then, we mimic the (parallel) random walk steps of Reference [19] by performing $\Theta(\log(n)/\epsilon)$ iterations where, in each iteration, each machine M first considers the tokens stored for its light 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 light vertex $w \neq u$ at M. Then, M sends a message $\langle \alpha[v], \text{dest:}v \rangle$ for each v where $\alpha[v]$ is nonzero, which is subsequently delivered to the destination machine using random routing (cf. Lemma 3.3). 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 according to the degree distribution of u (see Line 23 in Algorithm 1) 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.

Proposition 3.1. *Algorithm 1 correctly computes the* PageRank *with high probability.*

Proof. In Reference [19] it is shown that the random walk process, where each token is either terminated with probability ϵ or forwarded with probability $1 - \epsilon$ to a neighbor chosen uniformly at random, approximates the PageRank of the graph. Thus, it is sufficient to show that Algorithm 1 adheres to this random walk process.

Consider a node u and suppose that u holds ℓ tokens. If $\ell < k$, then according to Lines 8–16, we increment the corresponding entry of array $\alpha[v]$ for some uniformly at random chosen neighbor v of u and send a message $\langle c_v, dest : v \rangle$ to the machine M' hosting v. Upon receiving the message, M' increases its token count of v, as required.

Now, suppose that $\ell \geq k$ and consider an arbitrary neighbor v of u, hosted on machine M' and assume that M' hosts $n_u \geq 1$ neighbors of u in total. For any token of u, it follows from Line 23 that we choose machine M' with probability n_u/d_u, where d_u is the degree of u in the graph.

The algorithm then sends a message of type $\langle c_u, src : u \rangle$ to machine M', where c_u is the number of tokens of u for which M' was sampled as the destination machine. Upon processing this message in Lines 31–36, M' delivers each token to its locally hosted neighbors of u uniformly at random, and hence a specific neighbor v receives a token with probability $1/n_u$.

Combining these observations, we conclude that v receives a token with probability $n_u/d_u \cdot 1/n_u = 1/d_u$, conditioned on the token not having been terminated in Line 6 with probability ϵ, which corresponds to the random walk process of Reference [19]. □

Lemma 3.2. *Every machine M_i sends at most $O(n \log(n)/k)$ messages in any iteration r with high probability.*

ALGORITHM 1: Approximating the PageRank with reset probability $\epsilon > 0$. Code for machine M_i.

Let V_i denote the vertices hosted by machine M_i
Initialize 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 light 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$[u] \geq k$ **do** ▷ Process the heavy 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$[w] + 1$
            $c_v \leftarrow c_v - 1$

Proof. First, we consider messages that M_i needs to send on behalf of its hosted light vertices. We classify the light vertices into *send bins* $S_0, S_1, \ldots, S_{\lceil \log k \rceil - 1}$, according to the number of distinct messages that they require to be sent and, for each j, $0 \leq j \leq \lceil \log_2 k \rceil - 1$, we define the bin

$$S_j = \left\{ v \in V(G) \;\middle|\; \frac{k}{2^{j+1}} \leq \text{tokens}[v] < \frac{k}{2^j} \right\}. \tag{18}$$

By definition, the total number of messages generated for any light vertex in iteration r is at most $k-1$, and hence every light v is in some bin S_j.

Since $\Theta(\log n)$ tokens are generated initially for each vertex, we have $\Theta(n \log n)$ tokens in total, which implies that $|S_j| \leq \frac{2^{j+1} n \log n}{k}$, for all j. By the random vertex partitioning, we know that a machine M_i receives at most $O(|S_j| \log(n)/k)$ vertices from S_j with probability $\geq 1 - n^{-4}$; we denote this vertex set by $S_{i,j}$. Taking a union bound over the iterations of the algorithms (assuming a constant reset probability ϵ), the $O(\log_2 k)$ distinct bins, and over the k machines, it follows that

$$\forall M_i \; \forall j \in \{0, \ldots, \lceil \log_2 k \rceil - 1\} \colon |S_{i,j}| = O\left(\frac{2^{j+1} n \log n}{k^2}\right), \tag{19}$$

with probability $\geq 1 - n^{-2}$. According to Equation (18), each vertex in bin S_j holds less than $k/2^j$ tokens, and thus by Equation (19) the total number of messages produced by vertices in S_j that are located on machine M_i is

$$O\left(|S_{i,j}| \cdot \frac{k}{2^j}\right) = O\left(\frac{2^{j+1} n \log n}{k^2} \frac{k}{2^j}\right) = O\left(\frac{n \log n}{k}\right).$$

Since we have $\Theta(\log k)$ bins, the total number of messages generated by machine M_i for its light vertices is $O(n \log(n)/k) \cdot \Theta(\log k) = \tilde{O}(n/k)$ with high probability.

Now, consider the heavy vertices at M_i. By definition, each heavy vertex has at least k tokens, and hence there are at most $O(n \log(n)/k)$ heavy vertices at any point of the algorithm. Therefore, the random vertex partitioning implies that each machine will hold most $O(n \log(n)/k^2)$ many heavy vertices w.h.p. For processing the tokens of a heavy vertex u, we recall from Algorithm 1 that we need to send at most one message to each machine that holds a neighbor of u. This means that all messages generated for u can be sent and delivered in one round, and hence by taking a union bound over all the machines, it follows that each machine can send all tokens for its heavy vertices in $O(n \log(n)/k^2)$ rounds.

Finally, the lemma follows by taking a union bound over the $O(\log(n)/\epsilon)$ iterations of the algorithm. □

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 3.3. *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 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.*

Proof. We shall prove the statement for the case in which each machine is the source of $O(x)$ messages. The other case and its analysis are symmetric.

Since destinations of messages are chosen randomly, we choose to route each message to its (random) destination machine through the link that directly connects the source to the destination machine (which always exists because the network is complete). By a classic balls-into-bins result, each of the $k-1$ links of each machine is responsible for carrying $O((x \log x)/k)$ messages w.h.p., and the result follows. □

Lemma 3.4. *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.*

Proof. We first consider the messages generated due to a heavy vertex u. Recall from Algorithm 1 that each machine directly sends the messages that it generated for u to the destination machines, which requires just one round. As we have argued in Lemma 3.2, there are at most $O(n\log(n)/k^2)$ many heavy vertices per machine w.h.p., and hence all of their messages can be delivered within $O(n\log(n)/k^2)$ rounds.

In the remainder of the proof, we focus on messages generated while processing light vertices. To this end, we argue that each machine needs to receive at most $\tilde{O}(n/k)$ messages that were generated due to light vertices in Line 16, which according to the random routing result, can be delivered in $\tilde{O}(n/k^2)$ rounds. We proceed similarly to the analysis in Lemma 3.2. That is, we define *receive bins* $R_0, R_1, \ldots, R_{\lceil \log k \rceil - 1}$, where

$$R_j = \left\{ v \in V(G) \mid \frac{k}{2^{j+1}} \leq \lambda_v \leq \frac{k}{2^j} \right\}$$

and λ_v is the random variable that counts the number of tokens generated for light vertices that are received by v in iteration r. Consider any $v \in V(G)$ located at some machine M. The crucial point is that each v must be in exactly one of these bins, since Line 16 ensures that machine M receives at most one message of type $\langle \alpha[v], \text{dest}{:}v \rangle$ that is addressed to v from each distinct machine M'.

Similarly as in Lemma 3.2, it follows by the properties of the random vertex partitioning that each machine holds $\tilde{O}(|R_j|/k)$ vertices from R_j with high probability, and hence the total number of messages that each machine needs to receive (over all receive bins) is $\tilde{O}(n/k)$. Thus, by Lemma 3.3, all of these messages can be delivered in $\tilde{O}(n/k^2)$ rounds. Finally, it is shown in Reference [19] that all tokens are terminated in $O(\log(n)/\epsilon)$ steps and thus, assuming that $\epsilon > 0$ is a small constant, the claim follows by a union bound over the iterations of the algorithm. □

From Lemma 3.4, 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 3.5. *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 [20], with some crucial differences explained next. The key idea, which in its generality can be traced back to Reference [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 $h : V \to C$ 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.

Our algorithm differs from the one in Reference [20] in the way the $k^{1/3}$ subsets of vertices are constructed, in the use of proxy computation and in the routing of messages, which in our algorithm is randomized and hence requires a more involved analysis, allowing for a better time complexity for graphs where the number of edges m is $o(n^2)$.

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 [63].[8]

Proposition 3.6 ([63, 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 $t \geq 1/3\eta$. Let $e(G[R])$ denote the number of edges in the subgraph induced by R. Then,*

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

for some $c > 0$.[9]

We are now ready to analyze the algorithm.

Theorem 3.7. *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.*

Proof. Since there are $(k^{1/3})^3 = k$ possible triplets of non-intersecting subsets of $n/k^{1/3}$ nodes, all possible triangles are examined by the algorithm, and this proves its correctness.

We now argue that the algorithm terminates in $\tilde{O}(m/k^{5/3} + n/k^{4/3})$ rounds w.h.p. As part of the argument used to prove Lemma 4.1 of Reference [34] it is shown that every machine initially stores $\tilde{O}(m/k + \Delta)$ edges, where Δ is the maximum degree of the graph. If we apply Lemma 3.3 directly, then the communication phase that assigns the edges to their random proxies takes $\tilde{O}(m/k^2 + \Delta/k)$ rounds w.h.p. We now argue that the proxy assignment rule allows us to show an $\tilde{O}(m/k^{5/3})$ bound for this phase for every non-sparse graph.

Clearly, by the random proxy assignment, each machine receives only $\tilde{O}(m/k)$ messages. We next argue that each machine is responsible for designating only $\tilde{O}(m/k)$ edges w.h.p. Then, by Lemma 3.3, the time to send all the designation messages is $\tilde{O}(m/k^2)$ rounds.

[8] Observe that one cannot simply apply a Chernoff bound, since edges are not chosen independently; also, mimicking the argument for the proof of Lemma 4.1 in Reference [34] would give a bound of the form $\tilde{O}(m/k^{1/3})$, which is weaker, since we would be overcounting edges (as we would be counting also those edges with just one endpoint in the given machine).

[9] A careful inspection of the argument used by Rödl and Ruciński to establish this result reveals that the additional condition $t \geq 1/3\eta$, missing from their statement, is necessary for the result to hold. In fact, as stated, their result is implicitly assuming that both n and t grow to infinity [64].

For the sake of the analysis, we partition the non-isolated nodes of the input graph into $\log n$ sets, based on their degree: The ith set contains all the nodes whose degree is in $[\Delta/2^i, \Delta/2^{i+1})$, $0 \leq i \leq \log n - 1$. We now focus on the number of messages sent by some machine M. By a standard Chernoff bound, a node v_i with degree d_i in the ith set has $\tilde{O}(d_i/k)$ neighbors in M w.h.p. If n_i is number of nodes in the ith set, then the total number of neighbors (and hence messages) that M will send with respect to nodes in this set is $\tilde{O}(n_i d_i/k)$ w.h.p. Summing over all the $\log n$ sets, we have that the total number of messages sent by M is $\sum_{i=0}^{\log n-1} \tilde{O}(n_i d_i/k) = \tilde{O}(m/k)$ w.h.p. (via the union bound). Applying the union bound over all the machines, we have that the same bound holds for every machine.

The above argument does not take into account the messages sent by a machine initially to request the designation of an edge. A machine needs one round (to broadcast to all the other machines) to request such a designation. If some machine M sends $f \geq k \operatorname{polylog} n$ requests, then M must have f nodes with degree at least $2k \log n$. By the RVP, this implies that with high probability the total number of nodes with degree at least $2k \log n$ is at least $\Omega(fk)$. Hence, the number of edges in the graph is $m = \tilde{\Omega}(fk^2)$. Therefore, the number of rounds needed for broadcast, $\tilde{O}(f)$, is subsumed by $\tilde{O}(m/k^{5/3})$.

Next, we analyze the re-routing of each edge e from its edge proxy to all the machines that are assigned a copy of both of the endpoints of e. Observe that any two nodes, and therefore any edge, can be held by at most $k^{1/3}$ different machines: Consider an edge (a, b), and pick one machine M that has to receive it, because, among its three subsets of nodes, one (call it A) contains a and one (call it B) contains b. Edge (a, b) can be assigned only to those machines that contain *both* subsets A and B, and there are only $k^{1/3} - 1$ such machines in addition to M. Hence, re-routing the edges entails $mk^{1/3}$ messages to be traveling across the network.[10] We first bound the number of edges received by each machine. Fix one machine M. We shall apply Proposition 3.6 with $t = dn \log n/k^{1/3}$ for some positive constant d. We have two cases. If $m \geq nk^{1/3}/6d \log n$, then $m \geq n^2/6t$, which in turn implies $2m/n^2 \geq 1/3t$, and thus, we can apply Proposition 3.6 with $\eta = 2m/n^2$ obtaining, for machine M,

$$\Pr\left[e(G[R]) > 3\frac{2m}{n^2}\left(\frac{dn \log n}{k^{1/3}}\right)^2\right] < t \cdot e^{-cdn \log n/k^{1/3}},$$

that is, since $k \leq n$,

$$\Pr\left[e(G[R]) \leq \frac{6d^2 m \log^2 n}{k^{2/3}}\right] > 1 - e^{-\Omega(\log n)}.$$

Hence, we can apply Lemma 3.3 with $x = \tilde{O}(m/k^{2/3})$, which yields a round complexity of $\tilde{O}(m/k^{5/3})$ w.h.p. Now observe that each proxy has to send $\tilde{O}(m/k^{2/3})$ edges. We can apply Lemma 3.3 with $x = \tilde{O}(m/k^{2/3})$, which implies that the number of rounds needed for the proxies to send their edges is $\tilde{O}(m/k^{5/3})$ w.h.p., completing the analysis for the case $m \geq nk^{1/3}/6d \log n$.

However, if $m < nk^{1/3}/6d \log n$, then we shall apply Proposition 3.6 with $\eta = 1/3t = k^{1/3}/3dn \log n$, obtaining

$$\Pr\left[e(G[R]) > 3\frac{k^{1/3}}{3dn \log n}\left(\frac{dn \log n}{k^{1/3}}\right)^2\right] < t \cdot e^{-cdn \log n/k^{1/3}},$$

[10] Notice that each node is replicated $k^{2/3}$ times in the system, and therefore each edge is replicated $k^{4/3}$ times; however, we only need to re-route copies of edges that are *internal* to the triplets, and therefore copies of edges that have one endpoint in one triplet and the other endpoint in a different triplet need not be communicated. Hence, the total number of edges to be communicated is $mk^{1/3}$ and not $mk^{2/3}$.

that is, since $k \leq n$,

$$\Pr\left[e(G[R]) \leq \frac{dn \log n}{k^{1/3}}\right] > 1 - e^{-\Omega(\log n)}.$$

As in the previous case, we apply Lemma 3.3, now with $x = \tilde{O}(n/k^{1/3})$. The theorem follows. □

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, and clustering.

REFERENCES

[1] Giraph. http://giraph.apache.org/.

[2] Foto N. Afrati and Jeffrey D. Ullman. 2011. Optimizing multiway joins in a Map-Reduce environment. *IEEE Trans. Knowl. Data Eng.* 23, 9 (2011), 1282–1298.

[3] K. Avrachenkov, N. Litvak, D. Nemirovsky, and N. Osipova. 2007. Monte Carlo methods in PageRank computation: When one iteration is sufficient. *SIAM J. Number. Anal.* 45, 2 (2007), 890–904.

[4] Bahman Bahmani, Kaushik Chakrabarti, and Dong Xin. 2011. Fast personalized PageRank on MapReduce. In *Proceedings of the ACM SIGMOD Conference*. 973–984.

[5] B. Bahmani, A. Chowdhury, and A. Goel. 2010. Fast incremental and personalized PageRank. *PVLDB* 4 (2010), 173–184.

[6] Sayan Bandyapadhyay, Tanmay Inamdar, Shreyas Pai, and Sriram V. Pemmaraju. 2018. Near-optimal clustering in the k-machine model. In *Proceedings of the 19th International Conference on Distributed Computing and Networking (ICDCN'18)*. 15:1–15:10.

[7] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, and D. Sivakumar. 2004. An information statistics approach to data stream and communication complexity. *J. Comput. Syst. Sci.* 68, 4 (2004), 702–732.

[8] P. Berkhin. 2005. A survey on PageRank computing. *Internet Math.* 2, 1 (2005), 73–120.

[9] Jonathan W. Berry, Luke A. Fostvedt, Daniel J. Nordman, Cynthia A. Phillips, C. Seshadhri, and Alyson G. Wilson. 2015. Why do simple algorithms for triangle enumeration work in the real world? *Internet Math.* 11, 6 (2015), 555–571.

[10] Jonathan W. Berry, Bruce Hendrickson, Randall A. LaViolette, and Cynthia A. Phillips. 2011. Tolerating the community detection resolution limit with edge weighting. *Phys. Rev. E* 83, 5 (2011).

[11] Sergey Brin and Lawrence Page. 1998. The anatomy of a large-scale hypertextual Web search engine. In *Proceedings of the 7th International World-Wide Web Conference (WWW'98)*. 107–117.

[12] Keren Censor-Hillel, Petteri Kaski, Janne H. Korhonen, Christoph Lenzen, Ami Paz, and Jukka Suomela. 2015. Algebraic methods in the congested clique. In *Proceedings of the 34th ACM Symposium on Principles of Distributed Computing (PODC'15)*. 143–152.

[13] Avery Ching, Sergey Edunov, Maja Kabiljo, Dionysios Logothetis, and Sambavi Muthukrishnan. 2015. One trillion edges: Graph processing at Facebook-scale. *PVLDB* 8, 12 (2015), 1804–1815.

[14] Shumo Chu and James Cheng. 2012. Triangle listing in massive networks. *ACM Trans. Knowl. Discov. Data* 6, 4 (2012), 17.

[15] Fan Chung and Olivia Simpson. 2015. 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'15)*. 77–189.

[16] Thomas M. Cover and Joy A. Thomas. 2006. *Elements of Information Theory*. Wiley-Interscience.

[17] Atish Das Sarma, Sreenivas Gollapudi, and Rina Panigrahy. 2011. Estimating PageRank on graph streams. *J. ACM* 58, 3 (2011), 13.

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

[19] Atish Das Sarma, Anisur Rahaman Molla, Gopal Pandurangan, and Eli Upfal. 2015. Fast distributed PageRank computation. *Theor. Comput. Sci.* 561 (2015), 113–121.

[20] Danny Dolev, Christoph Lenzen, and Shir Peled. 2012. "Tri, tri again": Finding triangles and small subgraphs in a distributed setting. In *Proceedings of the 26th International Symposium on Distributed Computing (DISC'12)*. 195–209.

[21] Andrew Drucker, Fabian Kuhn, and Rotem Oshman. 2014. On the power of the congested clique model. In *Proceedings of the 33rd ACM Symposium on Principles of Distributed Computing (PODC'14)*. 367–376.

[22] Michael Elkin, Hartmut Klauck, Danupon Nanongkai, and Gopal Pandurangan. 2014. Can quantum communication speed up distributed computation? In *Proceedings of the 33rd ACM Symposium on Principles of Distributed Computing (PODC'14)*. 166–175.

[23] Brooke Foucault Welles, Anne Van Devender, and Noshir Contractor. 2010. Is a friend a friend?: Investigating the structure of friendship networks in virtual worlds. In *Proceedings of the 28th International Conference on Human Factors in Computing Systems*. 4027–4032.

[24] Ioannis Fudos and Christoph M. Hoffmann. 1997. A graph-constructive approach to solving systems of geometric constraints. *ACM Trans. Graph.* 16, 2 (1997), 179–216.

[25] Mohsen Ghaffari and Merav Parter. 2016. MST in log-star rounds of congested clique. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC'16)*. 19–28.

[26] Seth Gilbert and Lawrence Li. 2020. How fast can you update your MST? (Dynamic algorithms for cluster computing). *CoRR* abs/2002.06762 (2020).

[27] Tao Guo, Xin Cao, Gao Cong, Jiaheng Lu, and Xuemin Lin. 2017. Distributed algorithms on exact personalized PageRank. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD'17)*. 479–494.

[28] James W. Hegeman, Gopal Pandurangan, Sriram V. Pemmaraju, Vivek B. Sardeshmukh, and Michele Scquizzato. 2015. Toward optimal bounds in the congested clique: Graph connectivity and MST. In *Proceedings of the 34th ACM Symposium on Principles of Distributed Computing (PODC'15)*. 91–100.

[29] Tanmay Inamdar, Shreyas Pai, and Sriram V. Pemmaraju. 2018. Large-scale distributed algorithms for facility location with outliers. In *Proceedings of the 22nd International Conference on Principles of Distributed Systems (OPODIS'18)*. 5:1–5:16.

[30] Taisuke Izumi and François Le Gall. 2017. Triangle finding and listing in CONGEST networks. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC'17)*. 381–389.

[31] Svante Janson. 2004. Large deviations for sums of partly dependent random variables. *Rand. Struct. Algor.* 24, 3 (2004), 234–248.

[32] Tomasz Jurdzinski and Krzysztof Nowicki. 2018. MST in $O(1)$ rounds of congested clique. In *Proceedings of the 29th ACM-SIAM Symposium on Discrete Algorithms (SODA'18)*. 2620–2632.

[33] Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. 2010. A model of computation for MapReduce. In *Proceedings of the 21st ACM-SIAM Symposium on Discrete Algorithms (SODA'10)*. 938–948.

[34] Hartmut Klauck, Danupon Nanongkai, Gopal Pandurangan, and Peter Robinson. 2015. Distributed computation of large-scale graph problems. In *Proceedings of the 26th ACM-SIAM Symposium on Discrete Algorithms (SODA'15)*. 391–410.

[35] Christian Konrad, Sriram Pemmaraju, Talal Riaz, and Peter Robinson. 2019. The complexity of symmetry breaking in massive graphs. In *Proceedings of the 33rd International Symposium on Distributed Computing (DISC'19)*. 26:1–26:18.

[36] Paraschos Koutris, Paul Beame, and Dan Suciu. 2016. Worst-case optimal algorithms for parallel query processing. In *Proceedings of the 19th International Conference on Database Theory (ICDT'16)*. 8:1–8:18.

[37] Jakub Lacki, Slobodan Mitrovic, Krzysztof Onak, and Piotr Sankowski. 2020. Walking randomly, massively, and efficiently. In *Proccedings of the 52nd ACM SIGACT Symposium on Theory of Computing (STOC'20)*. 364–377.

[38] Longbin Lai, Lu Qin, Xuemin Lin, Ying Zhang, and Lijun Chang. 2016. Scalable distributed subgraph enumeration. *PVLDB* 10, 3 (2016), 217–228.

[39] Amy Nicole Langville and Carl Dean Meyer. 2003. Survey: Deeper inside PageRank. *Internet Math.* 1, 3 (2003), 335–380.

[40] Christoph Lenzen. 2013. Optimal deterministic routing and sorting on the congested clique. In *Proceedings of the 32nd ACM Symposium on Principles of Distributed Computing (PODC'13)*. 42–50.

[41] Christoph Lenzen and Roger Wattenhofer. 2011. Tight bounds for parallel randomized load balancing. In *Proceedings of the 43rd ACM Symposium on Theory of Computing (STOC'11)*. 11–20.

[42] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. 2014. *Mining of Massive Datasets*. Cambridge University Press.

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

[44] Nancy A. Lynch. 1996. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc.

[45] Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. 2010. Pregel: A system for large-scale graph processing. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD'10)*. 135–146.
[46] Danupon Nanongkai. 2014. Distributed approximation algorithms for weighted shortest paths. In *Proceedings of the 46th ACM Symposium on Theory of Computing (STOC'14)*. 565–573.
[47] Danupon Nanongkai, Atish Das Sarma, and Gopal Pandurangan. 2011. A tight unconditional lower bound on distributed randomwalk computation. In *Proceedings of the 30th ACM Symposium on Principles of Distributed Computing (PODC'11)*. 257–266.
[48] Danupon Nanongkai and Michele Scquizzato. 2019. Equivalence classes and conditional hardness in massively parallel computations. In *Proceedings of the 23rd International Conference on Principles of Distributed Systems (OPODIS'19)*. 33:1–33:16.
[49] Hung Q. Ngo, Christopher Ré, and Atri Rudra. 2013. Skew strikes back: New developments in the theory of join algorithms. *SIGMOD Rec.* 42, 4 (2013), 5–16.
[50] Rotem Oshman. 2014. Communication complexity lower bounds in distributed message-passing. In *Proceedings of the 21st International Colloquium on Structural Information and Communication Complexity (SIROCCO'14)*. 14–17.
[51] L. Page, S. Brin, R. Motwani, and T. Winograd. 1999. *The PageRank Citation Ranking: Bringing Order to the Web*. Technical Report. Stanford InfoLab.
[52] Gopal Pandurangan, David Peleg, and Michele Scquizzato. 2016. Message lower bounds via efficient network synchronization. In *Proceedings of the 23rd International Colloquium on Structural Information and Communication Complexity (SIROCCO'16)*. 75–91.
[53] Gopal Pandurangan, David Peleg, and Michele Scquizzato. 2020. Message lower bounds via efficient network synchronization. *Theor. Comput. Sci.* 810 (2020), 82–95.
[54] Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. 2016. Tight bounds for distributed graph computations. *CoRR* abs/1602.08481 (2016).
[55] Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. 2018. Fast distributed algorithms for connectivity and MST in large graphs. *ACM Trans. Parallel Comput.* 5, 1 (2018).
[56] Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. 2018. On the distributed complexity of large-scale graph computations. In *Proceedings of the 30th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'18)*. 405–414.
[57] Ha-Myung Park, Sung-Hyon Myaeng, and U. Kang. 2016. PTE: Enumerating trillion triangles on distributed systems. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'16)*. 1115–1124.
[58] David Peleg. 2000. *Distributed Computing: A Locality-sensitive Approach*. Society for Industrial and Applied Mathematics.
[59] David Peleg and Vitaly Rubinovich. 2000. A near-tight lower bound on the time complexity of distributed minimum-weight spanning tree construction. *SIAM J. Comput.* 30, 5 (2000), 1427–1442.
[60] Daniel A. Reed and Jack Dongarra. 2015. Exascale computing and Big Data. *Commun. ACM* 58, 7 (2015), 56–68.
[61] Fazlollah M. Reza. 1961. *An Introduction to Information Theory*. Courier Corporation.
[62] Igor Rivin. 2001. Counting cycles and finite dimensional L^p norms. *arXiv:math/0111106* (2001).
[63] Vojtech Rödl and Andrzej Ruciński. 1994. Random graphs with monochromatic triangles in every edge coloring. *Rand. Struct. Algor.* 5, 2 (1994), 253–270.
[64] Andrzej Ruciński. 2017. Personal communication.
[65] Christian Scheideler. 1998. *Universal Routing Strategies for Interconnection Networks*. Lecture Notes in Computer Science, Vol. 1390. Springer.
[66] Leslie G. Valiant. 1982. A scheme for fast parallel communication. *SIAM J. Comput.* 11, 2 (1982), 350–361.
[67] Leslie G. Valiant. 1990. A bridging model for parallel computation. *Commun. ACM* 33, 8 (1990), 103–111.
[68] Sergei Vassilvitskii. 2015. Models for Parallel Computation (A Hitchhikers' Guide to Massively Parallel Universes). Retrieved from: http://grigory.us/blog/massively-parallel-universes/.
[69] Nan Wang, Jingbo Zhang, Kian-Lee Tan, and Anthony K. H. Tung. 2010. On triangulation-based dense neighborhood graph discovery. *Proc. VLDB Endow.* 4, 2 (2010), 58–68.
[70] S. Wasserman and K. Faust. 1994. *Social Network Analysis: Methods and Applications*. Cambridge University Press.
[71] Duncan J. Watts and Steven H. Strogatz. 1998. Collective dynamics of "small-world" networks. *Nature* 393 (1998), 440–442.
[72] David P. Woodruff and Qin Zhang. 2017. When distributed computation is communication expensive. *Distrib. Comput.* 30, 5 (2017), 309–323.

Received October 2019; revised January 2021; accepted March 2021