FrogWild! – Fast PageRank Approximations on Graph Engines

Ioannis Mitliagkas
ECE, UT Austin
ioannis@utexas.edu

Michael Borokhovich
ECE, UT Austin
michaelbor@utexas.edu

Alexandros G. Dimakis
ECE, UT Austin
dimakis@austin.utexas.edu

Constantine Caramanis
ECE, UT Austin
constantine@utexas.edu

ABSTRACT

We propose FrogWild, a novel algorithm for fast approximation of high PageRank vertices, geared towards reducing network costs of running traditional PageRank algorithms. Our algorithm can be seen as a quantized version of power iteration that performs multiple parallel random walks over a directed graph. One important innovation is that we introduce a modification to the GraphLab framework that only partially synchronizes mirror vertices. This partial synchronization vastly reduces the network traffic generated by traditional PageRank algorithms, thus greatly reducing the per-iteration cost of PageRank. On the other hand, this partial synchronization also creates dependencies between the random walks used to estimate PageRank. Our main theoretical innovation is the analysis of the correlations introduced by this partial synchronization process and a bound establishing that our approximation is close to the true PageRank vector.

We implement our algorithm in GraphLab and compare it against the default PageRank implementation. We show that our algorithm is very fast, performing each iteration in less than one second on the Twitter graph and can be up to 7× faster compared to the standard GraphLab PageRank implementation.

1. INTRODUCTION

Large-scale graph processing is becoming increasingly important for the analysis of data from social networks, web pages, bioinformatics and recommendation systems. Graph algorithms are difficult to implement in distributed computation frameworks like Hadoop MapReduce and Spark. For this reason several in-memory graph engines like Pregel, Giraph, GraphLab and GraphX [14, 13, 20, 18] are being developed. There is no full consensus on the fundamental abstractions of graph processing frameworks but certain patterns such as vertex programming and the Bulk Synchronous Parallel (BSP) framework seem to be increasingly popular.

PageRank computation [15], which gives an estimate of the importance of each vertex in the graph, is a core component of many search routines; more generally, it represents, de facto, one of the canonical tasks performed using such graph processing frameworks. Indeed, while important in its own right, it also represents the memory, computation and communication challenges to be overcome in large scale iterative graph algorithms.

In this paper we propose a novel algorithm for fast *approximate calculation* of high PageRank vertices. Note that even though most previous works calculate the complete PageRank vector (of length in the millions or billions), in many graph analytics scenarios a user wants a quick estimation of the most important or relevant nodes – distinguishing the 10^{th} most relevant node from the $1,000^{th}$ most relevant is important; the $1,000,000^{th}$ from the $1,001,000^{th}$ much less so. A simple solution is to run the standard PageRank algorithm for fewer iterations (or with an increased tolerance). While certainly incurring less overall cost, the per-iteration cost remains the same; more generally, the question remains whether there is a more efficient way to approximately recover the heaviest PageRank vertices.

In this paper we address this problem. Our algorithm (called FrogWild for reasons that will become subsequently apparent) significantly outperforms the simple reduced iterations heuristic in terms of *running time, network communication and scalability*. We note that, naturally, we compare our algorithm and reduced-iteration-PageRank within the same framework: we implemented our algorithm in GraphLab PowerGraph and compare it against the built-in PageRank implementation. A key part of our contribution also involves the proposal of what appears to be simply a technically minor modification within the GraphLab framework, but nevertheless results in significant network-traffic savings, and we believe may nevertheless be of more general interest beyond PageRank computations.

Contributions: We consider the problem of fast and efficient (in the sense of time, computation and communication costs) computation of the high PageRank nodes, using a graph engine. To accomplish this we propose and analyze an new PageRank algorithm specifically designed for the graph engine framework, and, significantly, we propose a modification of the standard primitives of the graph engine framework (specifically, GraphLab PowerGraph), that

enables significant network savings. We explain in further detail both our objectives, and our key innovations.

Rather than seek to recover the full PageRank vector, we aim for the top k PageRank vertices (where k is considered to be approximately in the order of $10 - 1000$). Given an output of a list of k vertices, we define two natural accuracy metrics that compare the true top-k list with our output. The algorithm we propose, FrogWild operates by starting a small (sublinear in the number of vertices n) number of random walkers (*frogs*) that jump randomly on the directed graph. The random walk interpretation of PageRank enables the frogs to jump to a completely random vertex (teleport) with some constant probability (set to 0.15 in our experiments, following standard convention). After we allow the frogs to jump for time equal to the mixing time of this non-reversible Markov chain, their positions are sampled from the invariant distribution π which is normalized PageRank. The standard PageRank iteration can be seen as the continuous limit of this process (*i.e.*, the frogs become water), which is equivalent to power iteration for stochastic matrices.

The main *algorithmic contributions* of this paper are comprised of the following three innovations. First, we argue that discrete frogs (a quantized form of power iteration) is significantly better for distributed computation when one is interested only in the large entries of the eigenvector π. This is because each frog produces an independent sample from π. If some entries of π are substantially larger and we only want to determine those, a small number of independent samples suffices. We make this formal using standard Chernoff bounds (see also [17, 8] for similar arguments). On the contrary, during standard PageRank iterations, vertices pass messages to all their out-neighbors since a non-zero amount of water must be transferred. This tremendously increases the network bandwidth especially when the graph engine is over a cluster with many machines.

One major issue with simulating discrete frogs in a graph engine is teleportations. Graph frameworks partition vertices to physical nodes and restrict communication on the edges of the underlying graph. Global random jumps would create dense messaging patterns that would increase communication. Our second innovation is a way of obtaining an identical sampling behavior without teleportations. We achieve this by initiating the frogs at uniformly random positions and having them perform random walks for a life span that follows a geometric random variable. The geometric probability distribution depends on the teleportation probability and can be calculated explicitly.

Our third innovation involves a simple proposed modification for graph frameworks. Most modern graph engines (like GraphLab PowerGraph [10]) employ vertex-cuts as opposed to edge-cuts. This means that each vertex of the graph is assigned to multiple machines so that graph edges see a local vertex mirror. One copy is assigned to be the master and maintains the master version of vertex data while remaining replicas are mirrors that maintain local cached read–only copies of the data. Changes to the vertex data are made to the master and then replicated to all mirrors at the next synchronization barrier. This architecture is highly suitable for graphs with high-degree vertices (as most real-world graphs are) but has one limitation when used for a few random walks: imagine that vertex v_1 contains one frog that wants to jump to v_2. If vertex v_1 has very high degree, it is very likely that multiple replicas of that vertex exist, possibly one in each machine in the cluster. In an edge-cut scenario only one message would travel from $v_1 \to v_2$, assuming v_1 and v_2 are located in different physical nodes. However, when vertex-cuts are used, the state of v_1 is updated (i.e., contains no frogs now) and this needs to be communicated to all mirrors. It is therefore possible that *a single random walk can create a number of messages equal to the number of machines in the cluster*.

We modify PowerGraph to expose a scalar parameter p_s per vertex. By default, when the framework is running, in each super-step all masters synchronize their programs and vertex data with their mirrors. Our modification is that for each mirror we flip an independent coin and synchronize with probability p_s. Note that when the master does not synchronize the vertex program with a replica, that replica will not be active during that super-step. Therefore, we can avoid the communication and CPU execution by performing limited synchronization in a randomized way.

FrogWild is therefore executed asynchronously but relies on the Bulk Synchronous execution mode of PowerGraph with the additional simple randomization we explained. The name of our algorithm is inspired by HogWild [16], a lock-free asynchronous stochastic gradient descent algorithm proposed by Niu *et al.*. We note that PowerGraph does support an asynchronous execution mode [10] but we implemented our algorithm by a small modification of synchronous execution. As discussed in [10], the design of asynchronous graph algorithms is highly nontrivial and involves locking protocols and other complications. Our suggestion is that for the specific problem of simulating multiple random walks on a graph, simply randomizing synchronization can give significant benefits while keeping design simple.

While the parameter p_s clearly has the power to significantly reduce network traffic – and indeed, this is precisely born out by our empirical results – it comes at a cost: the standard analysis of the Power Method iteration no longer applies. The main challenge that arises is the theoretical analysis of the FrogWild algorithm. The model is that each vertex is separated across machines and each connection between two vertex copies is present with probability p_s. A single frog performing a random walk on this new graph defines a new Markov Chain and this can be easily designed to have the same invariant distribution π equal to normalized PageRank. The complication is that the trajectories of frogs are *no longer independent*: if two frogs are in vertex v_1 and (say) only one mirror v_1' synchronizes, both frogs will need to jump through edges connected with that particular mirror. Worse still, this correlation effect increases, the more we seek to improve network traffic by further decreasing p_s. Therefore, it is no longer true that one obtains independent samples from the invariant distribution π. Our theoretical contribution is the development of an analytical bound that shows that these dependent random walks still can be used to obtain $\hat{\pi}$ that is provably close to π with high probability. We rely on a coupling argument combined with an analysis of pairwise intersection probabilities for random walks on graphs. In our convergence analysis we use the *contrast bound* [6] for non-reversible chains.

1.1 Notation

Lowercase letters denote scalars or vectors. Uppercase letters denote matrices. The (i, j) element of a matrix A

is A_{ij}. We denote the conjugate transpose of a matrix A by $A^\dagger$ and the transpose by A'. For a time-varying vector x, we denote its value at time t by x^t. When not otherwise specified, $\|x\|$ denotes the l_2-norm of vector x. We use Δ^{n-1} for the probability simplex in n dimensions, and and $e_i \in \Delta^{n-1}$ for the indicator vector for item i. For example, $e_1 = [1, 0, ...0]$. For the set of all integers from 1 to n we write $[n]$.

2. PROBLEM AND MAIN RESULTS

We now make precise the intuition and outline given in the introduction. We first define the problem, giving the definition of PageRank, the PageRank vector, and therefore its top elements. We then define the algorithm, and finally state our main analytical results.

2.1 Problem Formulation

Consider a directed graph $G = (V, E)$ with n vertices ($|V| = n$) and let A denote its adjacency matrix. That is, $A_{ij} = 1$ if there is an edge from j to i. Otherwise, the value is 0. Let $d_{\text{out}}(j)$ denote the number of *successors* (out-degree) of vertex j in the graph. We assume that all nodes have at least one successor, $d_{\text{out}}(j) > 0$. Then we can define the transition probability matrix P as follows:

$$P_{ij} = A_{ij}/d_{\text{out}}(j). \quad (1)$$

The matrix is left-stochastic, which means that each of its rows sums to 1. We call $G(V, E)$ the *original graph*, as opposed to the PageRank graph which includes a probability of transitioning to any given vertex. We now define this transition probability matrix, and the PageRank vector.

DEFINITION 1 (PAGERANK [15]). *Consider the matrix*

$$Q \triangleq (1 - p_T)P + p_T \frac{1}{n} 1_{n \times n}.$$

where $p_T \in [0, 1]$ is a parameter, most commonly set to 0.15. The PageRank vector $\pi \in \Delta^{n-1}$ is defined as the principal right eigenvector of Q. That is, $\pi \triangleq v_1(Q)$. By the Perron-Frobenius theorem, the corresponding eigenvalue is 1. This implies the fixed-point *characterization of the PageRank vector, $\pi = Q\pi$.*

The PageRank vector assigns high values to *important* nodes. Intuitively, important nodes have many important predecessors (other nodes that point to them). This recursive definition is what makes PageRank robust to manipulation, but also expensive to compute. It can be recovered by exact eigendecomposition of Q, but at real problem scales this is prohibitively expensive. In practice, engineers often use a few iterations of the power method to get a "good-enough" approximation.

The definition of PageRank hinges on the left-stochastic matrix Q, suggesting a connection to Markov chains. Indeed, this connection is well documented and studied [1, 9]. An important property of PageRank from its random walk characterization, is the fact that π is the invariant distribution for a Markov chain with dynamics described by Q. A non-zero p_T, also called the *teleportation probability*, introduces a uniform component to the PageRank vector π. We see in our analysis that this implies ergodicity and faster mixing for the random walk.

2.1.1 Top PageRank Elements

Given the true k vertices of highest PageRank (breaking ties randomly for simplicity) and the output list of k vertices by an approximate PageRank algorithm we define accuracy using two metrics.

DEFINITION 2 (MASS CAPTURED). *Given a subset $S \subset [n]$ of size k, and the true PageRank vector, π, we define the* mass captured *by S as follows.*

$$\mu(S) \triangleq \pi(S) = \sum_{i \in S} \pi(i)$$

Let $\mu_{\max,k} = \max_{U \subset [n], |U|=k} \pi(U)$. Then, we can also define the normalized captured mass.

$$\overline{\mu}(S) \triangleq \frac{\mu(S)}{\mu_{\max,k}} \in [0, 1]$$

The second metric we use is the exact identification probability, *i.e.* the fraction of elements in the output list that are also in the true top-k list. Note that the second metric is limited in that it does not give partial credit for high PageRank vertices that were not in the top-k list. In our experiments in Section 3, we mostly use the normalized captured mass accuracy metric but also report the exact identification probability for some cases – typically the results are similar.

We subsequently describe our algorithm. We attempt to approximate the heaviest elements of the invariant distribution of a Markov Chain, by simultaneously performing multiple random walks on the graph. The main modification to PowerGraph, is the exposure of a parameter, p_s, that controls the probability that a given master node synchronizes with any one of its mirrors. Per step, this leads to a proportional reduction in network traffic. The main contribution of this paper is to show that we get results of comparable or improved accuracy, while maintaining this network traffic advantage. We demonstrate this empirically in Section 3.

2.2 Algorithm

During setup, the graph is partitioned using GraphLab's default ingress algorithm. At this point each one of N frogs is born on a vertex chosen uniformly at random. Each vertex i carries a counter initially set to 0 and denoted by $c(i)$. Scheduled vertices execute the following program.

Incoming frogs from previously executed vertex programs, are collected by the `init()` function. At `apply()` every frog dies with probability $p_T = 0.15$. This, along with a uniform starting position, effectively simulates the 15% uniform component from Definition 1.

A crucial part of our algorithm is the change in synchronization behaviour. The <`sync`> step only synchronizes a p_s fraction of mirrors leading to commensurate gains in network traffic (cf. Section 3). This patch on the GraphLab codebase was only a few lines of code. Section 3 contains more details regarding the implementation.

The `scatter()` phase is only executed for edges e incident to a mirror of i that has been synchronized. Those edges draw a binomial number of frogs to send to their other endpoint. The rest of the edges perform no computation. The frogs sent to vertex j at the last step will be collected at the `init()` step when j executes.

FrogWild! vertex program

Input parameters: p_s, $p_T = 0.15$

apply(i) $K(i) \leftarrow$ [# incoming frogs]

For every incoming frog:

With probability p_T, frog dies:

$c(i) = c(i) + 1$,

$K(i) = K(i) - 1$.

<**sync**> For every *mirror* m of vertex i:

With probability p_s:

Synchronize state with mirror m.

scatter($e = (i, j)$) [Only on synchronized mirrors]

Generate Binomial number of frogs:

$$x \sim Bin\left(K(i), \frac{1}{d_{\text{out}}(i)p_s}\right)$$

Send x frogs to vertex j: `signal(j,x)`

Parameter p_T is the teleportation probability from the random surfer model in [15]. To get PageRank using random walks, one could adjust the transition matrix P as described in Definition 1 to get the matrix Q. Alternatively, the process can be replicated by a random walk following the original matrix P, and teleporting at every time, with probability p_T. The destination for this teleportation is chosen uniformly at random from $[n]$. We are interested in the position of a walk at a predetermined point in time as that would give us a sample from π. This holds as long as we allow enough time for mixing to occur.

Due to the inherent markovianity in this process, one could just consider it starting from the last teleportation before the predetermined stopping time. When the mixing time is large enough, the number of steps performed between the last teleportation and the predetermined stopping time, denoted by X, is geometrically distributed with parameter p_T. This follows from the time-reversibility in the teleportation process: inter-teleportation times are geometrically distributed, so as long as the first teleportation event happens before the stopping time, then $X \sim \text{Geom}(\text{p}_\text{T})$.

This establishes that, the FrogWild! process – where a frog performs a geometrically distributed number of steps following the original transition matrix P – closely mimics a random walk that follows the adjusted transition matrix, Q. Hence, to show our main result, Theorem 1, we analyze the latter process.

Using a binomial distribution to independently generate the number of frogs in the `scatter()` phase closely models the effect of random walks. The marginal distributions are correct, and the number of frogs, that did not die during the `apply()` step, is preserved in expectation. For our implementation we resort to a more efficient approach. Assuming $K(i)$ frogs survived the `apply()` step, and M mirrors where picked for synchronization, then we send $\lceil \frac{K(i)}{M} \rceil$ frogs to $\min(K(i), M)$ mirrors. If the number of available frogs is less than the number of synchronized mirrors, we pick $K(i)$ arbitrarily.

2.3 Main Result

Our analytical results essentially provide a high probability guarantee that our algorithm produces a solution that approximates well the PageRank vector. Recall that our main modification of our algorithm involves limiting the synchronization of master nodes and its mirrors. In theory we introduce the following broad model to deal with partial synchronization.

Definition 3 (Edge Erasure Model). *An edge erasure model is a process that is independent from the random walks (up to time t) and temporarily erases a subset of all edges at time t. The event $E^t_{i,j}$ represents the erasure of edge (i, j) from the graph for time t. The edge is not permanently removed from the graph, it is just disabled and considered again in the next step. The edge erasure models we study satisfy the following properties.*

1. *Edges are erased independently for different vertices,*
$$\mathbb{P}(E^t_{i,j}, E^t_{i,k}) = \mathbb{P}(E^t_{i,j})\mathbb{P}(E^t_{i,k})$$
and across time,
$$\mathbb{P}(E^t_{i,j}, E^s_{i,j}) = \mathbb{P}(E^t_{i,j})\mathbb{P}(E^s_{i,j}).$$
2. *Each outgoing edge is preserved (not erased) with probability at least p_s.*
$$\mathbb{P}(\overline{E^t_{i,j}}) \geq p_s$$
3. *Erasures are not negatively correlated.*
$$\mathbb{P}(\overline{E^t_{i,j}}|\overline{E^t_{i,k}}) \geq p_s$$
4. *Erasures in a neighbourhood are symmetric. Any subset of out-going edges of vertex i, will be erased with exactly the same probability as another subset of the same cardinality.*

The main two edge erasure models we consider are described here. They both satisfy all required properties. Our theory holds for both[1], but in our implementation and experiments we use "At Least One Out-Edge Per Node."

Example 4 (Independent Erasures). *Every edge is preserved independently with probability p_s.*

Example 5 (At Least One Out-Edge Per Node). *This edge erasure model, decides all erasures for node i independently, like* Independent Erasures*, but if all out-going edges for node i are erased, it draws and enables one of them uniformly at random.*

Our results tell us that partial synchronization does not change the distribution of a single random walk. To make this and our other results clear, we need the simple definition.

Definition 6. *We denote the state of random walk i at its t^{th} step by s^t_i.*

Then, we see that $\mathbb{P}\big(s^{t+1}_1 = i \big| s^t_1 = j\big) = 1/d_{\text{out}}(j)$, and $x^{t+1}_1 = Px^t_1$. This follows simply by the symmetry assumed in Definition 3. Thus if we were to sample in serial, the modification of the algorithm controlling (limiting) synchronization would not affect each sample, and hence would not affect our estimate of the invariant distribution. However,

[1] *Independent Erasures* can lose some walkers, when it temporarily leads to some nodes having zero out-degree.

we start multiple (all) random walks simultaneously. In this setting, the fundamental analytical challenge stems from the fact that any set of random walks with intersection are now correlated. The key to our result is that we can control the effect of this correlation, as a function the parameter p_s and the *pairwise probability* that two random walks intersect. We define this formally.

DEFINITION 7. *Suppose two walkers l_1 and l_2 perform t steps under the edge erasure model. The probability that they meet is defined as follows.*

$$p_{\cap}(t) \triangleq \mathbb{P}\left(\exists\ \tau \in [0,t],\ \text{s.t. } s^{\tau}_{l_1} = s^{\tau}_{l_2}\right) \tag{2}$$

DEFINITION 8 (ESTIMATOR). *Given the positions of N random walks at time t, $\{s^t_l\}_{l=1}^N$, we define the following estimator for the invariant distribution π.*

$$\hat{\pi}_N(i) \triangleq \frac{\left|\{l : l \in [N], s^t_l = i\}\right|}{N} = \frac{c(i)}{N} \tag{3}$$

Here $c(i)$ refers to the tally maintained by the FROGWILD! *vertex program.*

Now we can state the main result. Here we give a guarantee for the quality of the solution furnished by our algorithm.

THEOREM 1 (MAIN THEOREM). *Consider N frogs following the adjusted transition matrix Q of Definition 1, under the erasure model of Definition 3. The frogs start at independent locations, distributed uniformly and stop after t steps. The top-k set of the estimator $\hat{\pi}_N$ (Definition 8), captures mass close to the optimal. Specifically, with probability at least $1-\delta$,*

$$\mu(\hat{S}^*) \geq \mu_{\max,k} - 2\epsilon',$$

where

$$\epsilon' < \sqrt{k} C \lambda_2^t + \sqrt{\frac{k}{\delta}\left[\frac{1}{N} + (1-p_s^2)p_{\cap}(t)\right]},$$

for some constant $C > 0$ and $\lambda_2 < 1$.

We defer the proof to the appendix. The essence of the result is provided by the following theorem, which guarantees accurate estimation even under the edge erasure model.

The proof is deferred to Appendix A.2. The guaranteed accuracy via this result depends on the probability that two walkers will intersect. Via a simple argument, that probability is the same as the meeting probability for independent walks. The next theorem calculates this probability.

THEOREM 2 (INTERSECTION PROBABILITY). *Consider two independent random walks obeying the same ergodic transition probability matrix, Q with invariant distribution π, as described in Definition 1. Furthermore, assume that both of them are initially distributed uniformly over the state space of size n. The probability that they meet within t steps, is bounded as follows,*

$$p_{\cap}(t) \leq \frac{1}{n} + \frac{t\|\pi\|_{\infty}}{p_T},$$

where $\|\pi\|_{\infty}$, denotes the maximal element of the vector π.

The proof is based on the observation that the l_{∞} norm of a distribution controls the probability that two independent samples coincide. We show that for all steps of the random walk, that norm is controlled by the l_{∞} norm of π. We defer the full proof to Appendix A.3.

2.4 Prior Work

There is a very large body of work on computing and approximating PageRank on different computation models (*e.g.* see [4, 7, 17, 8, 3] and references therein). To the best of our knowledge, our work is the first to specifically design an approximation algorithm for high-PageRank nodes for graph engines. Our base-line comparisons come from the graph framework papers since PageRank is a standard benchmark for running-time, network and other computations. Our implementation is on GraphLab (PowerGraph) and significantly outperforms the built-in PageRank algorithm. This algorithm is already shown in [10, 18] to be significantly more efficient compared to other frameworks like Hadoop, Spark, Giraph *etc.*

3. EXPERIMENTS

In this section we describe the simulations performed for comparing the PageRank algorithms for GraphLab v2.2 (PowerGraph) [13]. We compare two algorithms: GRAPHLAB PR – the basic built-in algorithm provided as a part of the GraphLab *graph analytics toolkit*, and FROGWILD – the algorithm proposed in the current work. Since FROGWILD is an approximation algorithm, it would be unfair to compare its performance to the exact GRAPHLAB PR. So, in order to speedup the GRAPHLAB PR, we tuned its parameters (tolerance and number of iterations) in a way that allows maximum possible accuracy with fastest running time.

Several performance metrics were compared, namely, running time, network usage, and accuracy. The metrics do not include time and network usage required for loading the graph into the GraphLab framework (also know as the *ingress time*). They reflect only what is consumed by the engine executing the PageRank algorithm.

3.1 The Systems

We have performed the experiments on two systems. The first system is a cluster of 20 virtual machines, created using VirtualBox 4.3 [19] on a single physical server. The physical server is based on an Intel® Xeon® CPU E5-1620 with 4 cores at 3.6 GHz, and 16 GB of RAM. As the second system, we used clusters of up to 24 EC2 machines on AWS (Amazon web services) [2]. All the instances were of the type m3.xlarge which are based on Intel® Xeon® CPU E5-2670 and have 4 vCPU and 15 GB RAM.

3.2 The Data

For the VirtualBox system, we use the LiveJournal graph [12] with 4.8M vertices and 69M edges. For the AWS system, in addition to the LiveJournal graph, we use the Twitter graph [11] which has 41.6M nodes and 1.4B edges.

3.3 Implementation

FROGWILD is implemented using the standard GAS (gather, apply, scatter) model. We implement three functions: `init()`, `apply()`, and the `scatter()`. The purpose of the `init()` function is to collect the random walks sent to the node by its neighbors using `scatter()` in the previous iteration. In the first iteration, `init()` generates a random fraction of the initial total amount of random walks. The last implies that the initial random walks are randomly distributed across the nodes. FROGWILD defines the length of random walks to be geometrically distributed (see Section 2.2). However, for the sake of efficiency, we impose an upper bound on the length

of random walks. The algorithm is executed for the constant number of iterations (experiments show good results with even 3 iterations) after which all the random walks are stopped simultaneously. The `apply()` function checks whether the current iteration is the last, in which case it saves the amount of random walks received in `init()` as the PageRank of the vertex. The `scatter()` function is responsible to randomly distribute the random walks received in `init()` to the neighbors of the vertex.

The `scatter()` phase is the most challenging part of the implementation since it is responsible for most of the network traffic generated. In order to reduce information exchange between the machines, we used a couple of interesting ideas. First, notice that random walks in the FrogWild do not have identity. Hence, random walks destined to the same neighbor can be combined to a single message which will carry only their amount. The second optimization is more complicated and involves slight enhancement of the GraphLab's infrastructure. Before starting the scatter execution, GraphLab synchronizes the vertex data and the vertex program between the master and all the mirrors of the node. This is because vertex data and program's variables can be changed in the apply phase, which is executed only on the master replica of a node.

Notice that in our FrogWild implementation there is no need to synchronize vertex data during the whole algorithm execution, since the single vertex's variable, PageRank, is updated only on the last iteration (recall that the node's PageRank is proportional to the amount of random walks that *stopped* at the node). The second observation is that not all the replicas will receive random walks for their parts of node's neighbors. Thus, sending the vertex program to all the mirrors is wasteful, and led us to implement the *random replicas' synchronization*. Namely, we expose to the user parameter p_s which is a fraction of replicas to synchronize (non-synchronized replicas remain idle on the upcoming scatter phase). All the above optimization enhancements required adding only few (about 10) lines to the GraphLab codebase. Our modification of the GraphLab engine as well as our FrogWild vertex program implementation can be found in [5].

3.4 Results

FrogWild is significantly faster and uses less network and computing resources compared to GraphLab PR. Let us start with the Twitter graph and the AWS system. In Figure 1(a) we can see that while the GraphLab PR takes 7.68 sec. per iteration (for 12 nodes), the FrogWild takes 0.99 sec, even with $p_s = 1$, thus achieving more than $7x$ speedup. Reducing the value of p_s will further decrease the running time, however, with the accuracy tradeoff which we will discuss shortly. In order to compare the total running time of our approximation FrogWild algorithm to the GraphLab PR, we reduced the number of iteration performed by the GraphLab PR to 1 and 2. The latter achieves reasonable balance between the running time and the accuracy. The total running time (see Figure 1(b)) shows that FrogWild outperforms GraphLab PR even if it runs for just 1 or 2 iterations. Notice also a large speedup when reducing the p_s from 1 to 0.1, but again, the accuracy metric should be considered.

The network usage improvement can be seen on Figure 1(c). There is a $1000x$ improvement comparing to the exact GraphLab PR, and more than $10x$ with respect to the 1 or 2 iterations GraphLab PR. Another interesting metric is the CPU time used by the algorithms. It gives the total computing time spent by each CPU in the system. Notice that this time may be larger than the total running time since many CPUs in the system work in parallel. This metric shows how "heavy" is the algorithm for the system which is important especially when the algorithm is executed along with another computing tasks. In Figure 1(d) we can see that the CPU usage is much lower for FrogWild.

We now turn to compare the approximation metrics for the PageRank algorithm. For various k, we check the two accuracy metrics: Mass captured (Figure 2(a)) and the Exact identification (Figure 2(b)). Mass captured – is the total PageRank that the reported top-k vertices worth in the exact ranking. Exact identification – is the number of vertices in the intersection of the reported top-k and the exact top-k lists. We can see that the approximation achieved by the FrogWild for $p_s = 1$ and $p_s = 0.7$ always outperforms the GraphLab PR with 1 iteration. The approximation achieved by the FrogWild with $p_s = 0.4$ is relatively good for the both metrics, and with $p_s = 0.1$ is reasonable for the *Mass captured* metrics.

In Figure 3 we can see the tradeoff between the accuracy, total running time, and the network usage. The performance of FrogWild is evaluated for various number of iterations and the values of p_s. The results show that with the accuracy comparable to the GraphLab PR, our FrogWild has much less running time and network usage. Figure 4 illustrate how much network byte we save using FrogWild. The area of each circle is proportional to the amount of network bytes sent by each algorithm.

Now let us show some results for the LiveJournal graph on the VirtualBox system. Figure 5 shows the effect of initial random walks and the number of iteration of the FrogWild on the achieved accuracy. These experiments allow us to tune the parameters for the FrogWild. We can see that good accuracy and the running time (see Figure 6) is achieved for $800K$ initial random walks and 4 iterations of the FrogWild. Also for the LiveJournal graph we can see, in Figure 7, that our algorithm is faster and uses much less network while still maintaining good PageRank accuracy.

4. REFERENCES

[1] A. Agarwal and S. Chakrabarti. Learning random walks to rank nodes in graphs. In *Proceedings of the 24th international conference on Machine learning*, pages 9–16. ACM, 2007.

[2] Amazon web services. http://aws.amazon.com, 2014.

[3] R. Andersen, C. Borgs, J. Chayes, J. Hopcraft, V. S. Mirrokni, and S.-H. Teng. Local computation of pagerank contributions. In *Algorithms and Models for the Web-Graph*, pages 150–165. Springer, 2007.

[4] P. Berkhin. A survey on pagerank computing. *Internet Mathematics*, 2(1):73–120, 2005.

[5] M. Bhorokhovich and I. Mitliagkas. FrogWild! code repository. `https://github.com/michaelbor/frogwild`, 2014. Accessed: 2014-10-30.

[6] P. Bremaud. *Markov chains: Gibbs fields, Monte Carlo simulation, and queues*, volume 31. springer, 1999.

[7] A. Z. Broder, R. Lempel, F. Maghoul, and J. Pedersen. Efficient pagerank approximation via

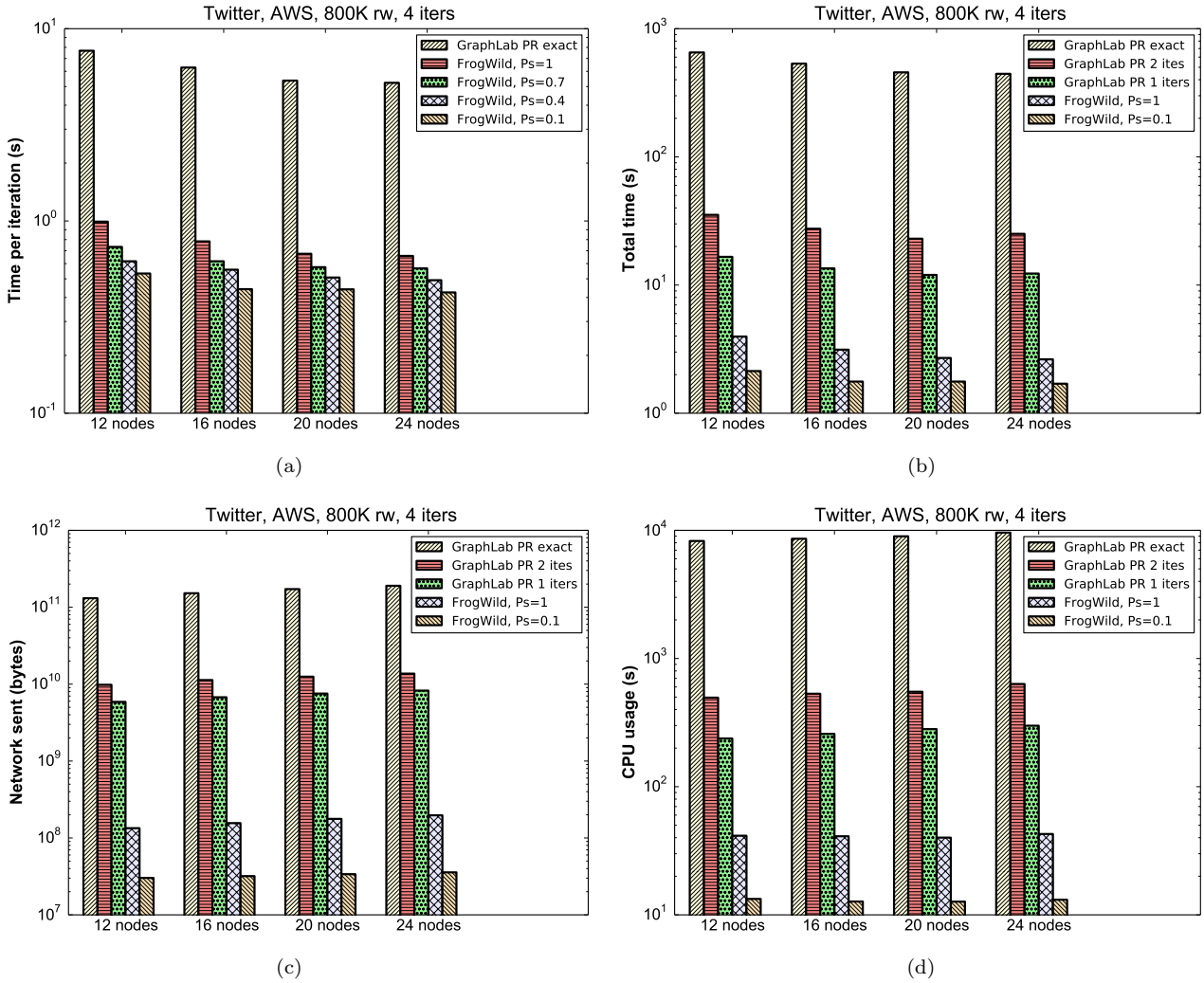


Figure 1: PageRank performance for various number of nodes. (a) – Running time per iteration. (b) – Total running time of the algorithms. (c) – Total network bytes sent by the algorithm during the execution (does not include ingress time). (d) – Total CPU usage time. Notice, this metric may be larger than the total running time since many CPUs run in parallel.

graph aggregation. *Information Retrieval*, 9(2):123–138, 2006.

[8] A. Das Sarma, D. Nanongkai, G. Pandurangan, and P. Tetali. Distributed random walks. *Journal of the ACM (JACM)*, 60(1):2, 2013.

[9] S. Fortunato and A. Flammini. Random walks on directed networks: the case of pagerank. *International Journal of Bifurcation and Chaos*, 17(07):2343–2353, 2007.

[10] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *OSDI*, volume 12, page 2, 2012.

[11] H. Kwak, C. Lee, H. Park, and S. Moon. What is Twitter, a social network or a news media? In *WWW '10: Proceedings of the 19th international conference on World wide web*, pages 591–600, New York, NY, USA, 2010. ACM.

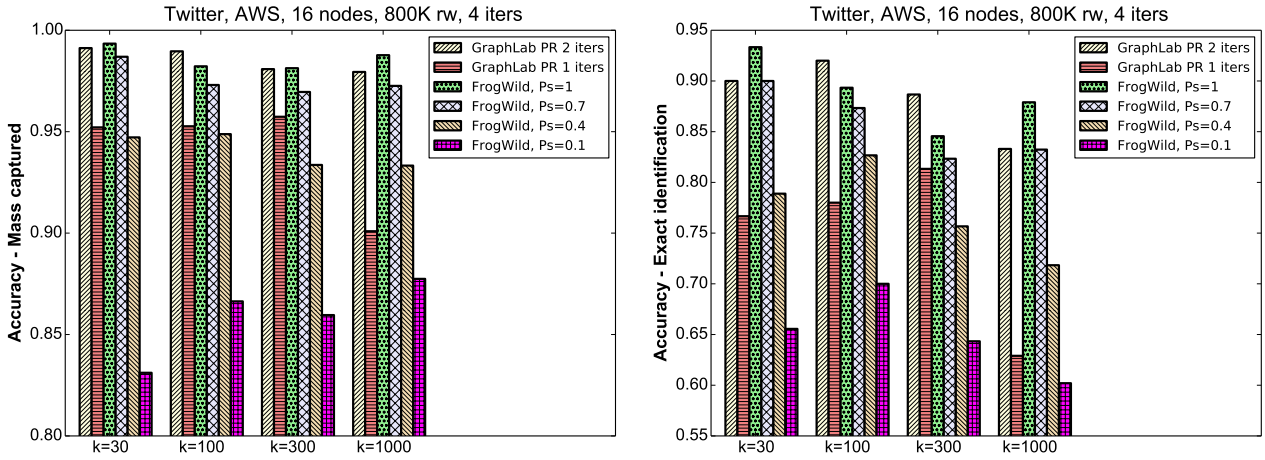
[12] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

[13] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein. Graphlab: A new parallel framework for machine learning. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, July 2010.

[14] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146. ACM, 2010.

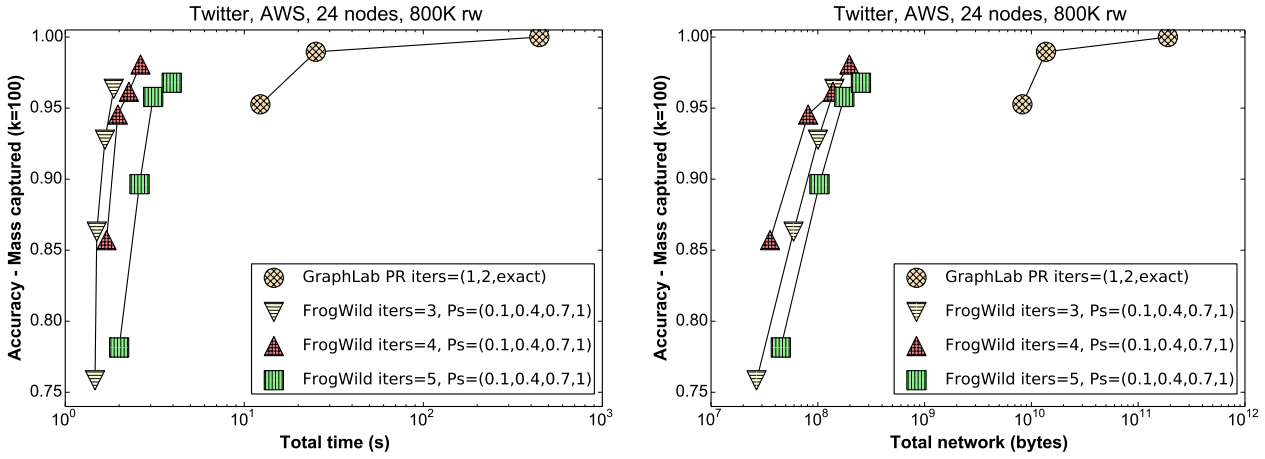
[15] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. 1999.

[16] B. Recht, C. Re, S. Wright, and F. Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 693–701, 2011.



(a) (b)

Figure 2: PageRank approximation accuracy for various number of top-k PageRank vertices. (a) – Mass captured. The total PageRank that the reported top-k vertices worth in the exact ranking. (b) – Exact identification. The number of vertices in the intersection of the reported top-k and the exact top-k lists.



(a) (b)

Figure 3: PageRank approximation accuracy with the "Mass captured" metric for top-100 vertices. (a) - Accuracy versus total running time. (b) - Accuracy versus total network bytes sent.

[17] A. D. Sarma, S. Gollapudi, and R. Panigrahy. Estimating pagerank on graph streams. *Journal of the ACM (JACM)*, 58(3):13, 2011.

[18] N. Satish, N. Sundaram, M. A. Patwary, J. Seo, J. Park, M. A. Hassaan, S. Sengupta, Z. Yin, and P. Dubey. Navigating the maze of graph analytics frameworks using massive graph datasets.

[19] VirtualBox 4.3. www.virtualbox.org, 2014.

[20] R. S. Xin, J. E. Gonzalez, M. J. Franklin, and I. Stoica. Graphx: A resilient distributed graph system on spark. In *First International Workshop on Graph Data Management Experiences and Systems*, page 2. ACM, 2013.

APPENDIX

A. THEOREM PROOFS

A.1 Proof for Theorem 1

PROOF. First we use the result of Theorem 3 to bound the distance between the PageRank vector π and the estimator $\hat{\pi}_N$. Note that, by construction, the adjusted matrix Q is always ergodic and $\min_i \pi(i) \geq c/n$ for $c = p_T$. All of the other assumptions of Theorem 3 are satisfied, hence,

$$\|\hat{\pi}_N - \pi\|_2 < C\lambda_2 + \sqrt{\frac{1}{\delta}\left[\frac{1}{N} + (1-p_s^2)p_\cap(t)\right]},$$

with probability at least $1-\delta$. Here $C = \left(\frac{1-p_T}{p_T}\right)$, and $\lambda_2 = \lambda_2(\tilde{Q}Q)^t$. We define the top-$k$ sets

$$\hat{S}^* = \text{argmax}_{S\subset[n],|S|=k}\hat{\pi}_N(S)$$

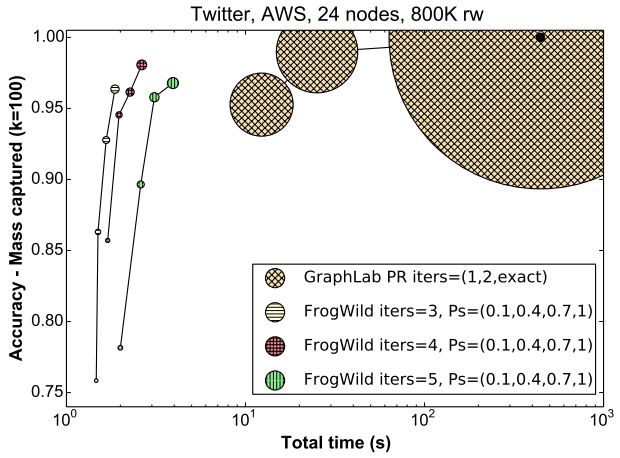


Figure 4: PageRank approximation accuracy with the "Mass captured" metric for top-100 vertices. The area of each circle is proportional to the total network bytes sent by the specific algorithm.

$$S^* = \text{argmax}_{S\subset[n],|S|=k}\pi(S).$$

Let $\pi\big|_S$ denote the restriction of vector π to the set S. That is, $\pi\big|_S(i) = \pi(i)$ if $i \in S$ and 0 otherwise. Now we show that for any set S of cardinality k,

$$\begin{aligned}|\pi(S) - \hat{\pi}_N(S)| &\leq \|(\pi - \hat{\pi}_N)\big|_S\|_1 \leq \sqrt{k}\|(\pi - \hat{\pi}_N)\big|_S\|_2 \\ &\leq \sqrt{k}\|\pi - \hat{\pi}_N\|_2 \triangleq \epsilon' \qquad (4)\end{aligned}$$

Here we used the fact that for k-length vector x, $\|x\|_1 \leq \sqrt{k}\|x\|_2$ and $\|x\big|_S\| \leq \|x\|$. As a consequence of (4),

$$\begin{aligned}\hat{\pi}_N(\hat{S}^*) &= \max_{S\subset[n],|S|=k}\hat{\pi}_N(S) \\ &\geq \hat{\pi}_N(S^*) \geq \pi(S^*) - \epsilon'. \qquad (5)\end{aligned}$$

Finally, using (4) and (5) in order we get

$$\begin{aligned}\mu(\hat{S}^*) = \pi(\hat{S}^*) &\geq \hat{\pi}_N(\hat{S}^*) - \epsilon' \\ &\geq \pi(S^*) - 2\epsilon' \geq \mu_{\max,k} - 2\epsilon'.\end{aligned}$$

This concludes the proof. □

A.2 Theorem 3

THEOREM 3 (CONVERGENCE WITH ERASURES). *Let N walkers start from independent, uniformly random positions and take t steps according to the ergodic transition matrix P, under the edge erasure model. Let π denote the unique stationary distribution of P and assume that $\min_i \pi(i) \geq \frac{c}{n}$ for some constant $c \leq 1$. The estimator $\hat{\pi}_N$, described in Definition 8, falls inside a neighbourhood of π with good probability. Specifically, with probability at least $1-\delta$,*

$$\|\hat{\pi}_N - \pi\|_2 < \left(\frac{1-c}{c}\right)\lambda_2(\tilde{P}P)^t + \sqrt{\frac{1}{\delta}\left[\frac{1}{N} + (1-p_s^2)p_\cap(t)\right]},$$

where $\tilde{P} = D^{-1}P'D$, for $D = \text{diag}(\pi)$.

For the convergence of the associated Markov chain we rely on Proposition 11, taken from [6]. First we need the following definition.

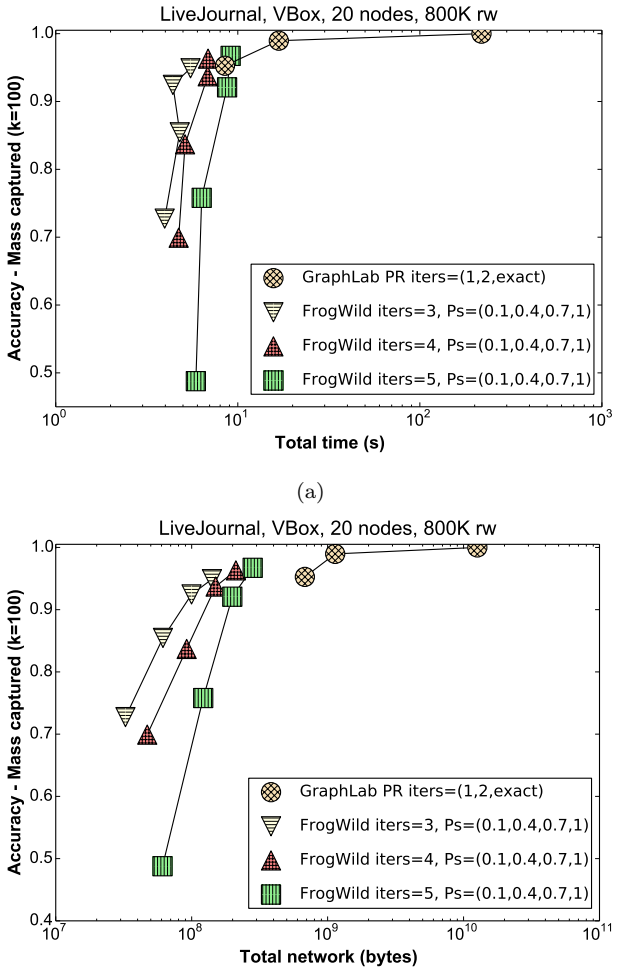


Figure 7: (a) – Accuracy (fraction of nodes in the true top 100 that exists in the top 100 reported by the algorithm) of the PageRank algorithms as a function of running time. The running time varies due to the number of iterations performed by the algorithms. (b) – Accuracy as a function of network usage. The network usage varies due to the number of iterations performed by the algorithms.

DEFINITION 9 (χ^2-CONTRAST). *The χ^2-contrast $\chi^2(\alpha;\beta)$ of α with respect to β is defined by*

$$\chi^2(\alpha;\beta) = \sum_i \frac{(\alpha(i) - \beta(i))^2}{\beta(i)}.$$

LEMMA 10. *Let $\pi \in \Delta^{n-1}$ a distribution satisfying $\min_i \pi(i) \geq \frac{c}{n}$ for constant $c \leq 1$, and let $u \in \Delta^{n-1}$ denote the uniform distribution. Then, $\chi^2(u;\pi) \leq \left(\frac{1-c}{c}\right)$.*

PROOF.

$$\begin{aligned}\chi^2(u;\pi) &= \sum_i \frac{(1/n - \pi(i))^2}{\pi(i)} = \sum_i \left(\frac{1}{n^2\pi(i)} - \frac{1}{n}\right) \\ &= \frac{1}{n}\sum_i \frac{1 - n\pi(i)}{n\pi(i)} \leq \frac{1}{n}\sum_i \frac{1-c}{c} = \frac{1-c}{c}\end{aligned}$$

Here we used the assumed lower bound on $\pi(i)$ and the fact that $(1-x)/x$ is decreasing in x. $\square$

PROPOSITION 11 ([6]). *Consider an ergodic transition matrix P on a finite state space, and let π denote its unique stationary distribution. Let, $\tilde{P} = D^{-1}P'D$ be the* multiplicative reversibilization *of P. Then for any starting distribution ν, after t steps we get $\|P^t\nu - \pi\|_1^2 \leq \lambda_2(\tilde{P}P)^t\chi^2(\nu;\pi)$.*

Now we have all the necessary tools to prove the main theorem.

PROOF OF THEOREM 3. We know, that the individual walk distributions (marginals) follow the dynamics $x_l^{t+1} = Px_l^t$, for all $l \in [N]$, i.e. $x_l^t = x_1^t$. First we show that $\|\hat{\pi}_N - x_1^t\|_2$ is small.

$$\mathbb{P}(\|\hat{\pi}_N - x_1^t\|_2 > \epsilon) \leq \frac{\mathbb{E}[\|\hat{\pi}_N - x_1^t\|_2^2]}{\epsilon^2} \tag{6}$$

Here we used the Chebyshev inequality. We use s_l^t to denote the position of walker l at time t as a vector. For example, $s_l^t = e_i$, if walker l is at state i at time t. Now let us break down the norm on the numerator of (6).

$$\begin{aligned}\|\hat{\pi}_N - x_1^t\|_2^2 &= \left\|\frac{1}{N}\sum_l (s_l^t - x_1^t)\right\|_2^2 \\ &= \frac{1}{N^2}\sum_l \|s_l^t - x_1^t\|_2^2 + \frac{1}{N^2}\sum_{l\neq k}(s_l^t - x_1^t)'(s_k^t - x_1^t) \quad (7)\end{aligned}$$

For the diagonal terms we have:

$$\begin{aligned}\mathbb{E}[\|s_l^t - x_1^t\|_2^2] &= \sum_{i\in[n]} \mathbb{E}\left[\|s_l^t - x_1^t\|_2^2 | s_l^t = i\right]\mathbb{P}(s_l^t = i) \\ &= \sum_{i\in[n]} \|e_i^t - x_1^t\|_2^2 x_1^t(i) = 1 - \|x_1^t\|_2^2 \leq 1 \quad (8)\end{aligned}$$

Under the edge erasures model, the trajectories of different walkers are not generally independent. For example, if they happen to meet, they are likely to make the same decision for their next step, since they are faced with the same edge erasures. Now we prove that even when they meet, we can consider them to be independent with some probability that depends on p_s.

Consider the position processes for two walkers, $\{s_1^t\}_t$ and $\{s_2^t\}_t$. At each step t and node i a number of out-going edges are erased. Any walkers on i, will choose uniformly at random from the remaining edges. Now consider this *alternative process.*

DEFINITION 12 (BLOCKING WALK). *A blocking walk on the graph under the erasure model, follows these steps.*

1. *Walker l finds herself on node i at time t.*
2. *Walker l draws her next state uniformly from the* full *set of out-going edges.*

$$w \sim \text{Uniform}(\mathcal{N}_o(i))$$

3. *If the edge (i,w) is erased at time t, the walker cannot traverse it. We call this event a* block *and denote it by B_l^t. In the event of a block:*
   - *Walker redraws her next step from the out-going edges of i* not erased *at time t.*
   - *Otherwise, w is used as the next state.*

Clearly, a blocking walk is exactly equivalent to our original process; walkers end up picking a destination uniformly at random among the edges not erased. From now on we focus on this description of our original process. We use the same notation: $\{s_l^t\}_t$ for the position process and $\{x_l^t\}_t$ for the distribution at time t.

Let us focus on just two walkers, $\{s_1^t\}_t$ and $\{s_2^t\}_t$ and consider a third process: two independent random walks on the same graph. We assume that these walks operate on the complete graph, i.e. no edges are erased. We denote their positions by $\{v_1^t\}_t$ and $\{v_2^t\}_t$ and their marginal distributions by $\{z_1^t\}_t$ and $\{z_2^t\}_t$.

DEFINITION 13 (TIME OF FIRST INTERFERENCE). *For two blocking walks, τ_I denotes the earliest time at which they meet and at least one of them experiences blocking.*

$$\tau_I = \min\left\{t : \{s_1^t = s_2^t\} \cap (B_1^t \cup B_2^t)\right\}$$

We call this quantity the time of first interference.

LEMMA 14 (PROCESS EQUIVALENCE). *For two walkers, the blocking walk and the independent walk are identical until the time of first interference. That is, assuming the same starting distributions, $x_1^0 = z_1^0$ and $x_2^0 = z_2^0$, then*

$$x_1^t = z_1^t \quad \text{and} \quad x_2^t = z_2^t \qquad \forall t \leq \tau_I.$$

PROOF. The two processes are equivalent for as long as the blocking walkers make *independent* decisions effectively picking *uniformly from the full set of edges* (before erasures). From the independence in erasures across time and vertices in Definition 3, as long as the two walkers do not meet, they are making an independent choices. Furthermore, since erasures are symmetric, the walkers will be effectively choosing uniformly over the full set of out-going edges.

Now consider any time t that the blocking walkers meet. As long as neither of them blocks, they are by definition taking independent steps uniformly over the set of all outgoing edges, maintaining equivalence to the independent walks process. This concludes the proof. $\square$

LEMMA 15. *Let all walkers start from the uniform distribution. The probability that the time of first interference comes before time t is upper bounded as follows. $\mathbb{P}(\tau_I \leq t) \leq (1-p_s^2)p_\cap(t)$*

PROOF. Let M_t be the event of a meeting at time t, $M_t \triangleq \{s_1^t = s_2^t\}$. In the proof of Theorem 2, we establish that $\mathbb{P}(M_t) \leq \rho^t/n$,where ρ is the maximum row sum of the transition matrix P. Now denote the event of an interference at time t as follows. $I_t \triangleq M_t \cap (B_1^t \cup B_2^t)$, where B_1^t denotes the event of blocking, as described in Definition 12. Now,

$$\mathbb{P}(I_t) = \mathbb{P}(M_t \cap (B_1^t \cup B_2^t)) = \mathbb{P}(B_1^t \cup B_2^t | M_t)p_\cap(t).$$

For the probability of a block given that the walker have met at time t,

$$\begin{aligned}\mathbb{P}(B_1^t \cup B_2^t | M_t) &= 1 - \mathbb{P}(\overline{B_1^t} \cap \overline{B_2^t} | M_t) \\ &= 1 - \mathbb{P}(\overline{B_2^t} | \overline{B_1^t}, M_t)\mathbb{P}(\overline{B_1^t}\,|M_t) \leq 1 - p_s^2.\end{aligned}$$

To get the last inequality we used, from Definition 3, the lower bound on the probability that an edge is not erased, and the lack of negative correlations in the erasures.

Combining the above results, we get

$$\mathbb{P}(\tau_I \le t) = \mathbb{P}\left(\sum_{\tau=1}^{t} \mathbb{I}_{\{I_\tau\}} \ge 1\right) \le \mathbb{E}\left[\sum_{\tau=1}^{t} \mathbb{I}_{\{I_\tau\}}\right] = \sum_{\tau=1}^{t} \mathbb{P}(I_\tau)$$
$$\le \sum_{\tau=1}^{t} (1-p_s^2)\mathbb{P}(M_\tau) = \frac{1-p_s^2}{n} \sum_{\tau=1}^{t} \rho^\tau = (1-p_s^2)p_\cap(t)$$

which proves the statement. □

Now we can bound the off-diagonal terms in (8).

$$\mathbb{E}\Big[(s_l^t - x_1^t)'(s_k^t - x_1^t)\Big] = \mathbb{E}\Big[(s_l^t - x_1^t)'(s_k^t - x_1^t)\big|\tau_I \le t\Big]\mathbb{P}(\tau_I \le t)$$
$$+ \mathbb{E}\Big[(s_l^t - x_1^t)'(s_k^t - x_1^t)\big|\tau_I > t\Big]\mathbb{P}(\tau_I > t)$$

In the second term, the case when l, k have not interfered, by Lemma 14, the trajectories are independent and the cross-covariance is 0. In the first term, the cross-covariance is maximized when $s_l^t = s_k^t$. That is,

$$\mathbb{E}\Big[(s_l^t - x_1^t)'(s_k^t - x_1^t)\big|\tau_I \le t\Big] \le \mathbb{E}[\|s_l^t - x_1^t\|_2^2] \le 1$$

From this we get

$$\mathbb{E}\Big[(s_l^t - x_1^t)'(s_k^t - x_1^t)\Big] \le (1-p_s^2)p_\cap(t), \tag{9}$$

and in combination with (8), we get from (7) that

$$\mathbb{E}\Big[\|\hat{\pi}_N - x_1^t\|_2^2\Big] \le \frac{1}{N} + \frac{(N-1)(1-p_s^2)p_\cap(t)}{N}.$$

Finally, we can plug this into (6), to get

$$\mathbb{P}(\|\hat{\pi}_N - x_1^t\|_2 > \epsilon) \le \frac{1 + (1-p_s^2)p_\cap(t)(N-1)}{N\epsilon^2}.$$

We can now combine this bound, with a bound on $\|x_1^t - \pi\|_2$ using the triangle inequality to get the theorem statement. We first use the fact that $\|x_1^t - \pi\|_2 \le \|x_1^t - \pi\|_1$ and then apply Proposition 11 on the RHS, assuming a uniform starting distribution and invoking Lemma 10.

A.3 Proof of Theorem 2

Proof. Let $u \in \Delta^{n-1}$ denote the uniform distribution over $[n]$, i.e. $u_i = 1/n$. The two walks start from the same initial uniform distribution, u, and independently follow the same law, Q. Hence, at time t they have the same marginal distribution, $p^t = Q^t u$. From the definition of the augmented transition probability matrix, Q, in Definition 1, we get that

$$\pi_i \ge \frac{p_T}{n}, \quad \forall i \in [n].$$

Equivalently, there exists a distribution $q \in \Delta^{n-1}$ such that

$$\pi = p_T u + (1-p_T)q.$$

Now using this, along with the fact that π is the invariant distribution associated with Q (i.e. $\pi = Q^t\pi$ for all $t \ge 0$) we get that for any $t \ge 0$,

$$\|\pi\|_\infty = \|Q^t\pi\|_\infty$$
$$= \|Q^t p_T u + Q^t(1-p_T)q\|_\infty$$
$$\ge p_T\|Q^t u\|_\infty.$$

For the last inequality, we used the fact that Q and q contain non-negative entries. Now we have a useful upper bound for the maximal element of the walks' distribution at time t.

$$\|p^t\|_\infty = \|Q^t u\|_\infty \le \frac{\|\pi\|_\infty}{p_T} \tag{10}$$

Let M_t be the indicator random variable for the event of a meeting at time t.

$$M_t = \mathbb{I}_{\{\text{walkers meet at time } t\}}$$

Then, $\mathbb{P}(M_t = 1) = \sum_{i=1}^n p_i^t p_i^t = \|p^t\|_2^2$. Since p^0 is the uniform distribution, i.e. $p_i^0 = \frac{1}{n}$ for all i, then $\|p^0\|_2^2 = \frac{1}{n}$. We can also bound the l_2 norm of the distribution at other times. First, we upper bound the l_2 norm by the l_∞ norm.

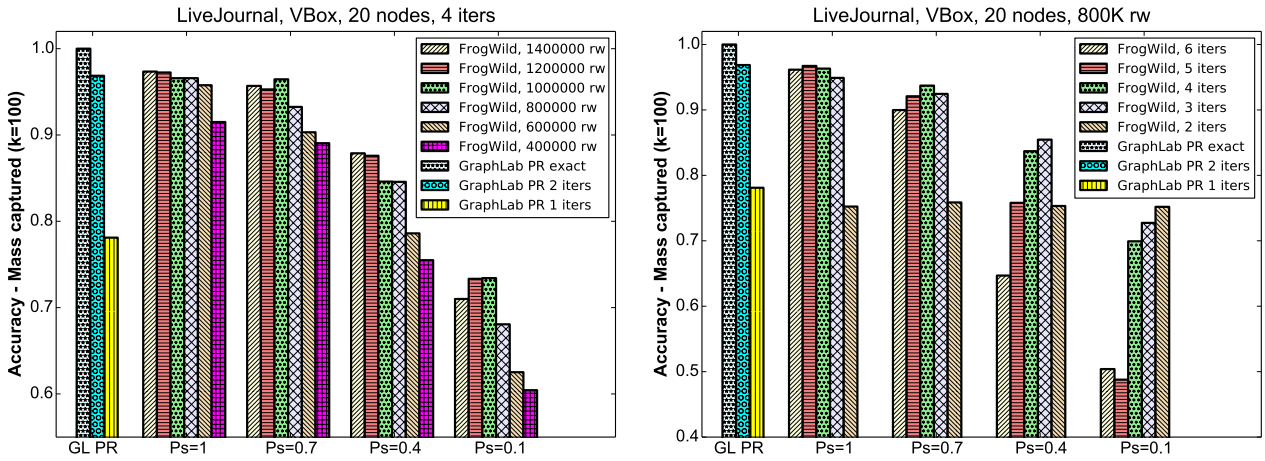
$$\|p\|_2^2 = \sum_i p_i^2 \le \sum_i p_i\|p\|_\infty = \|p\|_\infty$$

Here we used the fact that $p_i \ge 0$ and $\sum p_i = 1$.

Now, combining the above results, we get

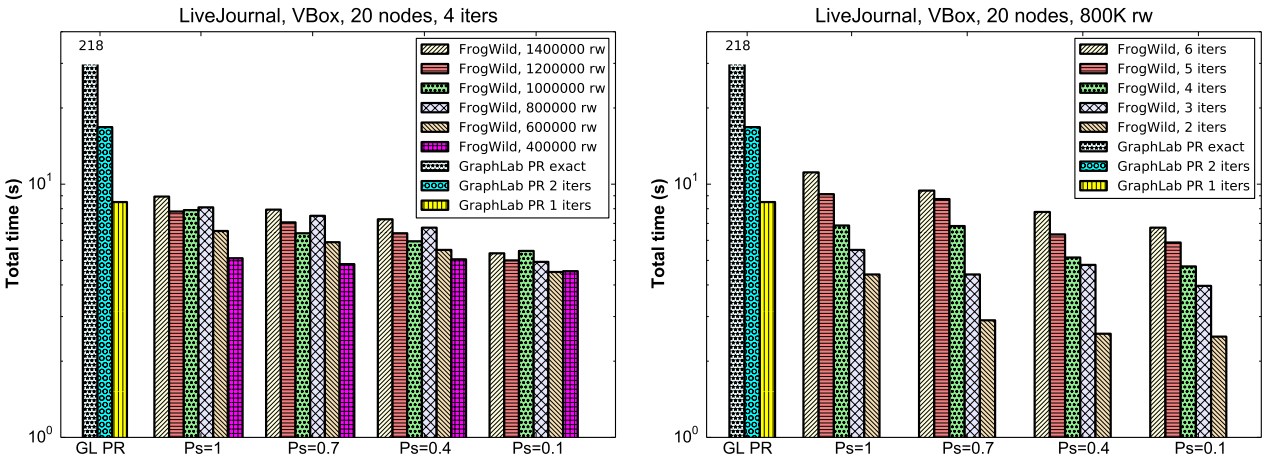
$$p_\cap(t) = \mathbb{P}\left(\sum_{\tau=0}^{t} M_\tau \ge 1\right) \le \mathbb{E}\left[\sum_{\tau=0}^{t} M_\tau\right] = \sum_{\tau=0}^{t} \mathbb{E}[M_\tau]$$
$$= \sum_{\tau=0}^{t} \mathbb{P}(M_\tau = 1) = \sum_{\tau=0}^{t} \|p^\tau\|_2^2 \le \sum_{\tau=0}^{t} \|p^\tau\|_\infty$$
$$\le \frac{1}{n} + \frac{t\|\pi\|_\infty}{p_T}.$$

For the last inequality, we used (10) for $t \ge 1$ and $\|p^0\|_2^2 = 1/n$. This proves the theorem statement. □



(a) (b)

Figure 5: PageRank approximation accuracy with the "Mass captured" metric for top-100 vertices. (a) – Accuracy for various number of initial random walks in the FrogWild. (b) – Accuracy for various number of iterations of FrogWild.



(c) (d)

Figure 6: Total running time of the PageRank algorithms. (a) – Total running time for various number of initial random walks in the FrogWild. (b) – Total running time for various number of iterations of FrogWild.