Solving for a Single Component of the Solution to a Linear System, Asynchronously

Christina E. Lee celee@mit.edu
Asuman Ozdaglar asuman@mit.edu
Devavrat Shah devavrat@mit.edu

Laboratory for Information and Decision Systems
Massachusetts Institute of Technology

July 23, 2015

Abstract

We present synchronous and asynchronous randomized variants of an algorithm for approximating a single component of the solution to a system of linear equations $Ax = b$, where A is a positive definite real matrix and $b \in \mathbb{R}^n$. This can equivalently be formulated as solving for x_i in $x = Gx + z$ for some G and z such that the spectral radius of G is less than 1. We consider the setting where n is large, yet G is sparse, i.e., each row has at most d nonzero entries. Our algorithm relies on the Neumann series characterization of the component x_i. Both variants of our algorithm produce an estimate $\hat{x}_i$ such that $|\hat{x}_i - x_i| \leq \epsilon \|x\|_2$, and we provide convergence rate guarantees when $\|G\|_2 < 1$. Our algorithms obtain an approximation for x_i in constant time with respect to the size of the matrix when $d = O(1)$ and $1/(1 - \|G\|_2) = O(1)$. This requires analyzing the product of random matrices which are drawn from distributions that are time and path dependent. We also give general conditions under which the asynchronous algorithm converges asymptotically regardless of the order and frequency of updates.

1 Introduction

Imagine that you are a small restaurant owner in a city. You would like to obtain a quantitative estimate of how your popularity and reputation compares with your competitors within a 5 mile radius of you. You want to compare the PageRank[1] of the associated websites of your restaurant and other similar restaurants, and the Bonacich centrality[2] of your corresponding Facebook pages. Both PageRank and Bonacich centrality can be formulated as the solution to a system of linear equations, where the dimension is as large as the webpages in the webgraph or the number of Facebook pages, which is an overwhelming computational expense for our hypothetical small restaurant owner. In this paper, we investigate the question: can we obtain estimates of a few coordinates of the solution vector without the expense of approximating the entire solution vector?

We consider approximating the i^{th} component of the solution to the linear system of equations $Ax = b$, where A is a nonsingular $n \times n$ real matrix, and b is a vector in $\mathbb{R}^n$. When A is positive

[1] PageRank is defined as the solution to $x = \alpha \mathbf{1}/n + (1-\alpha) P^T x$, where P is the adjacency matrix of the webgraph, α is a given parameter, $\mathbf{1}$ is the vector of all ones, and n is the dimension.

[2] Bonacich centrality is defined as the solution to $x = (I - \alpha G)^{-1} \mathbf{1}$, where G is the adjacency matrix of the social network, and α is a given parameter.

definite, there exists a choice of G and z such that $\rho(G) < 1$,[3] and this problem is equivalent to approximating the i^{th} component of the solution to $x = Gx + z$. We consider the setting where n is large, yet G is sparse, i.e., the number of nonzero entries in every row of G is at most d. This form of sparsity comes from settings in which the matrix is derived from an underlying bounded degree graph.

Solving large systems of linear equations is a problem of great interest due to its relevance to a variety of applications across science and engineering, such as solving large scale optimization problems, approximating solutions to partial differential equations, and modeling network centralities (e.g. PageRank and Bonacich centrality). Due to the large scale of these systems, it becomes useful to have an algorithm which can approximate only a few components of the solution without computing over the entire matrix. As solving a system of linear equations is fundamentally a problem which involves the full matrix, obtaining a single component solution is non-trivial.

1.1 Contributions

In this paper, we propose and analyze two variants of an algorithm which provides an estimate $\hat{x}_i$ for a single component of the solution vector to $x = Gx + z$, such that $|\hat{x}_i - x_i| \le \epsilon \|x\|_2$. One variant is synchronous and the other is asynchronous and randomized. We show that our algorithm converges when $\rho(G) < 1$ and provide bounds on the convergence rate when $\|G\|_2 < 1$.[4] Assuming that each row of G has at most d nonzero entries, we prove that the synchronous local algorithm uses at most $O(\min(d\epsilon^{\ln(d)/\ln(\|G\|_2)}, dn\ln(\epsilon)/\ln(\|G\|_2)))$ multiplications, and with probability at least $1-\delta$, the asynchronous randomized algorithm uses at most $O(\min(d(\epsilon\sqrt{\delta/5})^{-d/(1-\|G\|_2)}, -dn\ln(\epsilon\sqrt{\delta})/(1 - \|G\|_2)))$ multiplications. We can compare the bounds for the synchronous and asynchronous variants by considering that $1 - \|G\|_2 \approx -\ln(\|G\|_2)$ when $\|G\|_2 \approx 1$. The two right hand expressions within the convergence rate bounds are essentially the same, and provide a comparison of our algorithm to standard linear iterative methods. The left hand expressions provide a local analysis utilizing the sparsity of G. They show that our algorithm is constant cost with respect to n as long as $d = O(1)$ and $1/(1 - \|G\|_2) \approx -1/\ln(\|G\|_2) = O(1)$.

Our algorithm relies upon the Neumann series representation of the solution to $x = Gx + z$,[5]

$$x_i = e_i^T \sum_{k=0}^{\infty} G^k z = z^T \sum_{k=0}^{\infty} (G^T)^k e_i. \tag{1}$$

Since we focus on approximating only x_i, we can compute the inner terms of the summation using $(G^T)^k e_i$, which has the sparsity pattern of the k-hop neighborhood of node i. This allows us to guarantee that intermediate vectors involved in our algorithm are sparse as long as G is sparse. Our algorithm provides the estimate and residual error at each iteration, and recursively refines the estimate while contracting the error, providing clear termination conditions.

The asynchronous variant implements the local algorithm by adaptively sampling a coordinate from the current iterate, and applying a local update involving the corresponding row from the matrix G. Our model of asynchronous computation assumes that each node is equipped with a Poisson clock and when the clock ticks, it decides to do computation, eliminating coordination cost between nodes. Recently this model has been used in literature in the context of gossip algorithms

[3] Let $\rho(G)$ denote the spectral radius of G, i.e., the maximum eigenvalue of G in absolute value.

[4] Let $\|G\|_2 = \max_{\|v\|_2=1} \|Gv\|_2 = \sqrt{\rho(G^T G)}$. When G is symmetric, $\rho(G) = \|G\|_2$.

[5] Let e_i denote the standard basis vector which has as 1 at coordinate i and 0 elsewhere.

[13]. The updates satisfy an invariant which characterizes the error at every iteration. We show that the algorithm converges for any sequence of updates such that each coordinate updates infinitely often, allowing for finite communication delay. When the updates are chosen uniformly at random amongst the nonzero valued coordinates, with high probability, the error contracts by a time varying factor in each step, guaranteeing that the algorithm converges with a similar rate as the synchronous method, obtaining a tighter bound than the analysis found in an earlier work [1]. Establishing the convergence rate requires bounding the Lyapunov exponent for a product of random matrices drawn from time and path dependent distributions.

Throughout the paper, we will associate a graph to the matrix G, and use the concept of computation over this graph to discuss our notion of an asynchronous algorithm. Let $\mathcal{G}(G) = (\mathcal{V}, \mathcal{E})$ denote the graph where $\mathcal{V} = \{1, 2, \dots n\}$, and $(a, b) \in \mathcal{E}$ if and only if $G_{ab} \neq 0$. Each coordinate in x corresponds to a node in $\mathcal{V}$. The sparsity assumption on G means that the outdegree of any node in $\mathcal{G}(G)$ is at most d. The distance from node i to node j is the shortest path from i to j in $\mathcal{G}(G)$. Let $N_i(t) \subset \mathcal{V}$ denote the neighborhood of radius t around node i, which consists of the set of nodes within distance t from node i. The computation of our algorithm only involves coordinates within $N_i(t)$ for some t which is a function of ϵ, $\|G\|_2$, and d.

1.2 Related Work

There are few existing methods which have explored single component approximations of the solution vector. Standard techniques such as Gaussian elimination, factorization or decomposition, gradient methods, and linear iterative methods all output the full solution vector, and thus the computation involves all coordinates and all entries in the matrix [18, 4]. There are nearly linear time[6] approximation algorithms for sparse and symmetric diagonally dominant matrices A, however they involve global structures over the graph, such as graph sparsifiers or spanning trees [14, 9, 8].

The Ulam von Neumann algorithm is a Monte Carlo method which obtains an estimate for x_i by sampling random walks. It interprets the Neumann series representation of the solution x as a sum over weighted walks on $\mathcal{G}(G)$, and obtains an estimate by sampling random walks starting from node i over $\mathcal{G}(G)$ and appropriately reweighting to obtain an unbiased estimator [3, 17, 2]. The challenge is to control the variance of this estimator, which could be large. The classic choice for the sampling matrix requires $\|G\|_\infty < 1$, though there are modifications which propose other sampling matrices or use correlated sampling to reduce the variance [5, 6]. The scope of this algorithm is limited, as Ji, Mascagni, and Li proved that there is a class of matrices such that $\rho(G) < 1$, $\|G\|_\infty > 1$, and there does not exist any sampling matrix such that the variance of the corresponding estimator is finite [7]. In contrast, our algorithm exploits the sparsity of G and provides a convergent solution when $\rho(G) < 1$ and convergence rates when $\|G\|_2 < 1$.

Our work is most related to stationary linear iterative methods, which use updates of the form $x_{t+1} = Gx_t + z$ to recursively approximate leading terms of the Neumann series. The error after t iterations is given by $G^t(x - x_0)$, thus the number of iterations to achieve $\|x_t - x\|_2 \leq \epsilon \|x\|_2$, is given by $\ln(\epsilon)/\ln(\|G\|_2)$. For any t, x_t will be at least as dense as z, and there is no reason to assume that z is sparse. Thus a single update step could cost nd multiplications. These methods do not exploit the sparsity of G and the locality of computing a single component.

Our asynchronous method is closely related to work by Andersen *et al.* which focuses on computing PageRank, and provides an algorithm and analysis which rely on the conditions that

[6] $O(m \log^c n \log \epsilon^{-1})$, where m is the number of nonzero entries in A, and $c \in \mathbb{R}_+$ is a fixed constant.

G is a nonnegative scaled stochastic matrix, z is entry-wise positive and bounded strictly away from zero, and the solution x is a probability vector (i.e., consisting of nonnegative entries that sum to 1) [1]. In contrast, our algorithm focuses on general linear systems of equations. Moreover, while their analysis proves a linear decrease in the error, we prove that the second moment of our error contracts by a time dependent factor in each iteration, and thus our algorithm converges to the correct solution with a convergence rate which is similar to the synchronous variant. Our algorithm differs from the algorithm presented in Andersen *et al.* by a different choice of termination conditions, and the use of randomization in the update of every iteration.

The use of randomization in subsampling matrices as part of a subroutine in iterative methods has previously been used in the context of other global matrix algorithms, such as the randomized Kaczmarz method and stochastic iterative projection [15, 12, 11, 10, 16]. The randomized Kaczmarz method is used in the context of solving overdetermined systems of equations, subsampling rows to reduce the dimension of the computation matrix in each iteration. Stochastic iterative methods involve sampling a sparse approximation of matrix G to reduce the computation in each iteration while maintaining convergence. It may be of further interest to combine the method presented in this work with some of these approaches to obtain local algorithms for other settings.

1.3 Transforming $Ax = b$ to $x = Gx + z$

Given a system of linear equations of the form $Ax = b$, there are many methods for choosing G and z such that the equation is equivalent to the form given by $x = Gx + z$ with $\rho(G) < 1$ [18]. The Jacobi and Richardson methods are particularly suitable for our setting because they have additional properties that G is as sparse as A, and G_{ij} can be computed as a simple function of A_{ij} and A_{ii}. Given (A, b), there are potentially infinitely many ways to choose (G, z) to satisfy the condition that $\rho(G) < 1$. Finding the optimal choice[7] of (G, z) given (A, b) is beyond the scope of this paper.

Corollary 1.1. *If A is positive definite[8] or diagonally dominant, then we can use standard methods (e.g. Jacobi or Richardson), to choose a matrix G and vector z such that $\rho(G) < 1$, and the solution x which satisfies $x = Gx + z$ will also satisfy $Ax = b$.*

The Jacobi method chooses $G = -D^{-1}(A - D)$ and $z = D^{-1}b$, where D is a diagonal matrix such that $D_{uu} = A_{uu}$. The Richardson method chooses $G = I - \gamma A$ and $z = \gamma b$ for any γ such that $0 \leq \gamma \leq \min_{\|x\|_2=1}(2x^T Ax)/(x^T A^T Ax)$. If A is symmetric, then using the Richardson method with an optimal choice of γ results in a choice of G such that $\rho(G) = \|G\|_2 = (\kappa(A) - 1)/(\kappa(A) + 1)$.

2 Synchronous Local Algorithm

This method is a natural extension of the existing literature. Since we are only interested in computing the i^{th} component, we can compute $\sum_{k=0}^{\infty}(G^T)^k e_i$ and take the inner product with z at the end, instead of computing $\sum_{k=0}^{\infty} G^k z$, where z could be a dense vector. This small modification allows us to control the sparsity of intermediate vectors involved in the algorithm. We proceed to

[7] By optimal, we would like to minimize $\rho(G)$, which maximizes the convergence rate of the algorithm.

[8] Matrix A is positive definite, if for all nonzero x, $x^T Ax > 0$.

describe our algorithm. First, observe that x_i can be expressed according to (1) as

$$x_i = e_i^T \sum_{k=0}^{\infty} G^k z = \sum_{k=0}^{t-1} e_i^T G^k z + e_i^T \sum_{k=t}^{\infty} G^k z = \sum_{k=0}^{t-1} e_i^T G^k z + e_i^T G^t x$$

The algorithm keeps track of two intermediate vectors such that at iteration t, $p^{(t)} = (\sum_{k=0}^{t-1} e_i^T G^k)^T$ and $r^{(t)} = (e_i^T G^t)^T$. Then it follows that for all iterations t, $x_i = p^{(t)T} z + r^{(t)T} x$. To achieve this, we initialize the vectors with $p^{(0)} = 0$ and $r^{(0)} = e_i$, and we use the following updates:

$$p^{(t+1)} = p^{(t)} + r^{(t)}, \tag{2}$$

$$r^{(t+1)} = G^T r^{(t)}. \tag{3}$$

The algorithm computes the estimate according to $\hat{x}_i = p^{(t)T} z$. Thus the error will be exactly $x_i - \hat{x}_i = r^{(t)T} x$. We refer to $r^{(t)}$ as the residual vector.

```
Algorithm:  SynchronousLocalAlgorithm
$(G, z, i, \epsilon)$
Input: $G, i, \epsilon$
$p^{(0)} = 0, r^{(0)} = e_i, t = 0$
while $\|r^{(t)}\|_2 > \epsilon$ do
    $p^{(t+1)} = p^{(t)} + r^{(t)}$
    $r^{(t+1)} = G^T r^{(t)}$
    $t = t + 1$
end while
return  $\hat{x}_i = p^{(t)T} z$
```

The sparsity of $r^{(k)} = (G^T)^k e_i$ is upper bounded by the size of the local neighborhood $N_i(k)$, which in turn is upper bounded by d^k, and it follows that the sparsity of $p^{(k)}$ is also limited to $N_i(k)$. The number of multiplications that iteration k of our algorithm requires is bounded by $d|N_i(k)|$. Our algorithm keeps the computation local, and thus saves computation.

2.1 Convergence Results

Theorem 2.1. *If $\rho(G) < 1$,* SynchronousLocalAlgorithm *converges and produces an estimate $\hat{x}_i$ such that $|\hat{x}_i - x_i| \leq \epsilon \|x\|_2$. If $\|G\|_2 < 1$, the total number of multiplications is bounded by*

$$O\left(\min\left(d\epsilon^{\ln(d)/\ln(\|G\|_2)}, \frac{dn\ln(\epsilon)}{\ln(\|G\|_2)}\right)\right).$$

The error at iteration k is given by $x^T r^{(k)} = x^T (G^T)^k e_i \leq \|G\|_2^k \|x\|_2$. It follows that the algorithm converges when $\rho(G) < 1$, since $G^k \to 0$. The algorithm terminates within at most $\ln(\epsilon)/\ln(\|G\|_2)$ iterations, and outputs an estimate such that $|\hat{x}_i - x_i| \leq \epsilon \|x\|_2$. The number of multiplications in iteration k can be bounded by $\min(d\|r^{(k)}\|_0, dn) \leq \min(d^{k+1}, dn)$. The right hand expression in the theorem comes from bounding the cost of each iteration by dn, which holds even in the nonsparse setting. This bound obtains the same result as standard linear iterative methods. The left hand expression in the theorem comes from bounding the cost of iteration k by

d^{k+1}. If G is sparse, then the left hand bound may be tighter. When G is sufficiently sparse and its spectral properties behave well, i.e., $d = O(1)$ and $-1/\ln(\|G\|_2) = O(1)$, then we achieve an approximation for x_i in constant time with respect to the size of the matrix for large n.

When G is nonsymmetric, $\rho(G) < 1$, yet $\|G\|_2 \geq 1$, the algorithm still converges asymptotically, though our rate bounds no longer hold. We suspect that in this case, a similar convergence rate holds as a function of $\rho(G)$, due to Gelfand's spectral radius formula, which states that $\rho(M) = \lim_{k\to\infty} \|M^k\|_2^{1/k}$. The precise rate depends on the convergence of this limit.

2.2 Local Convergence Properties

We can visualize the algorithm in terms of computation over $\mathcal{G}(G)$. Multiplying $r^{(t)}$ by G^T corresponds to a message passing operation from each of the nonzero coordinates of $r^{(t)}$ along their adjacent edges in the graph. The sparsity of $r^{(t)}$ grows as a breadth first traversal over the graph starting from node i. It follows that $\|r^{(t)}\|_0 \leq |N_i(t)|$, which is the number of nodes involved in the computation up to time t. The termination condition guarantees that the algorithm only involves nodes that are within distance $\ln(\epsilon)/\ln(\|G\|_2)$ from the node i. We define the matrix $G_{N_i(t)}$ such that $G_{N_i(t)}(a, b) = G(a, b)$ if $(a, b) \in N_i(t) \times N_i(t)$, and is zero otherwise. It follows that

$$\|r^{(t)}\|_2 = \|e_i^T G^t\|_2 = \|e_i^T \prod_{k=1}^{t} G_{N_i(k)}\|_2 = \|e_i^T G_{N_i(t)}^t\|_2 \leq \|G_{N_i(t)}\|_2^t. \tag{4}$$

It is possible that for some choices of i and t, $\|G_{N_i(t)}\|_2 < \|G\|_2$, in which case the algorithm would converge more quickly as a function of the local neighborhood. If G corresponds to a scaled adjacency matrix of an unweighted undirected graph, then it is known that

$$\max\left(d_{\text{average}}, \sqrt{d_{\max}}\right) \leq \rho(G) \leq d_{\max}. \tag{5}$$

In this case, we would only expect $\|G_{N_i(t)}\|_2$ to be smaller than $\|G\|_2$ if the local degree distribution of the neighborhood around node i is different from the global degree distribution.

3 Asynchronous Randomized Local Algorithm

The synchronous algorithm requires every nonzero coordinate to perform one message passing operation or update in each iteration, i.e., every coordinate $j \in N_i(t)$ updates at iteration t. In this section, we modify the algorithm to only apply the update to a single coordinate at a time, which corresponds to multiplying by only a single row of G, rather than the full matrix. This row is adaptively sampled among the nonzero coordinates of the current iterate, therefore the nodes in graph $\mathcal{G}(G)$ which correspond to nonzero coordinates of the current iterate are all connected by a path to node i. Since the coordinates can be updated in arbitrary order, the sparsity pattern of the iterates over time no longer corresponds to a breadth first traversal over $\mathcal{G}(G)$, but instead resembles a traversal which grows by adding adjacent nodes in any order beginning from node i. The computation involved in each iteration corresponds to a local operation involving only the node corresponding to the chosen row, and its neighboring edges. We will show that the algorithm converges even when the updates for different coordinates are performed in any order and with different rates of updates.

We rewrite the update equations of the synchronous algorithm to show the incremental updates that each coordinate of $r^{(t)}$ is involved in:

$$p^{(t+1)} = p^{(t)} + \sum_u e_u e_u^T r^{(t)}, \tag{6}$$

$$r^{(t+1)} = r^{(t)} + \sum_u (G^T - I) e_u e_u^T r^{(t)} \tag{7}$$

The asynchronous algorithm chooses a single coordinate u, and applies the updates corresponding to the calculations involving the u^{th} coordinate of $r^{(t)}$:

$$p^{(t+1)} = p^{(t)} + e_u e_u^T r^{(t)}, \tag{8}$$

$$r^{(t+1)} = r^{(t)} + (G^T - I) e_u e_u^T r^{(t)}. \tag{9}$$

The algorithm samples the coordinate u according to the following distribution:[9]

$$\text{For all } u, \quad \mathcal{P}(u) = \frac{\mathbb{I}\left(r_u^{(t)} \neq 0\right)}{\|r^{(t)}\|_0}. \tag{10}$$

To prove that the estimate converges asymptotically to the correct solution, we establish an invariant that for all t, $x_i = p^{(t)T} z + r^{(t)T} x$. This holds for all possible selections of update coordinates. Since the distribution $\mathcal{P}$ chooses uniformly among the nonzero coordinates of $r^{(t)}$, in expectation the update step corresponds to multiplying a scaled version of matrix G to vector $r^{(t)}$. We can prove that in addition $\|r^{(t)}\|_2$ contracts with high probability due to the choice of distribution $\mathcal{P}$.

```
Algorithm: AsynchronousLocalAlgorithm($G, z, i, \epsilon$)
Input: $G, i, \epsilon$
$p^{(0)} = 0, r^{(0)} = e_i, t = 0$
while $\|r^{(t)}\|_2 > \epsilon$ do
    Pick a coordinate $u$ with probability $\mathcal{P}(u)$
    $p^{(t+1)} = p^{(t)} + e_u e_u^T r^{(t)}$
    $r^{(t+1)} = r^{(t)} - (I - G^T) e_u e_u^T r^{(t)}$
    $t = t + 1$
end while
return $\hat{x}_i = p^{(t)T} z$
```

3.1 Convergence Results

Lemma 3.1 (Invariant). *The variables in the* AsynchronousLocalAlgorithm(G, z, i, ϵ) *satisfy the invariant that for all* t, $x_i = p^{(t)T} z + r^{(t)T} x$.

We prove that the invariant holds by induction. It follows that when the algorithm terminates, $|\hat{x}_i - x_i| = |r^{(t)T} x| \leq \epsilon \|x\|_2$. We can show that the algorithm converges to x_i as long as each coordinate is chosen infinitely often, regardless of the sequence in which the updates are performed.

[9] Let $\mathbb{I}(\cdot)$ denote the indicator function. For simpler notation, we suppress the dependence of $\mathcal{P}$ on $r^{(t)}$.

Theorem 3.2. *If $\rho(|G|) < 1$,[10] the estimate $p^{(t)T} z$ from the* AsynchronousLocalAlgorithm *converges to x_i for any sequence of updates as long as each coordinate is updated infinitely often.*

The solution x_i can be expressed as a weighted sum over all walks over the graph $\mathcal{G}(G)$ beginning at node i, where a walk beginning at node i and ending at node j has weight $\prod_{e \in \text{walk}} G_e z_j$. The updates ensure that we never double count a walk, and all uncounted walks are included in $r^{(t)}$. For any l, there is a finite time S_l after which all random walks of length less than or equal to l have been counted and included into $p^{(S_l)}$. This allows us to upper bound $|x_i - p^{(S_l)T} x|$ as a function of $|G|^{l+1}$, which converges to zero when $\rho(|G|) < 1$. If $\rho(|G|) \geq 1$, then the original Neumann series stated in (1) is only conditionally convergent. By the Reimann series theorem, the terms can be rearranged in such a way that the new series diverges, and rearranging the terms in the series corresponds to updating the coordinates in different orders, e.g., depth first as opposed to breadth first. This theorem and proof can be extended to show that the algorithm converges asymptotically even given communication delays, as long as the messages reach their destination in finite time.

The rate of convergence depends on the probability distribution for choosing the coordinate to update. We provide an analysis when the nonzero-valued coordinates are chosen with equal probability, as in the distribution $\mathcal{P}$. With high probability, the number of multiplications the asynchronous algorithm uses is bounded by a similar expression as the bound given for the synchronous algorithm.

Theorem 3.3. *If $\|G\|_2 < 1$, with probability 1, the* AsynchronousLocalAlgorithm *which updates coordinates uniformly according to $\mathcal{P}$ terminates and produces an estimate $\hat{x}_i$ such that $|\hat{x}_i - x_i| \leq \epsilon \|x\|_2$. With probability greater than $1 - \delta$, the total number of multiplications is bounded by*

$$O\left(\min\left(d\left(\epsilon\sqrt{\delta/2}\right)^{-d/(1-\|G\|_2)}, \frac{-dn\ln(\epsilon\sqrt{\delta})}{1 - \|G\|_2}\right)\right).$$

We can show that

$$\mathbb{E}_{\mathcal{P}}\left[r^{(t+1)} \middle| r^{(t)}\right] = \left(I - \left(\frac{I - G^T}{\|r^{(t)}\|_0}\right)\right) r^{(t)}. \tag{11}$$

Thus, in expectation, the error contracts in each iteration by at least $(1 - (1 - \|G\|_2)/\min(td, n))$. We use this to prove an upper bound on the expected L2-norm of the residual vector $r^{(t)}$, and we apply Markov's inequality to prove that the algorithm terminates with high probability within a certain number of multiplications. There is additional technical detail in the formal proof, as it needs to handle the fact the $\|r^{(t)}\|_0$ is dependent on the full history of the previous iterations. It first analyzes the algorithm for a modified distribution, where the scaling factor grows deterministically according to $\min(td, n)$ as opposed to $\|r^{(t)}\|_0$, and it uses a coupling argument to show that the upper bound on the termination time of the algorithm using the modified distribution translates to the original distribution $\mathcal{P}$. This establishes an upper bound on the Lyapunov exponent for a product of random matrices drawn from a time-dependent distribution.

By observing that $1 - \|G\|_2 \approx -\ln(\|G\|_2)$ when $\|G\|_2 \approx 1$, we can compare Theorems 3.3 and 2.1. The right hand expression again shows that our method is bounded by the cost of standard linear iterative methods. The left hand bound provides a local analysis utilizing the sparsity of G, showing that the computation remains local to a neighborhood of i when the graph is sufficiently

[10] Let $|G|$ denote the matrix whose $(i,j)^{\text{th}}$ entry is $|G_{ij}|$.

sparse. This bound grows exponentially in d, while the corresponding bound in Theorem 2.1 grows only polynomially in d. The rate of convergence of the asynchronous variant is slower by a factor of $d/\ln(d)$ because the provable contraction of the error in each iteration is now spread out among the nonzero coordinates of the current iterate. The different rates come from comparing the convergence of the error along with the number of nonzero entries in the iterates over time.

3.2 Asynchronous Implementation

We can implement this algorithm in a fully asynchronous way by using an asynchronous computation model which assumes each node is equipped with a Poisson clock with rate $\lambda_u(t)$ that indicates when the node is to perform computation, similar to classic gossip algorithms. We eliminate coordinate costs as long as the probability distribution $\mathcal{P}(u)$ is proportional to some function $f(u, p_u, r_u)$ which only depends on information which is local to coordinate u. The parameter of the Poisson clocks is chosen such that $\lambda_u(t) \propto f(u, p_u^{(t)}, r_u^{(t)})$. When the clock ticks, the node performs its corresponding update according to lines 4-5 in the algorithm code given above. This update is equivalent to updating $p_u = p_u + r_u$, $r_u = G_{uu} r_u$, and sending a message of $G_{uw} r_u$ to each of its neighbors w. Then node u updates the parameter of its Poisson clock, computed according to the new values of p_u and r_u. When its neighbor node w receives the message, it updates $r_w = r_w + G_{uw} r_u$, and updates the parameter of its Poisson clock according to the new value of r_w.

If there is no communication delay, then this correctly implements the asynchronous algorithm. This relies upon the memoryless property of Poisson processes, as well as the property that the probability of the Poisson clock of node u ticking the earliest amongst all other Poisson clocks is equal to $\lambda_u / \sum_{w=1}^{n} \lambda_w$, thus matching our desired probability $\mathcal{P}(u)$. If there is communication delay such that the messages take time to reach the destination node, then a more involved analysis is required to obtain bounds on the convergence rate. However, since the invariant still holds (when we take into account the residual weight which is in transit between nodes), the algorithm still converges to the solution eventually as long as each coordinate updates infinitely often and every message eventually reaches its destination in finite time.

3.3 Choosing Sampling Distribution $\mathcal{P}$

Our convergence rate bounds only apply when the algorithm samples uniformly among nonzero coordinates of the residual vector $r^{(t)}$, according to the distribution $\mathcal{P}$ defined by 10. However, this may not be the distribution which optimizes the convergence rate. Choosing a distribution which is not uniform amongst the nonzero coordinates is analogous to multiplying the residual vector by a reweighted matrix $\tilde{G}$ in expectation, where each row of $\tilde{G}$ corresponds to a row of G weighted by the probability of choosing that row in the new distribution. We provide simulations which illustrate the different convergence rates of choosing different sampling distributions $\mathcal{P}$.

Figure 1 shows results from computing Bonacich centrality of a node in a Erdos-Renyi graph with 1000 nodes and edge probability 0.0276, which is chosen arbitrarily to guarantee that the graph is sparse yet connected. We compare the standard linear iterative method, our synchronous algorithm, and the asynchronous algorithm with four different choices of the sampling distribution $\mathcal{P}$, where the probability of of choosing coordinate u is proportional to $\mathbb{I}(r_u \neq 0)$, $|r_u|$, r_u^2, or chosen as $\text{argmax}_u |r_u|$. Our simulations indicate that choosing coordinates with large values of $|r_u^{(t)}|$ seems to improve the convergence of the algorithm, in fact the algorithm which always chooses the largest coordinate to update seems to perform the best. This is consistent with our intuition, as the

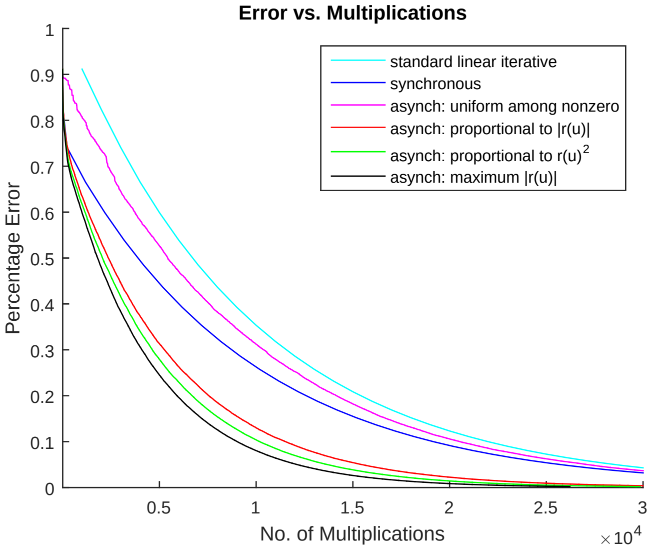

Figure 1: Comparison of convergence rate for algorithm with different choices of $\mathcal{P}$.

residual vector $r^{(t)}$ is directly related to the error in the estimation. By updating coordinates with large values of $r_u^{(t)}$, we hope to make more progress in reducing the error. Establishing theoretical analysis of this observation remains a challenging problem, as the sampling distribution in iteration t depends in a complex manner upon the full sample path of updates up to iteration t, as opposed to a simple scaling of the matrix as in uniform sampling. It is not obvious how to establish bounds for a product of random matrices drawn from a complex path dependent distribution.

4 Future Directions

It is an open problem to investigate the optimal choice of the probability distribution for sampling coordinates used within the asynchronous method, as simulations indicate that some distributions may lead to faster convergence rates than others. This is related to recent work which investigates weighted sampling for the randomized Kaczmarz method, although their methods depend on weighting according to the matrix G rather than the values in the intermediate vectors of the algorithm.

We hope that our work will initiate studies of sparsity-preserving iterative methods in asynchronous and distributed settings from an algorithmic perspective. Our methods could potentially be used as subroutines in other methods in which there is a need for local or asynchronous matrix computation. This is an initial study of local iterative methods for matrix computation, and thus it is unclear whether our algorithm achieves an optimal convergence rate. There is a wealth of literature which studies acceleration or preconditioning techniques for classic linear system solvers, and it would be interesting to see whether these acceleration techniques could be used to speed up the convergence of our local algorithm as a function of the spectral properties of the matrix.

References

[1] Reid Andersen, Christian Borgs, Jennifer Chayes, John Hopcraft, Vahab S. Mirrokni, and Shang-Hua Teng. Local computation of PageRank contributions. In *Algorithms and Models for the Web-Graph*, pages 150–165. Springer, 2007.

[2] J.H. Curtiss. *A theoretical comparison of the efficiencies of two classical methods and a monte carlo method for computing one component of the solution of a set of linear algebraic equations.* Courant Institute of Mathematical Sciences, New York University, 1954.

[3] George E. Forsythe and Richard A. Leibler. Matrix inversion by a monte carlo method. *Mathematical Tables and Other Aids to Computation*, pages 127–129, 1950.

[4] Gene H. Golub and Charles F. Van Loan. *Matrix computations / Gene H. Golub, Charles F. Van Loan.* Johns Hopkins studies in the mathematical sciences. Baltimore : The Johns Hopkins University Press, 2013., 2013.

[5] John H. Halton. A retrospective and prospective survey of the monte carlo method. *Siam review*, 12(1):1–63, 1970.

[6] John H. Halton. Sequential monte carlo techniques for the solution of linear systems. *Journal of Scientific Computing*, 9(2):213–257, 1994.

[7] Hao Ji, Michael Mascagni, and Yaohang Li. Convergence analysis of markov chain monte carlo linear solvers using ulam–von neumann algorithm. *SIAM Journal on Numerical Analysis*, 51(4):2107–2122, 2013.

[8] Jonathan A. Kelner, Lorenzo Orecchia, Aaron Sidford, and Zeyuan Allen Zhu. A simple, combinatorial algorithm for solving sdd systems in nearly-linear time. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 911–920. ACM, 2013.

[9] Ioannis Koutis, Gary L. Miller, and Richard Peng. A nearly-m log n time solver for sdd linear systems. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 590–598. IEEE, 2011.

[10] K. Sabelfeld and N. Mozartova. Sparsified randomization algorithms for large systems of linear equations and a new version of the random walk on boundary method. *Monte Carlo Methods and Applications*, 15(3):257–284, 2009.

[11] Karl Sabelfeld. Stochastic algorithms in linear algebra-beyond the markov chains and von neumann-ulam scheme. In *Numerical Methods and Applications*, pages 14–28. Springer, 2011.

[12] Karl Sabelfeld and Nadja Loshchina. Stochastic iterative projection methods for large linear systems. *Monte Carlo Methods and Applications*, 16(3-4):343–359, 2010.

[13] Devavrat Shah. Gossip algorithms. *Foundations and Trends in Networking*, 3(1):1–125, 2008.

[14] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *arXiv preprint cs/0607105*, 2006.

[15] Thomas Strohmer and Roman Vershynin. A randomized kaczmarz algorithm with exponential convergence. *Journal of Fourier Analysis and Applications*, 15(2):262–278, 2009.

[16] Mengdi Wang and Dimitri P. Bertsekas. Stabilization of stochastic iterative methods for singular and nearly singular linear systems. *Mathematics of Operations Research*, 39(1):1–30, 2013.

[17] W.R. Wasow. A note on the inversion of matrices by random walks. *Mathematical Tables and Other Aids to Computation*, pages 78–81, 1952.

[18] Joan R. Westlake. *A handbook of numerical matrix inversion and solution of linear equations*, volume 767. Wiley New York, 1968.

A Proof of Theorem 2.1

The initial vectors and update rules are chosen such that $p^{(t)} = (\sum_{k=0}^{t-1} G^k)^T e_i$, and $r^{(t)} = (G^t)^T e_i$. Therefore for all t, $x_i = z^T p^{(t)} + x^T r^{(t)}$, and the error in the estimate is given by $x^T r^{(t)}$. Since the algorithm terminates when $\|r^{(t)}\|_2 \leq \epsilon$, then it follows that

$$|\hat{x}_i - x_i| = |r^{(t)T} x| \leq \|r^{(t)}\|_2 \|x\|_2 \leq \epsilon \|x\|_2. \tag{12}$$

In order to upper bound the computation, we observe that

$$\|r^{(t)}\|_2 = \|(G^T)^t e_i\|_2 \leq \|G\|_2^t. \tag{13}$$

Therefore the algorithm terminates with $\|r^{(t)}\|_2 < \epsilon$ within at most $\ln(\epsilon)/\ln(\|G\|_2)$ iterations. Since each row of G has at most d nonzero entries, $\|r^{(t)}\|_0 \leq d^t$. The number of multiplications in each iteration is at most $d\|r^{(t)}\|_0$. Therefore, we can upper bound the total number of multiplications by

$$\begin{aligned} d\left(\sum_{t=0}^{\lfloor \ln(\epsilon)/\ln(\|G\|_2)\rfloor} d^t\right) &= \left(\frac{d}{d-1}\right)\left(d^{\lfloor \ln(\epsilon)/\ln(\|G\|_2)\rfloor + 1} - 1\right) \\ &= O\left(d \cdot d^{\ln(\epsilon)/\ln(\|G\|_2)}\right) = O\left(d\epsilon^{\ln(d)/\ln(\|G\|_2)}\right). \end{aligned} \tag{14}$$

Alternatively, we can naively bound the number of multiplications in each iteration by dn. Then it follows that the total number of multiplications is also bounded by $O(dn\ln(\epsilon)/\ln(\|G\|_2))$.

B Proof of Lemma 3.1

Recall that $x = z + Gx$. We prove that the invariant holds by using induction. First verify that for $t = 0$, $p^{(0)T} z + r^{(0)T} x = 0 + e_i^T x = x_i$. Then assuming that $x_i = p^{(t)T} z + r^{(t)T} x$, we show that

$$\begin{aligned} p^{(t+1)T} z + r^{(t+1)T} x &= p^{(t)T} z + (e_u e_u^T r^{(t)})^T z + r^{(t)T} x - (e_u e_u^T r^{(t)})^T x + (G^T e_u e_u^T r^{(t)})^T x \\ &= p^{(t)T} z + (e_u e_u^T r^{(t)})^T z + r^{(t)T} x - (e_u e_u^T r^{(t)})^T (z + Gx) + (G^T e_u e_u^T r^{(t)})^T x \\ &= p^{(t)T} z + r^{(t)T} x = x_i. \end{aligned}$$

C Proof of Theorem 3.2

Recall that the solution can be viewed as a weighted sum over all walks over the graph $\mathcal{G}(G)$ beginning at node i, where a walk beginning at node i and ending at node j has weight $\prod_{e \in \text{walk}} G_e z_j$. Let $\mathcal{W}(i)$ denote the set of all walks beginning from node i, then

$$x_i = e_i^T \sum_{k=0}^{\infty} G^k z = \sum_{W \in \mathcal{W}(i)} \prod_{e \in W} G_e z_{\text{endpoint}(W)}.$$

Observe that due to the form of the weights over each walk, we can express the weighted sum of all walks whose first l nodes are given by $(v_1, v_2, \dots v_l)$ with $\left(\prod_{k=1}^{l-1} G_{v_k v_{k+1}}\right) x_{v_l}$, since x_{v_l} captures the sum and weights of the remaining unfinished walks. In other words, the weighted sum of walks with a certain prefix is equal to the product of the weights from the prefix portion of the walk multiplied with the sum of the weights for all walks that could continue from the endpoint of the prefix walk. The value of the residual vector $r_u^{(t)}$ contains the product of the weights along the prefix portion of the walks that end at node u, thus explaining why $r^{(t)T} x$ is the weight of all yet uncounted walks.

We use this perspective to interpret the single coordinate updates in the algorithm to argue that there is conservation of computation, i.e., we never double count a walk, and all uncounted walks are included in the residual mass vector $r^{(t)}$. It follows from the above discussion that $x_u = z_u + \sum_w G_{uw} x_w$, which motivates the update in our algorithm, as the first term z_u captures the walks that end at node u, and each term within the summation $G_{uw} x_w$ counts the walks which take the next edge (u, w). Each time that a nonzero-valued coordinate is updated, the update $p_u^{(t+1)} = p_u^{(t)} + r_u^{(t)}$ corresponds to counting the walks which end at node u, and whose weight is contained within $r_u^{(t)}$. We also need to count the walks which have the same prefix, but do not yet terminate at node u, which is captured by multiplying $r_u^{(t)}$ by each of its adjacent nodes G_{uw} and adding that to $r_w^{(t+1)}$. We proceed to argue that for any r, there is a finite time t after which all random walks of length less than or equal to l have been counted and included into the estimate vector $p^{(t)}$.

Given a coordinate update sequence $(u_0, u_1, u_2, u_3, \dots u_t \dots)$, we require that each coordinate appears infinitely often. We can assume that $u_0 = i$, since at iteration 0 all the mass is at node i, thus it is the only node to update. Let S_1 denote the earliest time after which all of the neighbors of node i have been updated at least once:

$$S_1 = \min\{t \geq 1 : N_i(1) \subset \{u_0, u_1, \dots u_{t-1}\}\}.$$

This guarantees that $p^{(S_1)}$ includes the weights from all the length one walks from node i. We proceed to let S_2 denote the earliest time which all nodes within a 2-neighborhood of node i have updated once after time S_1:

$$S_2 = \min\{t \geq S_1 : N_i(2) \subset \{u_{S_1}, u_{S_1+1}, \dots u_{t-1}\}\}.$$

This now guarantees that $p^{(S_2)}$ includes the weights from all the length one and two walks from node i. We can iteratively define S_r as the time after which the estimate vector has counted and included all walks of length up to r:

$$S_l = \min\{t \geq S_{l-1} : N_i(l) \subset \{u_{S_{l-1}}, u_{S_{l-1}+1}, \dots u_{t-1}\}\}.$$

Since each coordinate appears infinitely often in the sequence, S_l is well defined and finite for all l.

Finally we upper bound the error by using a loose upper bound on the weights of all walks with length larger than l. By the invariant, it follows that $x_i = p^{(S_l)T} z + r^{(S_l)T} x$, which is the sum of all counted or included walks in $p^{(S_l)T} z$ and the remaining weight of uncounted walks in $r^{(S_l)T} x$. The weighted sum of all walks of length at most l from node i is expressed by $z^T \sum_{k=0}^{l} (G^T)^k e_i$. Thus the error, or weight of uncounted walks, must be bounded by the corresponding weighted sum of the absolute values of the walks of length larger than l:

$$-z^T \sum_{k=l+1}^{\infty} (|G|^T)^k e_i \leq r^{(S_l)T} x \leq z^T \sum_{k=l+1}^{\infty} (|G|^T)^k e_i.$$

It follows that

$$\left| x_i - p^{(S_l)T} z \right| \leq z^T (|G|^T)^{l+1} \sum_{k=0}^{\infty} (|G|^T)^k e_i,$$

which converges to zero as l goes to infinity as long as $\rho(|G|) < 1$.

D Proof of Theorem 3.3

Since the algorithm terminates when $\|r^{(t)}\|_2 \leq \epsilon$, it follows from Lemma 3.1 that $|\hat{x}_i - x_i| = |r^{(t)T} x| \leq \epsilon \|x\|_2$. Recall that the algorithm chooses a coordinate in each iteration according to the distribution $\mathcal{P}$, as specified in (10). To simplify the analysis, we introduce another probability distribution $\tilde{\mathcal{P}}$, which has a fixed size support of $\min(td, n)$ rather than $\|r^{(t)}\|_0$. We first analyze the convergence of a modified algorithm which samples coordinates according to $\tilde{\mathcal{P}}$. Then we translate the results back to the original algorithm.

Observe that for any $t \in \mathbb{Z}_+$, there exists a function $S_t : \mathbb{R}^n \to \{0,1\}^n$, which satisfies the properties that for any $v \in \mathbb{R}^n$ and $u = S_t(v)$, if $v_i \neq 0$, then $u_i = 1$, and if $\|v\|_0 \leq td$, then $\|u\|_0 = \min(td, n)$. In words, $S_t(v)$ is a function which takes a vector of sparsity at most td, and maps it to a binary valued vector which preserves the sparsity pattern of v, yet adds extra entries of 1 in order that the sparsity of the output is exactly $\min(td, n)$. We define the distribution $\tilde{\mathcal{P}}$ to choose uniformly at random among the nonzero coordinates of $S_t\left(r^{(t)}\right)$, as follows:[11]

$$\tilde{\mathcal{P}}(u) = \frac{e_u^T S_t\left(r^{(t)}\right)}{\min(td, n)}. \tag{15}$$

This is a valid probability distribution since for all t, $\|r^{(t)}\|_0 \leq td$. We first analyze the asynchronous algorithm which samples coordinates accoridng to $\tilde{\mathcal{P}}$. Lemma D.1 shows that in expectation, the error contracts in each iteration by $(1-(1-\|G\|_2)/\min(td, n))$. Lemma D.2 provides an upper bound on the expected L2-norm of the residual vector $r^{(t)}$. Then we apply Markov's inequality to prove that the algorithm terminates with high probability within a certain number of multiplications.

In order to extend the proofs from $\tilde{\mathcal{P}}$ to $\mathcal{P}$, we define a coupling between two implementations of the algorithm, one which sample coordinates according to $\tilde{\mathcal{P}}$, and the other which samples coordinates according to $\mathcal{P}$. We prove that in this joint probability space, the implementation which uses distribution $\mathcal{P}$ always terminates in number of iterations less than or equal to the

[11] For simpler notation, we suppress the dependence of $\tilde{\mathcal{P}}$ on t and $r^{(t)}$.

corresponding termination time of the implementation using $\tilde{\mathcal{P}}$. Therefore, computing an upper bound on the number of multiplications required under $\tilde{\mathcal{P}}$ is also an upper bound for the algorithm which uses $\mathcal{P}$.

Lemma D.1. *If $\|G\|_2 < 1$, for all t,*

(a) $\mathbb{E}_{\tilde{\mathcal{P}}}\left[r^{(t+1)}\middle| r^{(t)}\right] = \left(I - \left(\frac{I - G^T}{\min(td, n)}\right)\right) r^{(t)}$,

(b) $\left\|\mathbb{E}_{\tilde{\mathcal{P}}}\left[r^{(t)}\right]\right\|_2 \leq \min\left(t^{-(1-\|G\|_2)/d}, e^{-(t-1)(1-\|G\|_2)/n}\right)$.

Proof. We will use induction to get an expression for $\mathbb{E}_{\tilde{\mathcal{P}}}\left[r^{(t)}\right]$. Recall that $r^{(0)} = e_i$. Since there is only a single coordinate to choose from, $r^{(1)}$ is also predetermined, and is given by $r^{(1)} = G^T e_i$.

$$\mathbb{E}_{\tilde{\mathcal{P}}}\left[r^{(t+1)}\middle| r^{(t)}\right] = r^{(t)} - (I - G^T)\sum_u \tilde{\mathcal{P}}(u) e_u r_u^{(t)}. \tag{16}$$

By design of $\tilde{\mathcal{P}}$, we know that $\tilde{\mathcal{P}}(u) = 1/\min(td, n)$ for all u such that $r_u^{(t)} \neq 0$. Therefore,

$$\sum_u \tilde{\mathcal{P}}(u) e_u r_u^{(t)} = \frac{r_u^{(t)}}{\min(td, n)}. \tag{17}$$

We substitute this into (16) to show that

$$\mathbb{E}_{\tilde{\mathcal{P}}}\left[r^{(t+1)}\middle| r^{(t)}\right] = \left(I - \left(\frac{I - G^T}{\min(td, n)}\right)\right) r^{(t)}. \tag{18}$$

Using the initial conditons $r^{(1)} = G^T e_i$ and the law of iterated expectation, it follows that

$$\mathbb{E}_{\tilde{\mathcal{P}}}\left[r^{(t)T}\right] = e_i^T G \prod_{k=1}^{t-1}\left(I - \left(\frac{I - G}{\min(kd, n)}\right)\right). \tag{19}$$

Therefore,

$$\begin{aligned}
\left\|\mathbb{E}_{\tilde{\mathcal{P}}}\left[r^{(t)}\right]\right\|_2 &\leq \|G\|_2 \prod_{k=1}^{t-1}\left(1 - \left(\frac{1 - \|G\|_2}{\min(kd, n)}\right)\right), \\
&\leq \|G\|_2 \exp\left(-\sum_{k=1}^{t-1}\frac{1 - \|G\|_2}{\min(kd, n)}\right), \\
&\leq \|G\|_2 \min\left(\exp\left(-\sum_{k=1}^{t-1}\frac{1 - \|G\|_2}{kd}\right), \|G\|_2 \exp\left(-\sum_{k=1}^{t-1}\frac{1 - \|G\|_2}{n}\right)\right).
\end{aligned} \tag{20}$$

Since $\|G\|_2 < 1$ by assumption, and using the property that $\sum_{k=1}^{t-1}\frac{1}{k} > \ln(t)$, it follows that

$$\left\|\mathbb{E}_{\tilde{\mathcal{P}}}\left[r^{(t)}\right]\right\|_2 \leq \min\left(t^{-(1-\|G\|_2)/d}, e^{-(t-1)(1-\|G\|_2)/n}\right). \tag{21}$$

□

Lemma D.2. *If $\|G\|_2 < 1$, $d \geq 4$, and $n \geq 8$,*

$$\mathbb{E}_{\tilde{\mathcal{P}}}\left[\left\|r^t\right\|_2^2\right] \leq \min\left(2t^{-2(1-\|G\|_2)/d}, 4e^{-2(t-1)(1-\|G\|_2)/n}\right).$$

Proof. Observe that

$$\mathbb{E}_{\tilde{\mathcal{P}}}\left[\left\|r^{(t+1)}\right\|_2^2\right] = \mathbb{E}_{\tilde{\mathcal{P}}}\left[\left\|r^{(t+1)} - \mathbb{E}_{\tilde{\mathcal{P}}}\left[r^{(t+1)}\right]\right\|_2^2\right] + \left\|\mathbb{E}_{\tilde{\mathcal{P}}}\left[r^{(t+1)}\right]\right\|_2^2, \tag{22}$$

and

$$\mathbb{E}_{\tilde{\mathcal{P}}}\left[\left\|r^{(t+1)} - \mathbb{E}_{\tilde{\mathcal{P}}}\left[r^{(t+1)}\right]\right\|_2^2 \middle| r^{(t)}\right] = \mathbb{E}_{\tilde{\mathcal{P}}}\left[\left\|r^{(t+1)}\right\|_2^2 \middle| r^{(t)}\right] - \left\|\mathbb{E}_{\tilde{\mathcal{P}}}\left[r^{(t+1)}\middle| r^{(t)}\right]\right\|_2^2. \tag{23}$$

Based on the update equation $r^{(t+1)} = r^{(t)} - (I - G^T)e_u r_u^{(t)}$, we can compute that

$$\begin{aligned}
\mathbb{E}_{\tilde{\mathcal{P}}}\left[\left\|r^{(t+1)}\right\|_2^2 \middle| r^{(t)}\right] &= \sum_u \tilde{\mathcal{P}}(u)(r^{(t)} - (I - G^T)e_u r_u^{(t)})^T(r^{(t)} - (I - G^T)e_u r_u^{(t)}), \\
&= r^{(t)T}r^{(t)} - \left(\sum_u \tilde{\mathcal{P}}(u) r_u^{(t)} e_u^T\right)(I - G)r^{(t)} - r^{(t)T}(I - G^T)\left(\sum_u \tilde{\mathcal{P}}(u)e_u r_u^{(t)}\right) \\
&\quad + \sum_u \tilde{\mathcal{P}}(u) r_u^{(t)2}\left[(I - G)(I - G^T)\right]_{uu}.
\end{aligned} \tag{24}$$

By the design, $\tilde{\mathcal{P}}(u)r_u^{(t)} = r_u^{(t)}/\min(td, n)$, so that

$$\sum_u \tilde{\mathcal{P}}(u)e_u r_u^{(t)} = \frac{r_u^{(t)}}{\min(td, n)}. \tag{25}$$

Similarly, since $\tilde{\mathcal{P}}(u)r_u^{(t)2} = r_u^{(t)2}/\min(td, n)$,

$$\sum_u \tilde{\mathcal{P}}(u) r_u^{(t)2}\left[(I - G)(I - G^T)\right]_{uu} = \frac{r^{(t)T}Dr^{(t)}}{\min(td, n)}, \tag{26}$$

where D is defined to be a diagonal matrix such that

$$D_{uu} = \left[(I - G)(I - G^T)\right]_{uu} = 1 - 2G_{uu} + \sum_k G_{uk}^2. \tag{27}$$

Therefore, we substitute (25) and (26) into (24) to show that

$$\mathbb{E}_{\tilde{\mathcal{P}}}\left[\left\|r^{(t+1)}\right\|_2^2 \middle| r^{(t)}\right] = r^{(t)T}\left(I - \frac{2I - G - G^T - D}{\min(td, n)}\right)r^{(t)}. \tag{28}$$

We substitute (28) and Lemma D.1a into (23) to show that

$$\begin{aligned}
&\mathbb{E}_{\tilde{\mathcal{P}}}\left[\left\|r^{(t+1)} - \mathbb{E}_{\tilde{\mathcal{P}}}\left[r^{(t+1)}\right]\right\|_2^2 \middle| r^{(t)}\right], \\
&\quad = r^{(t)T}\left(I - \frac{2I - G - G^T - D}{\min(td, n)}\right)r^{(t)} - r^{(t)^T}\left(I - \left(\frac{I - G}{\min(td, n)}\right)\right)\left(I - \left(\frac{I - G^T}{\min(td, n)}\right)\right)r^{(t)}, \\
&\quad = r^{(t)T}\left(\frac{D}{\min(td, n)} - \frac{(I - G)(I - G^T)}{\min(td, n)^2}\right)r^{(t)}, \\
&\quad \leq \left\|\frac{D}{\min(td, n)} - \frac{(I - G)(I - G^T)}{\min(td, n)^2}\right\|_2 \left\|r^{(t)}\right\|_2^2.
\end{aligned} \tag{29}$$

By definition, for all u,

$$\|G\|_2 = \left\|G^T\right\|_2 = \max_{\|x\|_2=1} \left\|G^T x\right\|_2 \geq \sqrt{e_u^T G G^T e_u} = \sqrt{\sum_k G_{uk}^2}. \tag{30}$$

Therefore, $G_{uu}^2 \leq \sum_k G_{uk}^2 \leq \|G\|_2^2$, and

$$D_{uu} = 1 - 2G_{uu} + \sum_k G_{uk}^2 \leq 1 + 2\|G\|_2 + \|G\|_2^2 = (1 + \|G\|_2)^2. \tag{31}$$

Substitute (31) into (29) to show that

$$\mathbb{E}_{\tilde{\mathcal{P}}}\left[\left\|r^{(t+1)} - \mathbb{E}_{\tilde{\mathcal{P}}}\left[r^{(t+1)}\right]\right\|_2^2 \middle| r^{(t)}\right] \leq \frac{(1 + \|G\|_2)^2}{\min(td, n)} \left(1 + \frac{1}{\min(td, n)}\right) \left\|r^{(t)}\right\|_2^2. \tag{32}$$

We will use the two expressions given in Lemma D.1b to get different upper bounds on $\mathbb{E}_{\tilde{\mathcal{P}}}[\|r^{(t+1)}\|_2^2]$, and then take the minimum. The first bound is most relevant in the sparse setting when n is large and d and $\|G\|_2$ are small. We substitute (32) and the first expression in Lemma D.1b into (22) to show that

$$\mathbb{E}_{\tilde{\mathcal{P}}}[\|r^{(t+1)}\|_2^2] \leq a_t \mathbb{E}_{\tilde{\mathcal{P}}}[\|r^{(t)}\|_2^2] + b_t, \tag{33}$$

for

$$a_t = \frac{(1 + \|G\|_2)^2}{\min(td, n)} \left(1 + \frac{1}{\min(td, n)}\right), \tag{34}$$

and

$$b_t = (t + 1)^{-2(1 - \|G\|_2)/d}. \tag{35}$$

Therefore, $\mathbb{E}_{\tilde{\mathcal{P}}}[\|r^{(t+1)}\|_2^2] \leq \sum_{k=1}^t Q_k$ for

$$Q_k = \left(\prod_{m=k+1}^t a_m\right) b_k = \left(\prod_{m=k+1}^t \frac{(1 + \|G\|_2)^2}{\min(md, n)} \left(1 + \frac{1}{\min(md, n)}\right)\right) (k + 1)^{-2(1 - \|G\|_2)/d}. \tag{36}$$

The ratio between subsequent terms can be upper bounded by

$$\frac{Q_k}{Q_{k+1}} \leq \frac{(1 + \|G\|_2)^2}{\min((k + 1)d, n)} \left(1 + \frac{1}{\min((k + 1)d, n)}\right) \left(\frac{k + 1}{k + 2}\right)^{-2(1 - \|G\|_2)/d}. \tag{37}$$

For $k \geq 1$, $d \geq 4$, and $n \geq 8$,

$$\frac{Q_k}{Q_{k+1}} \leq \frac{4}{8} \left(1 + \frac{1}{8}\right) \left(\frac{2}{3}\right)^{2/4} < \frac{1}{2}. \tag{38}$$

It follows that

$$\mathbb{E}_{\tilde{\mathcal{P}}}\left[\left\|r^{(t+1)}\right\|_2^2\right] \leq Q_t \sum_{k=1}^t \left(\frac{1}{2}\right)^{t-k} \leq 2(t + 1)^{-2(1 - \|G\|_2)/d}. \tag{39}$$

We similarly obtain another bound by using the second expression of Lemma D.1b. This bound applies in settings when the residual vector $r^{(t)}$ is no longer sparse. By Lemma D.1b, $\mathbb{E}_{\tilde{\mathcal{P}}}[\|r^{(t+1)}\|_2^2] \leq a_t \mathbb{E}_{\tilde{\mathcal{P}}}[\|r^{(t)}\|_2^2] + b'_t$, for $b'_t = e^{-2t(1-\|G\|_2)/n}$. Therefore, $\mathbb{E}_{\tilde{\mathcal{P}}}[\|r^{(t+1)}\|_2^2] \leq \sum_{k=1}^t Q'_k$ for

$$Q'_k = \left(\prod_{m=k+1}^{t} \frac{(1+\|G\|_2)^2}{\min(md,n)} \left(1 + \frac{1}{\min(md,n)}\right)\right) e^{-2k(1-\|G\|_2)/n}. \tag{40}$$

The ratio between subsequent terms can be upper bounded by

$$\frac{Q'_k}{Q'_{k+1}} \leq \left(\frac{(1+\|G\|_2)^2}{\min((k+1)d,n)} \left(1 + \frac{1}{\min((k+1)d,n)}\right)\right) e^{2(1-\|G\|_2))/n}. \tag{41}$$

For $k \geq 1$, $d \geq 4$, and $n \geq 8$,

$$\frac{Q'_k}{Q'_{k+1}} \leq \frac{9e^{2(1-\|G\|_2)/n}}{16} < \frac{3}{4}. \tag{42}$$

It follows that

$$\mathbb{E}_{\tilde{\mathcal{P}}}[\|r^{(t+1)}\|_2^2] \leq Q'_t \sum_{k=1}^{t} \left(\frac{3}{4}\right)^{t-k} \leq 4e^{-2t(1-\|G\|_2)/n}. \tag{43}$$

□

By Markov's inequality, $\mathbb{P}(\|r^{(t)}\|_2 \geq \epsilon) \leq \delta$ for $\mathbb{E}_{\tilde{\mathcal{P}}}[\|r^{(t)}\|_2^2] \leq \delta\epsilon^2$. Therefore, we can directly apply Lemma D.2 to show that if $\|G\|_2 < 1$, $d \geq 4$, and $n \geq 8$, the algorithm terminates with probability at least $1-\delta$ for

$$t \geq \min\left(\left(\frac{2}{\delta\epsilon^2}\right)^{d/2(1-\|G\|_2)}, 1 + \frac{n}{2(1-\|G\|_2)} \ln\left(\frac{4}{\delta\epsilon^2}\right)\right). \tag{44}$$

Since we are concerned with asymptotic performance, the conditions $d \geq 4$ and $n \geq 8$ are insignificant. To bound the total number of multiplications, we multiply the number of iterations by the maximum degree d.

Translating analysis for $\tilde{\mathcal{P}}$ to $\mathcal{P}$

Let us consider implementation A, which samples coordinates from $\tilde{\mathcal{P}}$, and implementation B, which samples coordinates from $\mathcal{P}$. Let R_A denote the sequence of residual vectors $r^{(t)}$ derived from implementation A, and let R_B denote the sequence of residual vectors $r^{(t)}$ derived from implementation B. The length of the sequence is the number of iterations until the algorithm terminates. We define a joint distribution such that $\mathbb{P}(R_A, R_B) = \mathbb{P}(R_A)\mathbb{P}(R_B|R_A)$.

Let $\mathbb{P}(R_A)$ be described by the algorithm sampling coordinates from $\tilde{\mathcal{P}}$. The sequence R_A can be sampled by separately considering the transitions when non-zero valued coordinates are chosen, and the length of the repeat in between each of these transitions. Given the current iteration t and the sparsity of vector $r^{(t)}$, we can specify the distribution for the number of iterations until the next transition. If we denote $\tau_t = \min\{s : s > t \text{ and } r^{(s)} \neq r^{(t)}\}$, then

$$\mathbb{P}(\tau_t > k | r^{(t)}) = \prod_{q=1}^{k} \left(1 - \frac{\|r^{(t)}\|_0}{\min((t+q)d,n)}\right). \tag{45}$$

Conditioned on the event that a non-zero valued coordinate is chosen at a particular iteration t, the distribution over the chosen coordinate is the same as $\mathcal{P}$.

For all t, $r^{(t+1)} \neq r^{(t)}$ if and only if the algorithm chooses a non-zero valued coordinate of $r^{(t)}$ at iteration t, which according to $\widetilde{\mathcal{P}}$, occurs with probability $1 - \|r^{(t)}\|_0 / \min(td, n)$. Therefore, given the sequence R_A, we can identify in which iterations coordinates with non-zero values were chosen. Let $\mathbb{P}(R_B|R_A)$ be the indicator function which is one only if R_B is the subsequence of R_A corresponding to the iterations in which a non-zero valued coordinate was chosen.

We can verify that this joint distribution is constructed such that the marginals correctly correspond to the probability of the sequence of residual vectors derived from the respective implementations. For every (R_A, R_B) such that $\mathbb{P}(R_B|R_A) = 1$, it also follows that $|R_A| \geq |R_B|$, since R_B is a subsequence. For every q,

$$\{(R_A, R_B) : |R_A| \leq q\} \subset \{(R_A, R_B) : |R_B| \leq q\}, \implies \mathbb{P}(|R_A| \leq q) \leq \mathbb{P}(|R_B| \leq q). \tag{46}$$

Therefore, we can conclude that since the probability of the set of realizations such that implementation A terminates within the specified bound is larger than $1-\delta$, it also follows that implementation B terminates within the specified bound with probability larger than $1 - \delta$. Therefore, since we have proved Theorem 3.3 for implementation A, the result also extends to implementation B, i.e., our original algorithm.

E Comparison between Synchronous and Asynchronous Algorithms

The main difference between Theorem 2.1 and Theorem 3.3 is in the dependence on d. There is an intuitive reasoning which shows why our analyses result in such a gap. Let $C_S(t) = d^t$ denote the upper bound on the number of multiplications performed by the synchronous algorithm in the first t iterations. Let $C_A(t) = td$ denote the upper bound on the number of multiplications performed by the asynchronous algorithm in the first t iterations. For some $t_S > 0$, let $t_A = C_A^{-1}(C_S(t_S)) = d^{t_S - 1}$, such that $C_A(t_A) = C_S(t_S)$. This allows us to compare the error in the two algorithms given the same number of multiplications. From (13), the residual vector of the synchronous algorithm after t_S iterations is bounded by $\|r^{(t_S)}\|_2^2 \leq \|G\|_2^{2t_S}$. From Lemma D.2, the expected residual vector of the asynchronous algorithm after t_A iterations is bounded by

$$\mathbb{E}[\|r^{(t_A)}\|_2^2]\| \leq 2t_A^{-2(1-\|G\|_2)/d} \approx 2t_A^{2\ln(\|G\|_2)/d} = 2\|G\|_2^{2\ln(t_A)/d} = 2\|G\|_2^{2(t_S-1)\ln(d)/d}. \tag{47}$$

Therefore the ratio of the convergence rates of the two algorithms obtained by our analysis is given by $\ln(\|r^{(t_S)}\|_2)/\ln(\|\mathbb{E}[r^{(t_A)}]\|_2) \approx d/\ln(d)$.