An efficient Gossip-based algorithm to compute aggregation functions in a network

Raghuram Nagireddy and Haimonti Dutta

Columbia University, New York, NY
{rrn2111,hd2200}@columbia.edu
www.ccls.columbia.edu

Abstract. In this paper we present an improvement for a Gossip-based protocol called Push-Sum [12], and demonstrate its application on a distributed support vector machine (DSVM) algorithm called GADGET [8]. Push-Sum is commonly used to compute sums, averages and other aggregate functions in a distributed setting. Its performance relies on the mixing time of its underlying markov chain. We will show that by assigning a specific structure to this markov chain the push-sum algorithm can be made to converge faster. The claim will be substantiated by showing an improvement in performance over the Uniform Gossip protocol which has been the default implementation for Push-Sum, by applying the new protocol that we develop to GADGET which builds the SVM classifier on a network.

1 Introduction

Due to the unprecedented increase in connectivity of computers, large-scale networks have emerged and hence decentralized computation is becoming more common. The most efficient protocols for performing distributed tasks in such networks are the Gossip protocols, inspired by the social networks. Gossip protocols usually involve a node communicating with few of its neighbors and are agnostic of the network wide information. As a result they overcome the problems associated with other protocols such as link failures, complexity, instability and scalability. Even though Gossip protocols give probabilistic guarantees they are widely employed these days for their high resistance to failures. In very massive networks like P2P and sensor networks protocols like Gossip are the only way out as others are not tolerant to even moderate failures.

We are very often interested in aggregation functions like sums, averages over a network when dealing with massive datasets. A gossip-based algorithm called Push-Sum is one such aggregation algorithm developed by [12] and has proven convergence properties. This algorithm essentially simulates a Markov chain over the network by message passing between neighbors. The quantities of these messages shared are defined by the transition matrix of the Markov Chain which we refer to as **B**. It also determines which of the neighbors a node can communicate with. A short description of the mechanics of Push-Sum for computing the 'average' function is as follows: It starts by initializing two variables

w_i and S_i at each node i. w_i's are all set to 1 and S_i's are set to their respective initial values K_i which need to be averaged. Now in subsequent iterations, for each node i, S_i and w_i are updated by summing up b_{ji} fractions of each of its neighbors' values S_j and w_j respectively. This essentially simulates the 1-step state transitions of the markov chain defined by $\mathbf{B}$ with initial states S and w. Proceeding this way, after a sufficient time S_i's converge to $\pi_i \sum_{i=1}^{n} K_i$ where π is the steady state distribution of the chain (assuming there is one) and n is number of nodes in the network. Similarly w_i's converge to $n\pi_i$. Hence the average can be simply computed as $\frac{S_i}{w_i}$. The convergence of Push-Sum algorithm depends on the mixing time of its markov chain which is defined below.

Mixing Time: There are many possible definitions for Mixing time available in literature. We choose the simplest one based on total variation as follows [22]. For the transition matrix $\mathbf{B}$ with stationary distribution $\pi = [\pi_i]$, and assuming $\Delta_i(t)^1 = \frac{1}{2} \sum_{j=1}^{n} |\mathbf{B}^{\mathbf{t}}_{\mathbf{ij}} - \pi_{\mathbf{j}}|$, the ϵ mixing time is defined as

$$\tau(\epsilon, \mathbf{B}) = \max_{\mathbf{i}} \inf\{\mathbf{t} : \mathbf{\Delta_i}(\mathbf{s}) < \epsilon, \forall \mathbf{s} > \mathbf{t}\} \tag{1}$$

Mixing time indicates the amount of time a markov chain takes to converge to a stationary distribution. It also implies that lower the mixing time of Push-Sum, faster a node in a network arrives at an aggregate value.

When a node randomly chooses one neighbor to communicate with equal probability, the protocol is called Push-Sum with Uniform Gossip. The mixing time of Uniform gossip protocol is $O(\log_2 n + \log_2 \frac{1}{\epsilon})$ cycles which means it takes as many iterations for the variance in the aggregate function value across the network to fall below ϵ.

Our goal in this paper is to design transition matrices $\mathbf{B}$ for the markov chain induced by Push-Sum algorithm which make it converge faster than the commonly used Uniform Gossip algorithm. We first describe a homogeneous markov chain based on convex optimization formulation of B, that shows better performance than Uniform Gossip and later move on to a novel time-inhomogeneous chain which is the major contribution of our work. We will show that this chain has a mixing time of $O(\log_{r+1} n + \log_{r+1} \frac{1}{\epsilon})$.

The performance of the methods proposed in this paper are tested on an algorithm called GADGET [8] which is an efficient distributed support vector machine (DSVM) algorithm. Push-Sum is a crucial part of this algorithm and its mixing time directly affects GADGET's convergence. We show the results of our proposed variants of Push-Sum algorithm applied to GADGET to substantiate the claimed performance improvements. More details of DSVMs, GADGET are mentioned in next sections.

The rest of the paper is organized as follows: In section 2, the previous work on gossip protocols, distributed SVMs etc. is briefed, section 3 describes the first

[1] Note that Δ can also be seen as the variation distance between two probability distributions, in this case the distribution induced by the columns of B at time t and that of the stationary distribution π

proposed improvement of Push-Sum based on optimization, after that the second improvement based on the time-inhomogeneous markov chain is discussed which spans sections 4 and 5. Finally the empirical results are detailed in section 6.

2 Related Work

2.1 Gossip Protocols

Gossip protocols are based on the concept of rumor emerging from a node in a network and eventually spreading through the entire network (like for e.g, in social networks). These protocols bear a certain set of properties which make them highly suitable for massive, dynamic networks. Some of these are [22]: a node can only talk to its neighbors, a node i performs $O(d_i \log n)$ computation per unit time where d_i is its degree, no synchronization between a node and its neighbors etc. There are various types of gossip protocols in existence such as dissemination protocols (which work by nodes flooding the network with the information they know), anti-entropy protocols (or error correcting protocols for reliable message delivery), aggregates computing protocols etc. These protocols are utilized for tasks such as network scheduling, network optimization and separable function computation problems [22], distributed averaging [17], finding a rumor source [23] etc. We focus on aggregation protocols in this paper.

Gossip-based aggregation protocols are essential in networks such as sensor networks, P2P networks, mobile ad-hoc networks, social networks because such networks are not just formed for communication but for co-ordination and consensus as well. These protocols attempt to compute a function $f(x_1, x_2..x_n)$ at all nodes in a network where x_i is the information available at node i. For example f could be 'random', 'sum', 'average', 'smallest' etc. Push-Sum is one such aggregation protocol developed by [12] for computing the 'sum' function (there are slight variants of Push-Sum to compute other aggregate functions but all of them rely on the same core idea). It could be used to solve various problems such as computing the average cholesky factorization in a cycle in decentralized orthonormalization of a big matrix and spectral decomposition of a graph [11], linear synopses problem, random sampling and computing quantiles from a distribution [12].

We shift our attention to the convergence speed of Push-Sum algorithm as the primary goal of this paper is to improve this. It has been proven that the Push-Sum converges exponentially fast w.r.t network size by every node selecting one of its neighbors uniformly at random to share its information, which is also called Uniform Gossip. Uniform Gossip algorithm can be seen as a random walk on a graph [14]. Our work, we believe is an extension to this Uniform Gossip Push-Sum protocol. We design a time-inhomogeneous markov chain with faster convergence and provide analysis based on ideas from [7],[18],[19],[20],[21].

2.2 Distributed Support Vector Machine(DSVM)

Distributed Machine Learning algorithms are becoming very important these days because of the sheer sizes of the data sets involved in performing a learning task. For example one might want to learn, classify and predict the entire wikipedia content, the scientific articles from google scholar, the collections of online books at Amazon and many more. All these tasks will require processing tera-bytes of data and so a distributed machine learning algorithm is necessary. A set of algorithms designed primarily for classification in a distributed setting are the DSVMs. In a distributed setting the SVM [4] problem involves computing one unique classifier across all the nodes in a network with each node having a certain fraction of the data.

GADGET: There are many implementations of DSVMs which require a centralized computation of some sort [13] and sometimes working only on specific topologies [15]. GADGET [8] is a completely decentralized protocol which is not confined to specific topologies and hence is a natural choice for massive scale networks. The goal of this algorithm (like a general SVM classifier) is to compute the normal vector to the classification boundary (ignoring the offset for the moment). It starts by setting the components of the vector to zero. In subsequent iterations all nodes simultaneously execute the following steps in sequence:

1. compute the loss induced by their respective local classifiers
2. update the local component weights using the loss computed, by sub-gradient update rule
3. arrive at a global consensus on the classifier weights by calling Push-Sum
4. project the weight vector onto a ball of a chosen radius
5. compute the average of global weights computed so far

In step 3 above, Push-Sum is used to compute the global weight vector by averaging the weight vectors computed locally. As noted in the previous sections its convergence is impacted by the mixing speed of the underlying markov chain. Therefore an efficient markov chain designed for Push-Sum algorithm will reduce the mixing time and in-turn speeds up the GADGET algorithm (since Push-Sum is executed in every iteration of GADGET, the speeding factor is multiplied and it converges even more faster).

3 Reducing mixing time by optimizing the B matrix

There are many ways to obtain fast mixing markov chains for example, the maximum degree chain (with transition probability along an edge inversely proportional to the maximum degree of any node), the Metropolis-Hastings chain (with transition probabilities inversely proportional to the degrees of each node) etc. but these are heuristics based and do not give the best possible chains. Instead we observe that the mixing time of a markov chain depends on second largest eigenvalue modulus (SLEM) [3] of its transition matrix (assuming time-invariance). So we can deterministically reduce the mixing time in a general graph by bounding the SLEM. Assuming that $1 = \beta_0 > \beta_1 \geq \beta_2 .. \geq \beta_{m-1} \geq -1$ are the eigen values of **B** defined above, and $\beta_* = max(\beta_1, |\beta_{m-1}|)$, we have the following inequality:

$$4||B_{i.}^n - \pi||_{Var}^2 \leq \frac{1-\pi_i}{\pi_i}\beta_*^{2n} \tag{2}$$

Var in the above inequality stands for the Variation distance. This indicates that the mixing time is upper bounded by the SLEM of $\mathbf{B}$. So we can reduce the mixing time by simply minimizing this upper bound. Some analytic approaches exist in literature for bounding the SLEM. These are: coupling methods and strong stationary times [1] & [5],conductance [9], geometric bounds [6], and multicommodity flows [24] & [10]. The convex optimization approach proposed by [3] obtains a substantially fast convergence compared to the above mentioned methods, takes polynomial time to solve and also applies to a general graph. Hence it's the best choice for obtaining the fastest converging markov chain.

3.1 Semi Definite Program(SDP) formulation

From the above idea our task of minimizing the SLEM of $\mathbf{B}$ can be formulated as an optimization problem in many ways [3]. By observing that the SLEM is a convex function of $\mathbf{B}$ and that the constraints are all linear (like sums of rows to 1, positivity etc.) it is easy to see the convex optimization formulation as below:

$$\begin{aligned} &\text{minimize } \mu(B) = ||B - \frac{1}{n}\mathbf{1}\mathbf{1}^{\mathbf{T}}||_{\mathbf{2}} \\ &\text{subject to } B \geq 0, B\mathbf{1} = \mathbf{1}, \mathbf{B} = \mathbf{B}^{\mathbf{T}}, \\ &\qquad B_{ij} = 0, (i,j) \notin \varepsilon \end{aligned} \tag{3}$$

We work with the SDP formulation as it could be solved efficiently with the available sdp-solvers. By introducing a scalar s we can convert this into an SDP as follows:

$$\begin{aligned} &\text{minimize } s \\ &\text{subject to } -sI \preceq B - \frac{1}{n}\mathbf{1}\mathbf{1}^{\mathbf{T}} \preceq \mathbf{sI} \\ &\qquad B \geq 0, B\mathbf{1} = \mathbf{1}, \mathbf{B} = \mathbf{B}^{\mathbf{T}}, \\ &\qquad B_{ij} = 0, (i,j) \notin \varepsilon \end{aligned} \tag{4}$$

In the above formulations, $||.||_2$ is the L_2 norm and $(i,j) \notin \varepsilon$ means that there is no edge between nodes i and j in the network.

3.2 Solving SDP

The above SDP can be solved using softwares such as csdp (`http://infohost.nmt.edu/~borchers/csdp.html`), yalmip (`http://users.isy.liu.se/johanl/yalmip/`), sdpt3 (`http://www.math.nus.edu.sg/~mattohkc/`) etc. In this paper we experiment on networks with edge connectivity of less than 1000 and so we use the convex optimization package (`http://cvxr.com/cvx/`) which performs the best for such graphs. We also assume that the network has a fixed number of nodes and that the topology is static (meaning that the edge connectivity doesn't change with time). This may not be true for real networks, in which case re-optimization has to be done whenever there's a change in topology or a new node enters the network to ensure that the correct **B** is used for information sharing. However such dynamic networks are beyond the scope of discussion. Now in order to execute Push-Sum a node i needs to know the fraction of shares b_{ij} for its neighbors j. There are two ways to achieve this in any network. One way is as follows: A Gossip-based dissemination protocol (described in previous sections) is run where every node sends multi-cast messages to all its neighbors in a cycle, about its knowledge of the topology. After convergence all nodes will have entire topology information and will solve the SDP. Hence all nodes obtain the required information to perform Push-Sum. This method may be feasible since the protocol has to be run only once before Push-Sum and so is not a great computational overhead. Another way is that the SDP problem can be solved in a distributed fashion. The distributed SDP solver developed by [2] could be employed for this purpose. This method may be much more practical since it makes sure only the required information is made available to a node for executing Push-Sum whereas the old method floods a node with the entire topology information which violates one of the properties of Gossip protocol mentioned in last section. But the discussion as to what's the best distributed algorithm for optimization is beyond the scope of this paper. We start by assuming this optimization has been done and all nodes have the required information for running Push-Sum. The optimization overhead is excluded from the performance evaluation of Push-Sum for all the experiments conducted.

4 A new protocol with faster convergence

In this section we develop a time-inhomogeneous markov chain which converges faster than the Uniform Gossip protocol. Specifically we examine the convergence properties of a markov chain constructed as follows - In every time step a node uniformly chooses some r nodes from its neighbors. Then it shares $\frac{1}{r}^{th}$ fraction of its value(s) with each of the selected neighbors. All nodes do the same until convergence. Convergence for this algorithm is guaranteed by a property called Mass Conservation [12] which is later re-introduced in the convergence proof. The new protocol is as follows:

We assume that each node maintains 2 values $s_{t,i}$ and $w_{t,i}$ with $s_{0,i} = x_i$ and $w_{0,i} = 1$, and at time 0 each node sends $(s_{0,i}, w_{0,i})$ to itself. In each subsequent cycle nodes will execute algorithm 1. It performs the following sequence of steps: at time t, a node sums up the values and weights it received from its neighbors, chooses some r of its neighbors uniformly at random and sends equal shares of its current values and weights to them and also itself. With the same reasoning as in Uniform Gossip Push-Sum, the ratio of the current value and weight at a node will eventually converge to the final average.

Algorithm 1 Random r-neighbor Push Sum Protocol

```
1: Let $(\hat{s_k}, \hat{w_k})$ be the pairs sent to i in t-1
2: Let $s_{t,i} = \sum_k \hat{s_k}$, $w_{t,i} = \sum_k \hat{w_k}$
3: Choose r targets uniformly at random
4: Send the pairs $(\frac{1}{r+1}\hat{s_r}, \frac{1}{r+1}\hat{w_r})$ to the selected targets and also to itself
5: $\frac{s_{t,i}}{w_{t,i}}$ is the estimate of the average at time t
```

4.1 Convergence speed of the new protocol

Denote $\mathbf{v_{t,i}}$ as the contribution vector, which means $v_{t,i,j}$ is the contribution of node j at node i at time t. We work in terms of $\mathbf{v_{t,i}}$. It can be seen that $s_{t,i}$ can be expressed as $\mathbf{v_{t,i}.x} = \sum_\mathbf{j} \mathbf{v_{t,i,j}.x_j}$ and $w_{t,i}$ as $||\mathbf{v_{t,i}}||_\mathbf{1}$ where $||.||_1$ is the L_1 norm. In order to examine the run-time properties of algorithm 1 we need to understand *mass conservation.* It means that the sum of a node j's contribution at all nodes i in every time step is 1. Mathematically it means $\sum_i v_{t,i,j} = 1$ for all j. It also implies that $\sum_i w_{t,i} = n$. With this notation in place, we have the following theorem.

Theorem 1. *Assuming all nodes have access to equal number of neighbors in the network, the convergence speed of Algorithm 1 is $O(log_{r+1}n + log_{r+1}\frac{1}{\epsilon})$ where r is the number of neighbors selected, n the number of nodes in the network and ϵ the relative error in the final average computed across network.*

We need to define a quantity called potential function to understand the proof of Theorem 4.1. It is defined as $\Phi_t = \sum_{i,j}(v_{t,i,j} - \frac{w_{t,i}}{n})^2$ which essentially captures the variance of the contributions of all nodes over entire network. Following lemma shows that this quantity drops to less than $\frac{1}{r+1}$ of its previous value.

Lemma 1. *Let E stand for expectation, $N(k, k')$ stand for the number of nodes commonly chosen by nodes k and k'. We have,*

$$E[\Phi_{t+1}|\Phi_t = \phi] = \left(\frac{1}{r+1} - \frac{2E[N(k,k')]}{(r+1)^2}\right)\phi$$

.

Proof. For simplicity let's assume $v_{t,i,j} = v_{i,j}$ and $w_{t,i} = w_i$. Also let $f(k)$ denote the set of all neighbors of node k. We have,

$$
\begin{aligned}
\Phi_{t+1} =& \sum_{i,j} \left(\frac{1}{r+1}(v_{i,j} - \frac{w_i}{n}) + \sum_{k:i\in f(k)} \frac{1}{r+1}(v_{k,j} - \frac{w_k}{n}) \right)^2 \\
=& \frac{1}{(r+1)^2} \sum_{i,j}(v_{i,j} - \frac{w_i}{n})^2 + \frac{1}{(r+1)^2} \sum_{\substack{i,j,\\ k:i\in f(k)}} (v_{k,j} - \frac{w_k}{n})^2 \\
&+ \frac{2}{(r+1)^2} \sum_{i,j,k:i\in f(k)} (v_{i,j} - \frac{w_i}{n})(v_{k,j} - \frac{w_k}{n}) \\
&+ \frac{2}{(r+1)^2} \sum_{\substack{j,k,\\ k':k\neq k',\\ i:i\in f(k),f(k')}} (v_{k,j} - \frac{w_k}{n})(v_{k',j} - \frac{w_{k'}}{n}) \\
=& \frac{1}{r+1}\Phi_t + \frac{2}{(r+1)^2} \sum_{\substack{i,j,\\ k:i\in f(k)}} (v_{i,j} - \frac{w_i}{n})(v_{k,j} - \frac{w_k}{n}) \\
&+ \frac{2}{(r+1)^2} \sum_{\substack{j,k,\\ k'\neq k}} N(k,k')(v_{k,j} - \frac{w_k}{n})(v_{k',j} - \frac{w_{k'}}{n})
\end{aligned}
$$

Above, we used the fact that the expression $\sum_{i,j}(v_{i,j} - \frac{w_i}{n})^2$ repeats $r+1$ times as a result of uniformity in local topologies at all nodes across the network. Taking expectations on both sides we get,

$$
\begin{aligned}
E[\Phi_{t+1}|\Phi_t = \phi] =& \frac{1}{r+1}\phi + \frac{2}{(r+1)^2} \sum_{i,j,k} P(i \in f(k))(v_{i,j} - \frac{w_i}{n})(v_{k,j} - \frac{w_k}{n}) \\
&+ \frac{2}{(r+1)^2} \sum_{\substack{j,k,\\ k'\neq k}} E[N(k,k')](v_{k,j} - \frac{w_k}{n})(v_{k',j} - \frac{w_{k'}}{n})
\end{aligned}
$$

Again, making use of the fact that every node has same number of neighbors and that all nodes are chosen with equal probability, we have $P(i \in f(k)), E[N(k,k')]$ same for all i, k, k'. Let's call them p, e respectively. Hence the above equation reduces as,

$$
\begin{aligned}
E[\Phi_{t+1}|\Phi_t = \phi] =& \frac{1}{r+1}\phi + \frac{2p}{(r+1)^2}\sum_{i,j,k}(v_{i,j} - \frac{w_i}{n})(v_{k,j} - \frac{w_k}{n}) \\
&+ \frac{2e}{(r+1)^2}\sum_{\substack{j,k,\\ k'\neq k}}(v_{k,j} - \frac{w_k}{n})(v_{k',j} - \frac{w_{k'}}{n}) \\
=& \frac{1}{r+1}\phi + \frac{2p+2e}{(r+1)^2}\sum_j\sum_i(v_{i,j} - \frac{w_i}{n})\sum_k(v_{k,j} - \frac{w_k}{n}) \\
&- \frac{2e}{(r+1)^2}\sum_{k,j}(v_{k,j} - \frac{w_k}{n})^2 \\
=& \left(\frac{1}{r+1} - \frac{2e}{(r+1)^2}\right)\phi
\end{aligned}
$$

Above, we used the fact that $\sum\limits_k(v_{k,j} - \frac{w_k}{n}) = 0$ because of mass conservation. Also, we didn't bother to compute e in closed form as it turns out to be quite messy and is also not of great interest. □

Now, by taking expectations repeatedly, we get $E[\Phi_t] \leq n.(r+1)^{-t}$ from which it is easy to see that after $O(\log_{r+1} n + log_{r+1}\frac{1}{\epsilon})$ rounds, $E[\Phi_t] \leq \epsilon$ which proves Theorem 4.1.
The above proof suggests that if a node randomly chooses some r of its neighbors to communicate with instead of 1 as in Uniform Gossip, the Push-Sum algorithm converges at an improved logarithmic rate attributed to the bigger base of the logarithm. We also note that bigger the base better the convergence rate but it comes at the cost of increased communication overhead in which every node has to talk to more than 1 neighbor in a cycle. An optimal decision has to be made which strikes the right balance between communication costs and convergence speed of Push-Sum protocol. Another point to be made is that even though the analysis is done making a rather simplistic assumption that all nodes have equal number of neighbors, we believe similar results can be observed for convergence speed when we relax this assumption. But we foresee that the analysis will be much more complex.

5 Empirical Results

5.1 Datasets

As mentioned earlier, our task is to build a distributed support vector machine classifier using GADGET. We have used two datasets for this purpose. The first dataset comes from the Pascal Large Scale Learning Challenge (2008) [2] where the goal is to build a 2-class classifier with lowest error and computational time.

[2] More information at `http://largescale.ml.tu-berlin.de/instructions/`

It has 400,000 data points with 2000 features. All the features are numeric and are pre-processed to fall in the range [0,1]. Second dataset comes from UCIrvine [3] where the task is to predict forest cover type from cartographic variables (a binary classification problem). This has 581,012 data points and 54 features scaled to [0,1]. We build a binary SVM classifier on these datasets. For this purpose we split the data equally into M parts where M is the number of nodes in the network and so each node holds $\frac{1}{M}$ fraction of the data. The software package used for all experiments is the peersim simulator [16] version 1.0 [4]. It mainly has two types of classes - protocol and control which are implemented by nodes in the network to perform a task. Protocol objects hold the core logic which is run in every cycle of the simulation and Control objects serve as observers which can access/modify the state of simulation. We make use of the Control objects to check whether the variance of svm weights across the network has fallen below a certain level in which case we exit the simulation and return the global svm weights. We fix the network size to 500 nodes and the topology is generated using peersim's inbuilt functions. We generate uniformly random static topologies for our experiments with equal degrees for all nodes. GADGET is implemented as a protocol in peersim by using the new Push-Sum algorithm developed in this paper. We evaluate the performance of GADGET by counting the number of cycles taken for convergence.

5.2 Results using SDP

The following results show a comparison between the performances of the GADGET algorithm using Push-Sum with Uniform Gossip vs Push-Sum with optimized B. For the sake of computing the **B** matrix we schedule a Control object at the beginning of the simulation, which has access to the entire topology. Then we perform the optimization and make the **B** matrix global (we use the peersim's provision for defining global variables to bypass the additional complications of distributed optimization but we strictly disallow any other variable to be shared by nodes, we also exclude the time taken for optimization from run-time analysis). As described in the beginning GADGET is a distributed implementation of SVM and its run-time depends linearly on the mixing speed of Push-Sum (τ_{mix}). We present the results for the number of cycles taken to converge to the final SVM classifier over the network by varying the average degree of a node and the network size, one at a time. [**Results yet to be added here**]

5.3 Results with random r-neighbor Push-Sum protocol

Following results show a comparison in convergence time using Push-Sum with Uniform Gossip vs the new protocol. As in previous section, performance is mea-

[3] More information and dataset available for download at `http://archive.ics.uci.edu/ml/datasets/Covertype`

[4] Peersim simulator and the source code for GADGET implemented with new Push-Sum protocol along with download and installation instructions can be found at `http://www1.ccls.columbia.edu/~dutta/gadget.html`

sured by the number of cycles taken by the GADGET algorithm for convergence. **[Results yet to be added here]**

Future Work: So far we are successful in developing an improved Push-Sum protocol which is an essential part of many distributed algorithms. We have shown how the GADGET algorithm converges faster with the our protocol. Our future work involves coming up with a hybrid algorithm combining the optimization idea and the random r-neighbor Push-Sum protocol described in the paper, and give a concrete proof of its convergence speed which is valid in a general case, by relaxing any assumptions made for simplifying the proof shown in the paper. We are hoping that this would out-perform any existing decentralized aggregation protocols.

Acknowledgements: This research was supported by the NSF grant at Center for Computational Learning Systems (CCLS), Columbia University.

References

[1] Aldous, David. 1983. Random walks on finite groups and rapidly mixing Markov chains. *Pages 243–297 of: Seminar on probability, XVII*, vol. 986. Springer.

[2] Biswas, Pratik, & Ye, Yinyu. 2006. A distributed method for solving semidefinite programs arising from ad hoc wireless sensor network localization. *Pages 69–84 of: Multiscale optimization methods and applications.* Springer.

[3] Boyd, Stephen, Diaconis, Persi, & Xiao, Lin. 2003. Fastest Mixing Markov Chain on A Graph. *SIAM REVIEW*, **46**, 667–689.

[4] Cortes, Corinna, & Vapnik, Vladimir. 1995. Support vector machine. *Machine learning*, **20**(3), 273–297.

[5] Diaconis, Persi. 1988. Group representations in probability and statistics. *Lecture Notes-Monograph Series*, **11**, i–192.

[6] Diaconis, Persi, & Stroock, Daniel. 1991. Geometric bounds for eigenvalues of Markov chains. *The Annals of Applied Probability*, **1**(1), 36–61.

[7] Douc, Randal, Moulines, E, & Rosenthal, Jeffrey S. 2004. Quantitative bounds on convergence of time-inhomogeneous Markov chains. *The Annals of Applied Probability*, **14**(4), 1643–1665.

[8] Hensel, Chase, & Dutta, Haimonti. *GADGET SVM: a Gossip-bAseD sub-GradiEnT SVM solver.*

[9] Jerrum, Mark, & Sinclair, Alistair. 1989. Approximating the permanent. *SIAM journal on computing*, **18**(6), 1149–1178.

[10] Kahale, Nabil. 1997. A semidefinite bound for mixing rates of Markov chains. *Random Structures & Algorithms*, **11**(4), 299–313.

[11] Kempe, David, & McSherry, Frank. 2008. A decentralized algorithm for spectral analysis. *Journal of Computer and System Sciences*, **74**(1), 70–83.

[12] Kempe, David, Dobra, Alin, & Gehrke, Johannes. 2003. Gossip-Based Computation of Aggregate Information. IEEE Computer Society.

[13] Kokiopoulou, Effrosini, & Frossard, Pascal. 2006. Distributed SVM applied to image classification. *Pages 1753–1756 of: Multimedia and Expo, 2006 IEEE International Conference on.* IEEE.

[14] Lovász, László. 1993. Random walks on graphs: A survey. *Combinatorics, Paul erdos is eighty*, **2**(1), 1–46.

[15] Lu, Yumao, Roychowdhury, Vwani, & Vandenberghe, Lieven. 2008. Distributed parallel support vector machines in strongly connected networks. *Neural Networks, IEEE Transactions on*, **19**(7), 1167–1178.

[16] Montresor, Alberto, & Jelasity, Márk. 2009 (Sept.). PeerSim: A Scalable P2P Simulator. *Pages 99–100 of: Proc. of the 9th Int. Conference on Peer-to-Peer (P2P'09).*

[17] Rajagopalan, Shreevatsa, & Shah, Devavrat. 2011. Distributed averaging in dynamic networks. *Selected Topics in Signal Processing, IEEE Journal of*, **5**(4), 845–854.

[18] Saloff-Coste, L, & Zúñiga, J. 2007. Convergence of some time inhomogeneous Markov chains via spectral techniques. *Stochastic processes and their applications*, **117**(8), 961–979.

[19] Saloff-Coste, Laurent, & Zúñiga, J. 2009. Merging for time inhomogeneous finite Markov chains, Part I: singular values and stability. *Electronic Journal of Probability*, **14**, 1456–1494.

[20] Saloff-Coste, Laurent, & Zúñiga, J. 2010. Time inhomogeneous Markov chains with wave-like behavior. *The Annals of Applied Probability*, **20**(5), 1831–1853.

[21] Saloff-Coste, Laurent, & Zuniga, Jessica. 2010. Merging and stability for time inhomogeneous finite Markov chains. *arXiv preprint arXiv:1004.2296.*

[22] Shah, Devavrat. 2009. *Gossip Algorithms.* Tech. rept. Massachusetts Institute of Technology, Cambridge, MA.

[23] Shah, Devavrat, & Zaman, Tauhid. 2011. Rumors in a Network: Who's the Culprit? *Information Theory, IEEE Transactions on*, **57**(8), 5163–5181.

[24] Sinclair, Alistair. 1992. *Improved bounds for mixing rates of Markov chains and multicommodity flow.* Springer.

[25] Voulgaris, Spyros, Jelasity, Mrk, & Steen, Maarten Van. 2003. A Robust and Scalable Peer-to-Peer Gossiping Protocol. *Pages 47–58 of: In 2nd Intl Workshop Agents and Peer-toPeer Computing, LNCS 2872.* Springer.