Distributed Structural Estimation of Graph Edge-Type Weights from Noisy PageRank Orders

CME 323 Project Report, Spring 2015

David Daniels*, Eric Liu†, Charles Zhang‡

* David Daniels, Ph.D. Student, Business Administration, E-mail: ddaniels@stanford.edu
† Eric Liu, MS. Student, Computational and Mathematical Engineering, E-mail: ericql@stanford.edu
‡ Charles Zhang, MS. Student, Statistics, E-mail: cyzhang@stanford.edu

Abstract

We describe a distributed structural estimation approach for recovering graph edge-type weights that are revealed through orders generated by a specific type of influence propagation, Edge-Type Weighted PageRank. Our implementation combines numerical gradient descent with PageRank iterations using the Pregel framework.

1. Introduction

Networks have received a tremendous amount of multidisciplinary interest in recent academic research, largely due to the ability of online networks to facilitate transmissions at unprecedented scale and speed. However, whether such networks are informational[1], social[2], or economic[3], attention has typically focused on graphs with exactly one type of edge. In this paper, we study graphs featuring multiple types of edges, with each type possibly having a different weight. This is a more realistic approach for modeling many important networks in the real world which feature links of varying strength. For example, one early famous social network paper by Granovetter (1973) contrasted *weak ties* (e.g. between people who interact less than once per year) with *strong ties* (e.g. between people who interact at least twice per week), which are thought to indicate quantitatively and/or qualitatively different types of social connections. An emerging literature has begun to use data from online networks to predict tie strength, e.g. with tie strengths that are self-reported by human[4]. In this paper, we

[1] Brin and Page, 1998
[2] Kwak, Lee, Park, and Moon, 2010
[3] Elliott, Golub, and Jackson, 2014
[4] Gilbert and Karahalios, 2009

take a complementary approach and attempt to recover the tie strengths that are revealed via the propagation of actual influence along a network[5].

Our approach builds on PageRank, a popular algorithm for assessing the propagation of influence along a network[6], and has three key elements. First, we introduce Edge-Type Weighted Page Rank (EWPR), an extension of PageRank that allows for multiple edge types and possibly different edge-type weights. Second, we present algorithms (gradient descent and grid search) for finding the edge-type weight vector that best describes a given set of "influence" scores (e.g. such as those generated by EWPR itself with a particular set of weights) under the maintained hypothesis that influence operates as if characterized by EWPR with known edge types. (EWPR should not be confused with the "Weighted Page Rank" algorithm proposed by Xing and Ghorbani (2004), which proposes a different extension of PageRank that uses weighting based on inlinks and outlinks.) This "structural" approach to recovering network parameters is widely used by economists,[7] although to the best of our knowledge it has not yet been combined with PageRank-like algorithms from computer science. Third, modern networks are often massive, and frequently have very large numbers of edges. In such cases, a single-machine approach may be infeasible. We demonstrate how to use the high-speed cluster programming framework Spark to implement our approach over a distributed cluster.

Our approach has a wide range of potential applications. Networks of individuals and firms are common in many financial markets[8]; in such networks, some links are "social" (e.g. following a person on Twitter with the goal of receiving interesting information about them) while others are "economic" (e.g. making a financial investment with the goal of receiving a profit from them). Using our approach, an analyst could quantitatively assess the relative value of a social link vis-à-vis an economic link using only knowledge about the graph (including the edge types), and a given set of "influence" scores. For example, the online platform AngelList matches angel investors with start-ups who are seeking capital; such start-ups have subsequently raised over \$2.9 billion in venture capital and exit money[9]. On AngelList, it is possible for an investor to follow a start-up (creating a social link), invest in a start-up (creating an economic link), or neither. It seems plausible that an "invest" link might represent stronger influence than a "follow" link, but how much more is an open question. A

[5] For discussion on the relative strengths of self-reported versus revealed model primitives, see Beshears, Choi, Laibson, and Madrian, 2008
[6] Brin and Page, 1998
[7] e.g., Currarini, Jackson, and Pin, 2010
[8] Hochberg, Ljungqvist, and Lu, 2007
[9] Bernstein, Korteweg, and Laws, 2014

good answer to this question might help improve the design of systems in which people make financial decisions, i.e. organizations and markets.

2. Model Formulation

2.1 Structural Estimation - Weighted Graph

First, we define an *edge-type weighted graph* as a weighted graph $G = (V, E(e,t), \omega) \in \mathcal{G}^w$, where

V is the set of vertices with $|V| = n$,

E is the set of edges with $|E| = m$, each edge e having a type attribute $t \in \mathcal{T} = \{1, 2, \dots, T\}$ where $\mathcal{T}$ is the set of possible edge types, and

$\omega = (\omega_1, \dots, \omega_T) \in \mathbb{R}_+^{\mathrm{T}}$ is a weight vector that maps each edge type t to a weight ω_t, and hence assigns the weight of each edge through its edge-type.

We define the structural estimation problem in the weighted graph context as follows. We are given the underlying graph, which includes the vertices, the edges, and the edge-type of each edge, but without information on the weight vector. We are also given a real-valued function $f\colon \mathcal{G}^w \to \mathbb{R}$ that assigns a value to every weighted graph. The objective is to identify the set of weights ω_{opt} that generates the weighted graph with a desired output from f. A frequent endeavour is to find the set of weights that obtain the minimum of f, i.e.,

$$\omega_{\mathrm{opt}} = \underset{\omega}{\operatorname{argmin}}\, f\big(G(V, E, \omega)\big)$$

In the following sections, we consider the specific problem instance of choosing f as the distance between the ordering obtained from the weighted PageRank scores and some pre-defined ordering.

2.2 Weighted PageRank

Before algorithms for finding the optimal weight vectors can be discussed, the process to calculate weighted PageRank scores for weighted graphs must be precisely outlined. For a edge-type weighted directed graph $G = (V, E(e,t), \omega)$, we define the weighted stochastic adjacency matrix $A \in \mathbb{R}^{n\times n}$ as: $A_{ji} = \frac{w_{ij}}{\sum_j w_{ij}}$, where $w_{ij} = \omega_{t_{ij}}$ is the weight for edge e_{ij}, if $e_{ij} \in E$ (i.e. if $v_i \to v_j$), and $w_{ij} = 0$ if $e_{ij} \notin E$.

Alternatively, for $t = 1, 2, \dots, T$, let $B^{(t)}$ be the unnormalized adjacency matrix of $G^{(t)} = (V, \{(e,\tau)|\tau = t\})$, the unweighted sub-graph of G with edges only of type t. Then, it follows that $A = \left(\omega_1 B^{(1)} + \cdots + \omega_T B^{(T)}\right)C$, where C is a diagonal matrix that serves as the normalization factor, i.e. its diagonal entries take on values $C_{ii} = \sum_t \omega_t \left(\sum_j \left[B^{(t)}\right]_{ij}\right)$. Using this representation provides insights into the mechanism under which altering the weight vector ω may influence the page rank scores. When all the weights have equal values, the weighted graph degenerates into a normal unweighted graph.

Following the same framework as in the original PageRank paper by Brin and Page (1998), we are seeking a PageRank score vector $r \in \mathbb{R}^{\mathrm{n}}$ such that $r = (1-\delta)/n + \delta A r$, where δ is the teleporting coefficient, commonly set to 0.85. Equivalently, r is an eigenvector that corresponds to eigenvalue $\lambda = 1$ for the matrix $M = \delta A + \frac{1-\delta}{n}\mathbb{1}$, where $\mathbb{1} \in \mathbb{R}^{n\times n}$ is the matrix of all 1's. Because M is also a left stochastic matrix, by the Perron–Frobenius theorem, $\lambda = 1$ is the unique largest eigenvalue of M. On this basis, it is justified to apply the power iteration on M to calculate the stationary probability vector, r.

2.3 Formal Statement

For any given set of weights ω, we desire to calculate how the PageRank scores obtained from this set of weights differs from the true PageRank scores obtained from the real weights ω^*. Ideally, we would establish a distance function between two PageRank scores, $h^*(r_1, r_2)$, and let $f\left(G(V,E,\omega^*)\right) = h^*\left(PR\left(G(V,E,\omega)\right), r^*\right)$ where $PR: \mathcal{G}^{\mathrm{w}} \to \mathbb{R}^n$ gives the weighted PageRank scores of a graph.

However, in reality, because ω^* are unknown, we may also not be able to observe the real PageRank scores r^*. Hence, we assume that it is possible to observe some ordering of the vertices that represents the ordering of the PageRanks scores with some Gaussian noise. Namely, we observe a vector $p^* = order(r^* + \eta)$, where $\eta \sim \mathcal{N}(0, \sigma_\epsilon^2 I)$. We assume that σ_ϵ^2 is small compared to $\mathrm{s}_{\mathrm{r}^*}^2$, the sample variance of the PageRank score values. (In Section 4 Simulations, we choose $\sigma_\epsilon^2/\mathrm{s}_{\mathrm{r}^*}^2$ to be 0.1 and 0.2).

We define a distance metric $h(p_1, p_2)$, $p_1, p_2 \in \mathcal{P} \subset \mathbb{N}^n$, on the ordering space $\mathcal{P}$ = the set of all permutations of $(1, 2, \ldots, n)$. A possible choice of h is the Spearman's rank correlation distance. Another possibility is the L2-distance, given by

$$h(p_1, p_2) = \left(\sum_{i=1}^{n} ((p_1)_i - (p_2)_i)^2\right)^{1/2}$$

Given an edge-typed graph $G(V, E, \omega^*)$ (where ω^* is unknown) and some observed vertex ordering, p^*, we wish to find the optimal vector ω_{opt} that minimizes the distance in ordering between that obtained from the weighted PageRank scores of $G(V, E, \omega)$ and p^*. Using the Euclidean distance function, the edge-type weighted PageRank structural estimation problem becomes, given $G(V, E(e, t), \omega^*)$ (where ω^* is unknown) and an observed p^*, find:

$$\omega_{\text{opt}} = \underset{\omega}{\operatorname{argmin}} \left\| order\left(PR\big(G(V, E, \omega)\big)\right) - (p^*) \right\|_2$$

It is not immediately obvious the relationship between ω_{opt} and ω^*. However, the robust eigenvector theorem for stochastic matrices by Juditsky and Polyak (2012), when applied on ω^*, suggests that ω_{opt}^* will be in some sense close to ω^*. Section 4 especially considers this issue and runs simulations—the results are in favour of the closeness between ω_{opt}^* and ω^*.

3. Methodology

Since the structural estimation problem fixes the underlying structure of the graph, with perhaps a slight abuse of notation, we can consider $f(G(V, E(e, t), \omega)$ as instead a function $f(\omega)$: $\mathbb{R}^T \to \mathbb{R}$ that only takes a weight vector $\omega \in \mathbb{R}^T$ as input. Hence, we can thus discuss the gradient of f with respect to the weight vector ω.

However, it seems intractable to conduct a closed-form analysis of the gradient. This is because a change in weights results in a different linear combination of $\{B^{(t)}\}_{t=1,\ldots,T}$, which modifies the matrix M and hence r_ω, the eigenvector that corresponds to its largest eigenvalue $\lambda_1 = 1$. Because there is no intrinsic link on how modifying the matrix will change the orderings of the entries of its eigenvectors, it is sensible to resort to numerical methods in order to find the optimal weight vector ω_{opt}.

3.1 Single Machine Weighted PageRank

The consequence of having to resort to numerical methods is evident: a full weighted PageRank algorithm must be carried out each time the function f is evaluated at a different weight ω. This implies that any optimization effort to enhance the weighted PageRank algorithm will generate significant improvements.

The naïve approach is to follow the power method by iteratively applying the matrix M to some initial vector $v \in \mathbb{R}^N$ until at some step k, $M^k v$ is close to $M^{k-1}v$. Since we do not require exact PageRank scores, a more lenient criterion of "closeness" can be chosen as long as it reasonably ensures that the ordering of the entries of the resulting vector is unlikely to change significantly with further multiplications by M.

For this method, each multiplication requires $(2n-1)n$ flops, which is $\mathcal{O}(n^2)$. The total number of iterations requires an analysis of the eigendecomposition of the matrix M. Provided below is a brief analysis for the case when M is diagonalizable. Similar proof exists for a general M by considering the Jordan form.

Suppose $M = Q\Lambda Q^{-1}$ is its eigendecomposition, with $\Lambda = diag(\lambda_1, \ldots, \lambda_n)$ and $Q = [q_1|\ldots|q_n]$. We have $\lambda_1 \geq \cdots \geq \lambda_n$ as the eigenvalues of M and q_i is the eigenvector corresponding to eigenvalue λ_i. Then, in our case, $\lambda_1 = 1$ and $q_1 = r$. Suppose $v = \sum_i c_i q_i$ is chosen randomly, then we have $\mathbb{P}(\mathrm{c}_1 \neq 0) = 1$ and

$$M^k v = c_1(r + \frac{c_2}{c_1}(\lambda_2)^k q_2 + \cdots + \frac{c_n}{c_1}(\lambda_n)^k q_n$$

This means that the convergence rate, and hence the number of iterations, depends mostly on λ_2, and the convergence is geometric in the number of iterations k. Specifically, for an error tolerance ϵ defined as distance measured by some norm on the vector space $\mathbb{R}^N$, the number of iterations K needed to let the vector converge is

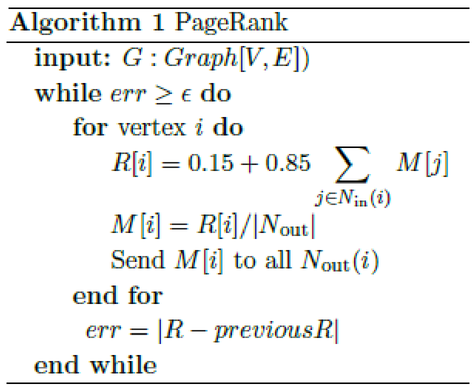
$$\mathcal{O}\left(\frac{\log(1/\epsilon) - cc_2/c_1}{\log(1/\lambda_2)}\right) \sim \mathcal{O}\left(\frac{\log(1/\epsilon)}{\log(1/\delta)}\right) \sim \mathcal{O}(\log(1/\epsilon))$$

In the above complexity analysis, we used the result from Haveliwala and Kamvar's theorem (2003) in *the Second Eigenvalue of Google Matrix Theorem*, whereby it is shown that $|\lambda_2| \leq \delta$. Hence, using the teleporting method (where typically δ is chosen to be 0.85) helps additionally in preventing a degeneratively slow convergence by disallowing λ_2 to go arbitrarily close to 1 (which would have resulted in slow convergence due to $\log(1/\lambda_2) \to 0^+$. Also note that an effort spent to choose a smart starting vector v by minimizing the ratio $\frac{\mathrm{c}_2}{\mathrm{c}_1}$ is rewarded minimally. Since λ_2 is out of our control, we will from here on refer to K as $\mathcal{O}(\log(1/\epsilon))$.

If A is a sparse matrix (which is usually the case for large graphs generated in real-life situations), we can improve the time complexity for each iteration within the PageRank multiplication. Instead of directly applying M, we update the vector using the original teleporting equation $v^{(k)} = (1-\delta)/n + \delta A v^{(k-1)}$. Suppose $|E| = m$ (i.e. A has m non-zero entries), then the multiplication requires $2m - n$ flops, which is $\mathcal{O}(m)$ assuming $m \gg n$.

3.2 Distributed Weighted PageRank – Pregel Framework

The Pregel implementation of PageRank proceeds as follows. In each superstep, each vertex updates its own value with a weighted sum of its neighbors' PageRanks, using received incoming messages. Then, each vertex sends an equal amount of its updated PageRank to each neighbor vertex, generating outgoing messages. This process repeats until convergence.



```
Algorithm 1 PageRank
  input: $G : Graph[V, E]$)
  while $err \geq \epsilon$ do
      for vertex $i$ do
          $R[i] = 0.15 + 0.85 \sum_{j \in N_{in}(i)} M[j]$
          $M[i] = R[i]/|N_{out}|$
          Send $M[i]$ to all $N_{out}(i)$
      end for
        $err = |R - previousR|$
  end while
```

(image source: CME 323: Distributed Algorithms and Optimization, Lecture 8, Reza Zadeh, Stanford University)

Suppose we have B machines with combiners. For each superstep, the number of messages sent is first upper-bounded by the total number of edges in the graph. Because with combiners, we first sum up the values corresponding to each key on each local machine before sending out the messages, the number of messages sent by each local machine is further upper-bounded by n. Hence the total communication cost for each superstep, which is the total number of messages sent, is $\min(m, nB)$.

In runtime, early supersteps will have a messageRDD size close to m, but as the number of steps increase, more vertices will *VoteToHalt()* and hence fewer and fewer messages will be sent as the vector $v^{(k)}$ approaches r. If we partition the graph by edges, we have each machine produce $\mathcal{O}(m/B)$ computations and sends out $\mathcal{O}(\min(m/B,\ n))$ messages.

The reduce size for each key is first upper-bounded by the maximum in-degree of the vertices in the graph. In extreme cases (such as a star graph), we can have the maximum in-degree of vertices in the order of $\mathcal{O}(m)$. However, if the graph is somewhat regular, we can expect the average reduce size is $\mathcal{O}(m/n)$. Further, the combiners help solve this curse of the last reducer problem for these extreme-case graphs. Because after combining, each vertex key will receive at most one message from each machine, the reduce size is further upper-bounded by B. Hence, we have that the reduce size as $\mathcal{O}(\min(m/n, B))$.

The analysis of number of iterations is analogous to that of weighted PageRank on a single machine, with one slight modification. Allowing the vertices to independently consider whether they will continue to update or halt implies that the actual calculated PageRank vector $\hat{v}^{(k)}$ is slightly different than $(1-\delta)/n + \delta A\hat{v}^{(k-1)}$ at every step. Having certain entries not updating with the rest of the vector will create a small $\mathcal{O}(\epsilon)$ at each step that potentially raises slightly the number of iterations until all vertices converge.

3.3 Single Machine Search Strategy

3.3.1 Grid Search

A simple search strategy is to perform a grid search on the entire weight space. In order for this to be feasible, we need to restrict the weights such that $\|\omega\|_1 = 1$. Note that this is justifiable because for any weight vector $\omega \in \mathbb{R}_+^{T}$, we have $f(\omega) = f(\omega/\|\omega\|_1)$ because $PR(\omega) = PR(\omega/\|\omega\|_1)$.

This search strategy is comprehensive, and as we shall see in section 4, provides a robust algorithm that recovers the true weights under noisy signals with high confidence. However, the major disadvantage of the grid search is its computationally intensive requirement. The total number of weighted PageRanks needed grows polynomially with the mesh steps, and exponentially with the number of dimensions, T. For a grid with mesh size 0.01, any number of edge types above single digits becomes computationally infeasible.

Explicitly, given a grid with mesh size α, the total number of weighted PageRanks performed is $(1/\alpha)^T$, and hence the total time complexity is $\mathcal{O}((1/\alpha)^T m \log(1/\epsilon))$.

3.3.2 Numerical Gradient Descent

The second proposed search strategy follows the gradient descent framework, with slight complication due to the lack of closed-form evaluation of the gradient. The basic gradient descent algorithm provides the update on the weight vector $\omega^{(l)}$ as

$$\omega^{(l+1)} = \omega^{(l)} - \gamma \, \nabla f\big(\omega^{(l)}\big)$$

where $\nabla f\big(\omega^{(l)}\big) = \left[\frac{\partial f}{\partial \omega_1}, \dots, \frac{\partial f}{\partial \omega_T}\right]^T_{\omega=\omega^{(l)}}$ is the gradient of the function f, and γ is the step size scaling factor chosen at each iteration l. Note that to avoid confusion, l enumerates the iteration in the search, each of which step gives a new weight vector after one or multiple PageRank algorithms. In contrast, k enumerates the iteration number in a single step within one PageRank algorithm – which is called a superstep in the distributed setting.

Because we cannot compute the closed-form of ∇f analytically, a numerical estimation approach appears appropriate. At each new point in the weight space, evaluating f requires running a full weighted PageRank—$\mathcal{O}(m \log(1/\epsilon))$—and a sort over all the PageRank scores to compute the ordering—$\mathcal{O}(n \log n)$ using TimSort, for instance. Hence, evaluating f incurs $\mathcal{O}(m \log(1\backslash\epsilon) + n \log n)$ cost.

To cut down computation costs, instead of using a central difference estimation method, we apply the forward difference method, obtaining

$$\frac{\partial f}{\partial \omega_t}\big(\omega^{(l)}\big) = \frac{f\big(\omega^{(l)} + \alpha e_t - \alpha e_T\big) - f\big(\omega^{(l)}\big)}{\alpha}$$

where $e_t = [0, \dots, 1, \dots, 0]$ is the unit vector in the t-th dimension. Due to the fact of the constraint $\|\omega\|_1 = 1$, we slightly modify the original gradient descent equation by updating the first $T-1$ components of the weight vector according to the equation, and the last component by $\omega_T^{(l+1)} = 1 - \sum_{t=1}^{T-1} \omega_t^{(l+1)}$. The forward difference method requires T PageRank computations, once for evaluation of f at the current weights and $T-1$ times to calculate the $T-1$ partial derivatives.

Hence, for one gradient descent to converge, the computation cost is $\mathcal{O}(T(m\log(1/\epsilon) + n\log n)L)$, where L is the total number of steps for the gradient descent to reach a local minimum, which depends on the condition of the matrix M. For a strictly convex function, we will be done after one gradient descent. However, since our function f is only piecewise convex (due to the discrete nature of orderings, as will be seen in section 4), retrieving the global minimum requires multiple runs of gradient descent with different initial guesses $\omega^{(0)}$. Letting S be the number of initializations we provide. Then the total computation cost is $\mathcal{O}(ST(m\log(1/\epsilon) + n\log n)L)$.

Comparing the gradient descent with the grid search strategy, we see that because the gradient descent is no longer exponential in T, it makes graphs with larger number edge-types much more tractable.

3.4 Distributed Search Strategies

An algorithm for the weights recovery problem in the distributed setting follows immediately from 3.2 and 3.3. One can distribute the computation of each single PageRank using the Pregel framework, and run the search strategy in sequence. One slight improvement is to also perform a parallel sort on the weighted PageRank's scores, which is needed for each evaluation of f at each set of points.

4. Simulation Results

We run simulations on generated graphs for two main purposes. First, generating random graphs with known true weight vector ω^* reveals the extent to which our proposed structural estimation framework is at all justified. Second, varying structural parameters of the graph (e.g. number of edge types) provides experimental run time estimates, and hence perhaps insights into rate of convergence and time complexity issues.

4.1 Weighted vs. Unweighted PageRank

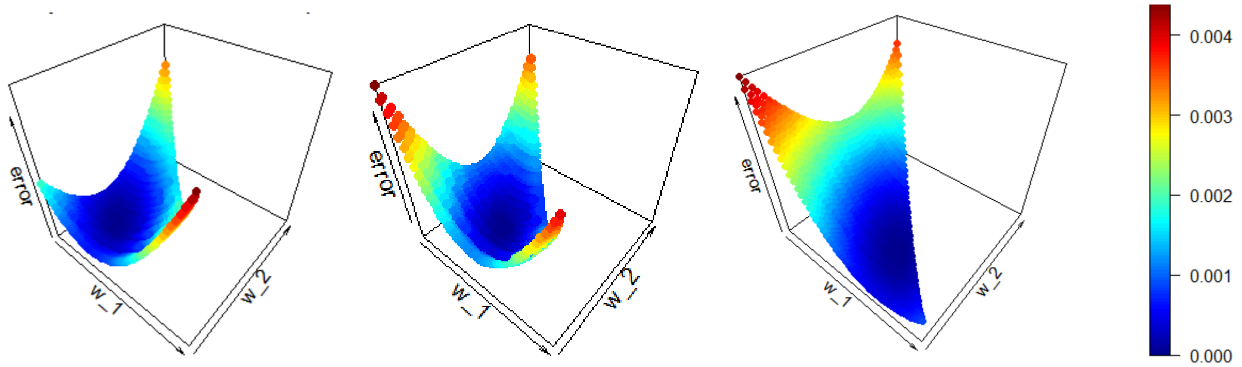
We evaluate the benefit of using weighted PageRank instead of its unweighted counterpart. We expect this benefit to be most significant if the true graph in general has high weights for edges going into a low in-degree vertex, and low weights for edges going into a high in-degree vertex.

A graph with 600 vertices is generated randomly by assigning a fixed probability of a directed edge forming from one vertex to another in each graph. To maintain some structural control over the graphs, we assign the edge-types of the graph in the following manner.

1. Sort the vertices by in-degrees, and assign each vertex a "vertex-type" by its quartile.
2. For each vertex-type, define a probability distribution over the edge-types an edge may be assigned. For graphical reasons, we restrict to 3 edge-types. We skew the distribution in favor for edge-type 3 for vertex-type with large in-degrees and edge-type 1 for vertex-type with small in-degrees.
3. Assign each edge its edge-type by drawing from the distribution provided by its destination vertex.

Next, we calculate the true weighted PageRank scores by providing three instances of the true weight vector ω^*. Then we do a grid search for each using the true PageRank ordering without perturbing the scores by Gaussian noise. The three graph f_1, f_2, f_3 are generated below.

Because Gaussian noises are not included in this simulation, it is expected that we obtain the global minimum at the true weight vectors. This can be seen in the graphs that the minimum of the surface lies very close in the region where the true weight vector lies. What is surprising, however, is that these graphs are smooth and strictly convex. This could be due to the fact that each perturbation of the matrix is renormalized so that the matrix remains stochastic (and hence the largest eigenvalue λ_1 is always 1 and is unique, which in turn guarantees the unique eigenvector that corresponds to this largest eigenvalue). Also, having a rank 1 teleporting matrix addition $\frac{1-\delta}{n}\mathbb{1}$ may also improve the stability of the eigenvector calculation (in addition to bounding the second largest eigenvalue λ_2.



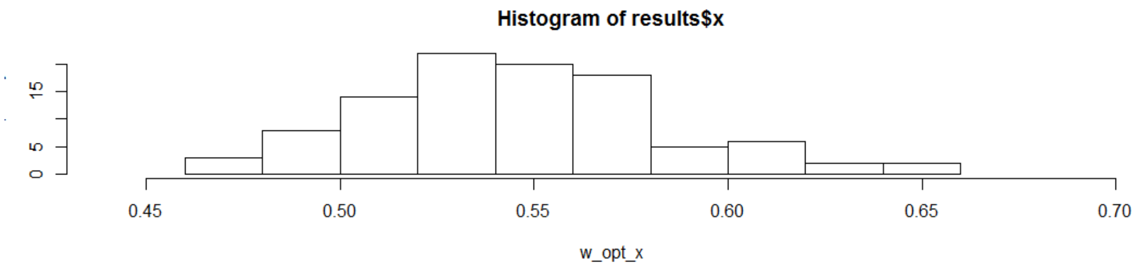
$f_1: \omega^* = (0.2, 0.3, 0.5)$ $f_2: \omega^* = (0.33, 0.33, 0.34)$ $f_3: \omega^* = (0.6, 0.3, 0.1)$

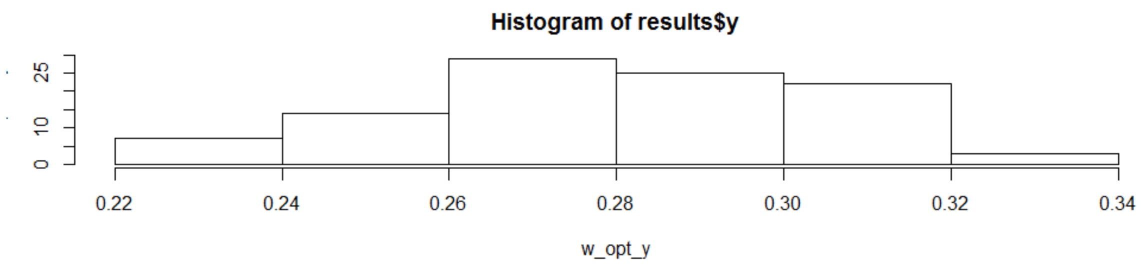
An interesting observation is that as the true edge-type weights gives higher emphasis on type 1 edges (which, as dictated by the method of experiment described above), we observe that the error obtained by using unweighted PageRank (where $\omega_1 = \omega_2 = 0.33$) increases significantly: when we have the true weight vector as $\omega^* = (0.6, 0.3, 0.1)$, the result from the unweighted PageRank and the weighted PageRank are most distinguishable. In other words, weighted PageRank are most valuable in terms of accuracy when we believe the value of scarcity—edges with destination vertices of high in-degrees are valued as lower link activities while edges with destination vertices of low in-degrees are valued as "rare" and hence more important link activities. If scarcity is expected, the unweighted PageRank will overvalue importance of nodes with high in-degrees and undervalue nodes with low in-degrees by assuming all edge weights are equal.

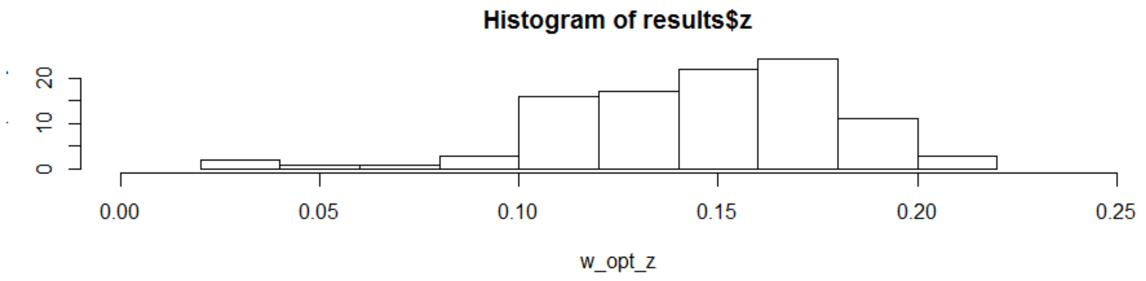
4.2 Optimal Weights Retrieval

The second simulation tests the robustness of the proposed algorithm in presence of Gaussian noises. Specifically, because in real life models only a noisy ordering of the PageRank scores can be observed, the effect of the permuted ordering due to these noises must be tested before a conclusion can be reached on whether it can be claimed that the optimal edge-type weight vector (obtained by minimizing the distance with respect to this noisy ordering) is in fact close to the true hidden weight vector.

Again, as in section 4.1, edge types are first assigned to each edge. Then, we assign a true weight vector $\omega^* = (0.5714,\ 0.2857,\ 0.1429) \sim \left(\frac{2^3+2^4}{2}, \frac{2^2+2^3}{2}, \frac{2^1+2^2}{2}\right)$. We run a Monte Carlo simulation and added Gaussian noises 100 times, with reasonable signal to noise ratio (choosing $\sigma_\epsilon = 0.3 s_{r^*}$, where s_{r^*} is the sample standard deviation of the true PageRank scores). Then, for each of the 100 simulations, we calculated the optimal weight vector that minimized f using our numerical gradient descent algorithm. The histograms of the estimated optimal weights are presented below.







It is not surprising that there are fluctuations in the recovered optimal weight vector. The average optimal weight vector obtained is $\overline{\omega}^* = (0.5754,\ 0.2848,\ 0.1398)$. The 95% confidence intervals from this Monte Carlo simulation are:

95% CI for ω_1^*: $(0.5677,\ 0.5831)$

95% CI for ω_2^*: $(0.2797,\ 0.2899)$

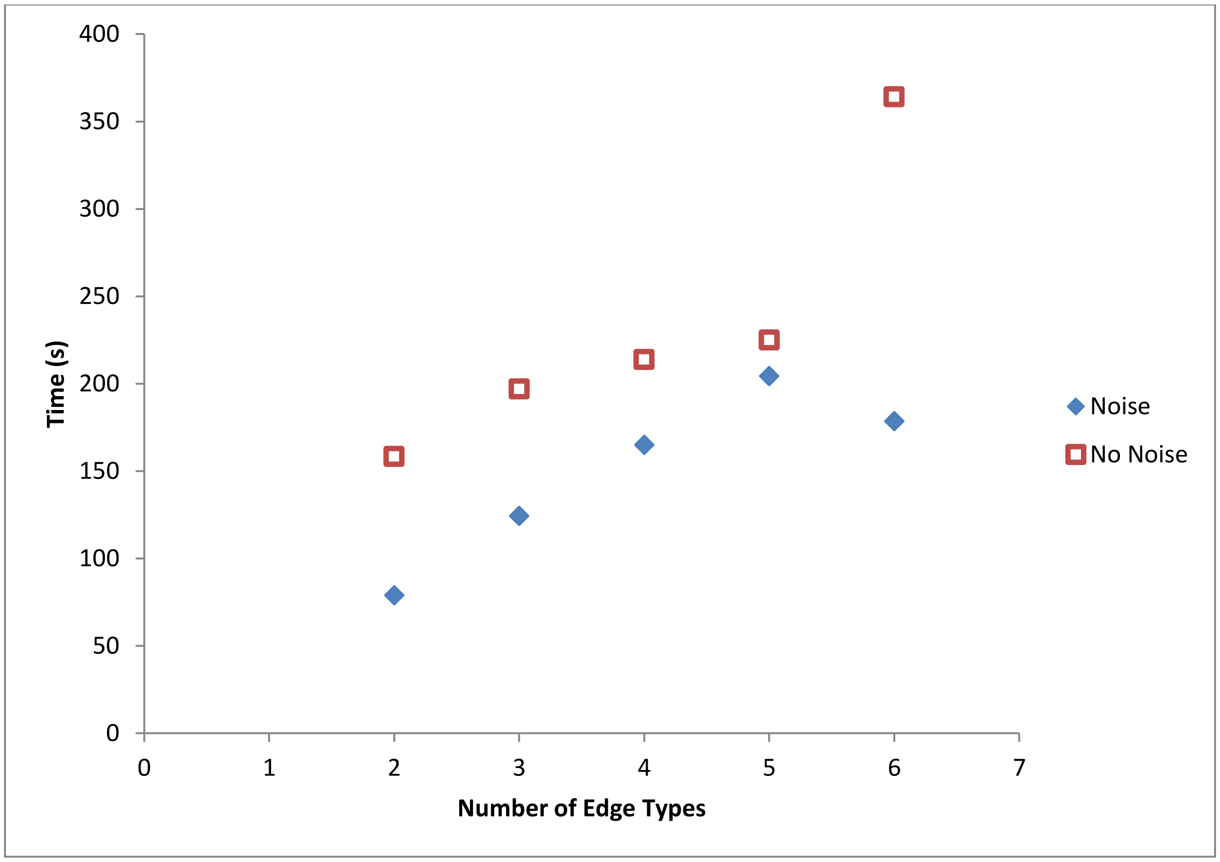
95% CI for ω_3^*: $(0.1325,\ 0.1471)$

Recall that the true edge-type weight vector is $\omega^* = (0.5714,\ 0.2857,\ 0.1429)$. Hence, in the simulation, our 95% confidence interval using optimal weights Monte Carlo simulation covered the true weight vector. Hence, this suggests that the algorithm is robust under noisy PageRank orderings. In real-life data analysis, instead of trusting on result of one optimal weight vector, bootstrapping techniques can be used on the data to create a bootstrapping CI that gives an interval estimate of the true weight vector instead of a point estimate.

4.3 Experimental Runtime Performance

Finally, we present experimental runtime performance results for our numerical gradient descent algorithm. (We use here a slightly different minimization objective – squared difference in PageRank scores rather than squared difference in PageRank order – because our Spark setup (4 laptop cores, 4 GB memory) was able to process the former metric, but not the latter, in a reasonable time for graphs of a size worth distributing.) Our analysis in 3.3.2

suggests that runtime should be linear in the number of parameters (i.e. edge type weights). This appears to be approximately true for simulated Erdos-Renyi random graphs with 600 nodes, 20% edge creation probabilities, and base edge-type weights set equal to 1 for edge type 1, 2 for edge type 2, etc.



5. Conclusion

In this paper, we have described and evaluated a distributed approach for recovering graph edge-type weights that are revealed through orders generated by a specific type of influence propagation, Edge-Type Weighted PageRank. Our preferred implementation combines numerical gradient descent with PageRank iterations using the Pregel framework. In future work, we are interested in developing techniques to further enhance performance by improving convergence results, reducing the number of iterations, and considering more self-communicative search strategies.

References

Bernstein, S., Korteweg, A. G., & Laws, K. (2014). Attracting Early Stage Investors: Evidence from a Randomized Field Experiment. *Rock Center for Corporate Governance at Stanford University Working Paper*, (185), 14-17.

Beshears, J., Choi, J. J., Laibson, D., & Madrian, B. C. (2008). How are preferences revealed?. *Journal of Public Economics*, *92*(8), 1787-1794.

Brin, S., & Page, L. (1998). The anatomy of a large-scale hypertextual Web search engine. *Computer networks and ISDN systems*, *30*(1), 107-117.

Currarini, S., Jackson, M. O., & Pin, P. (2010). Identifying the roles of race-based choice and chance in high school friendship network formation.*Proceedings of the National Academy of Sciences*, *107*(11), 4857-4861.

Elliott, M., Golub, B., & Jackson, M. O. (2014). Financial Networks and Contagion. *American Economic Review*, *104*(10), 3115-53.

Gilbert, E., & Karahalios, K. (2009, April). Predicting tie strength with social media. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 211-220). ACM

Granovetter, M. S. (1973). The strength of weak ties. *American journal of sociology*, 1360-1380.

Haveliwala, T., & Kamvar, S. (2003). The second eigenvalue of the Google matrix. *Stanford University Technical Report*.

Hochberg, Y. V., Ljungqvist, A., & Lu, Y. (2007). Whom you know matters: Venture capital networks and investment performance. *The Journal of Finance*,*62*(1), 251-301.

Juditsky, A., & Polyak, B. (2012). Robust eigenvector of a stochastic matrix with application to PageRank. *arXiv preprint arXiv:1206.4897*.

Kwak, H., Lee, C., Park, H., & Moon, S. (2010, April). What is Twitter, a social network or a news media?. In *Proceedings of the 19th international conference on World wide web* (pp. 591-600). ACM.

Xing, W., & Ghorbani, A. (2004, May). Weighted pagerank algorithm. In*Communication Networks and Services Research, 2004. Proceedings. Second Annual Conference on* (pp. 305-314). IEEE.