Edge-Weighted Personalized PageRank: Breaking A Decade-Old Performance Barrier

Wenlei Xie, David Bindel, Alan Demers, Johannes Gehrke
Cornell University
Ithaca, NY
{wenleix, bindel, ademers, johannes}@cs.cornell.edu

ABSTRACT

Personalized PageRank is a standard tool for finding vertices in a graph that are most relevant to a query or user. To personalize PageRank, one adjusts node weights or edge weights that determine teleport probabilities and transition probabilities in a random surfer model. There are many fast methods to approximate PageRank when the node weights are personalized; however, personalization based on edge weights has been an open problem since the dawn of personalized PageRank over a decade ago. In this paper, we describe the first fast algorithm for computing PageRank on general graphs when the edge weights are personalized. Our method, which is based on model reduction, outperforms existing methods by nearly *five orders of magnitude*. This huge performance gain over previous work allows us — for the very first time — to solve learning-to-rank problems for edge weight personalization *at interactive speeds*, a goal that had not previously been achievable for this class of problems.

Categories and Subject Descriptors

H.2.8 [**Database applications**]: Data mining

Keywords

Personalized PageRank; Model Reduction

1. INTRODUCTION

PageRank was first proposed to rank web pages [13], but the method is now used in a wide variety of applications such as object databases, social networks, and recommendation systems [9, 7, 20, 22]. In the PageRank model, a random walker moves through the nodes in a graph, at each step moving to a new node by transitioning along an edge (with probability α) or by "teleporting" to a position independent of the previous location (with probability $(1-\alpha)$). That is, the vector $x^{(t)}$ representing the distribution of walker locations at time t satisfies the discrete-time Markov process

$$x^{(t+1)} = \alpha P x^{(t)} + (1-\alpha)v \tag{1}$$

where the column-stochastic matrix P represents edge transition probabilities and the vector v represents teleportation probabilities. The PageRank vector x is the stationary vector for this process, i.e. the solution to

$$Mx = b, \text{ where } M = (I - \alpha P), \quad b = (1-\alpha)v. \tag{2}$$

The entries of x represent the importance of nodes.

Standard PageRank uses just graph topology, but many graphs come with weights on either nodes or edges. For example, we may weight nodes in a web graph by how relevant we think they are to a topic [26], or we might assign different weights to different types of relationships between objects in a database [9]. Weighted PageRank methods use node weights to bias the teleport vector v [29, 26, 8] or edge weights to bias the transition matrix P [44, 30, 9, 43]. Through these edge and node weights, PageRank can be *personalized* to particular users or queries. Concretely, in *node-weighted personalized PageRank*, we solve

$$Mx(w) = (1-\alpha)v(w), \quad w \in \mathbb{R}^d$$

where w is a vector of personalization parameters. In *edge-weighted personalized PageRank*, we solve

$$M(w)x(w) = (1-\alpha)v, \quad w \in \mathbb{R}^d.$$

For example, the personalization vector can specify the topic preference for the query, so the random walker will teleport to nodes associated with preferred topics (node-weighted personalized PageRank) or move through edges associated with preferred topics more frequently (edge-weighted personalized PageRank). The parameters are determined by users, queries, or domain experts [26, 9, 43, 42], or tuned by machine learning methods [34, 3, 7, 20, 21].

Despite the importance of both edge-weighted and node-weighted personalized PageRank, the literature shows an unfortunate dichotomy between the two problems. There exists a plethora of papers and efficient methods for node weight personalization (as we will discuss in Section 2), but not for the much harder problem of edge weight personalization! Yet many real applications are in dire need of edge-weighted personalized PageRank. For example, together with machine learning methods, edge-weighted personalized PageRank improves ranking results by incorporating user feedback [34, 7, 20, 21]; but training with existing methods takes hours. In TwitterRank [43], weighting edges by the similarity of user's topical interests gives better results

KDD'15, August 10-13, 2015, Sydney, NSW, Australia.
© 2015 ACM. ISBN 978-1-4503-3664-2/15/08 ...$15.00.
DOI: http://dx.doi.org/10.1145/2783258.2783278.

than just weighting the nodes according to those same interests. However, because of their inability to compute edge-weighted personalized PageRank online, the TwitterRank authors recommend simply taking a linear combination of topic-specific PageRank vectors instead of solving the "correct" problem of edge-weighted personalized PageRank.

In early work on the ObjectRank system, the authors mention the flexibility of personalizing edge weights as an advantage of their approach, but they do not describe a method for doing this fast. In subsequent work, the computation was accelerated by heuristics [41, 42], but it was still too slow. The later ScaleRank work [28] tried to address this issue, but only applies to a limited type of graph. Thus, despite its importance and serious attempts to address performance concerns, efficiently computing edge-weighted personalized PageRank remains an open problem.

In this paper, we introduce the first truly fast method to compute $x(w)$ in the edge-weighted personalized PageRank case. Our method is based on a novel *model reduction* strategy, where a reduced model is constructed offline, but it allows fast evaluation online. The speed of our new approach enables exciting interactive applications that have previously been impossible; for example, in one application described in Section 5, the reduced model is nearly *five orders of magnitude* faster than state-of-the-art algorithms. This is very important for standard algorithms with cost proportional to the graph size, but reduced models also speed up sublinear approximation algorithms for localized PageRank problems. Thus a variety of applications can benefit from the techniques that we describe.

We discuss some preliminaries in Section 2, then turn to the three main contributions of the paper in the following sections. In Section 3, we describe the first scalable method for edge-weighted personalized PageRank by applying the idea of *model reduction*. We also show how to use common, simple parameterizations to make our methods even faster, and we reason about cost/accuracy tradeoffs in model reduction for PageRank. In Section 4, we show how fast reduced models enable new applications, such as interactive learning to rank, that were not previously possible. And in a thorough experimental evaluation in Section 5, we show that our method outperforms existing work by nearly *five orders of magnitude*. Finally, we discuss related work in Section 6, and make some concluding remarks in Section 7.

2. PRELIMINARIES

In standard (weighted) PageRank, we consider a random walk on a graph with weighted adjacency matrix $A \in \mathbb{R}^{n\times n}$, in which a nonzero entry $A_{ji} > 0$ represents the weight on an edge from node j to node i. The weighted out-degree of node j is $d_i = \sum_j A_{ji}$, and we define the degree matrix D to be a diagonal matrix with diagonal entries d_j. Assuming no sinks (nodes with zero outgoing weight), the transition matrix for a random walk on this weighted graph is $P = AD^{-1}$. If the graph has sinks, one must define an alternative for what happens after a random walker enters a sink. For example, the walker might either stay put or teleport to some node in the graph chosen from an appropriate distribution.

When v is a dense vector (the *global* PageRank problem), the standard PageRank algorithm is the iteration (1), which can be interpreted as a random walk, a power iteration for an eigenvalue problem, or a Richardson or Jacobi iteration for (2) [22]. One can use more sophisticated iterative methods [32, 10], but (1) converges fast enough for many offline applications [22]. All these repeatedly multiply the matrix P by vectors, and therefore have arithmetic cost (and memory traffic) proportional to the number of edges in the graph. Hence, these methods are ill-suited to interactive applications for large graphs [8].

There are two standard methods to approximate PageRank online in time sublinear in the number of edges. The first case is when the teleportation vector v can be well approximated as a linear combination of a fixed set of reference distributions, i.e. $v \approx Vw$ where $V \in \mathbb{R}^{n\times d}$ and $w \in \mathbb{R}^d$; see [29, 26]. For example, in web ranking, the jth column of V may indicate the importance of different nodes as authorities to a reference topic j, and the weight vector w indicates how interesting each topic is to a particular ranking task. In this case, the solution to the PageRank problem is

$$x(w) = (1-\alpha)M^{-1}Vw. \tag{3}$$

If $M^{-1}V$ is precomputed offline by solving d ordinary PageRank problems, then any element of $x(w)$ can be reconstructed in $O(d)$ time, while the entire PageRank vector can be computed in $O(nd)$ time. Because some nodes are unimportant to almost any topic, it may be unnecessary to compute every element of the PageRank vector.

We also can estimate PageRank in sublinear time when the restart vector v is sparse. In a common case that we refer to as *localized PageRank*, the vector v is one for an entry corresponding to a particular vertex and zero in all other components. Since the random surfer often returns to this vertex, the localized PageRank vector reveals properties of the region around this vertex. The Monte Carlo method [8] and Bookmark-Coloring Algorithm (BCA) [11, 4] estimate the PageRank values for the important nodes close to the target vertex, which are often the most important components of the PageRank vector in recommendation systems. Though these PageRank values can be computed with sub-second latency for a specific set of parameters, in an application requiring frequent recomputation for different users and queries, the total latency may still be significant.

Unfortunately, none of these methods apply to fast computation of edge-weighted personalized PageRank. We will discuss how to quickly approximate edge-weighted personalized PageRank via model reduction in the following section.

3. MODEL REDUCTION

Model reduction is a general approach to approximating solutions of large linear or nonlinear systems. Applied to PageRank, the key steps in model reduction are:

1. Observe that $x(w)$ often lies close to a low-dimensional space. We build a basis for a k-dimensional reduced space in a data-driven way using the POD/PCA method on PageRank vectors for a sample of parameters, as we describe in Section 3.1
2. To pick an approximation in a k-dimensional reduced space, we need k equations. We describe how to choose these equations (offline) in Section 3.2, and how to form the reduced system of equations quickly (online) in Section 3.3–3.4.
3. After we solve the reduced system, we reconstruct the PageRank vector from the coordinates in the reduced space. We often do not need the full PageRank vector, and can compute a part of the vector at lower cost.

In the experiments in Section 5, we achieved good accuracy with reduced spaces of dimension $k \approx 100$.

3.1 Reduced Space Construction

The basic assumption in our methods is that for each parameter vector $w \in \mathbb{R}^d$, the PageRank vector $x(w)$ is well approximated by some element in a low-dimensional linear space (the trial space). We construct the trial space in two steps. First, we compute PageRank vectors $\{x^{(j)}\}_{j=1}^r$ for a sample set of parameters $\{w^{(j)}\}_{j=1}^r$. Then we find a basis for a k-dimensional reduced space $\mathcal{U}$ (typically $k < r$) containing good approximations of each sample $x^{(j)}$.

In our work, we choose the sample parameters $w^{(j)}$ uniformly at random from the parameter space. This works well for many examples, though a low-discrepancy sequence [33] might be preferable if the sample size r is small. Sparse grids [14] or adaptive sampling [35] might yield even better results, and we leave these to future work.

Once we have computed the $\{x^{(j)}\}$, we construct the reduced space $\mathcal{U}$ by the proper orthogonal decomposition (POD) method, also known as principal components analysis (PCA) or a truncated Karhunen-Loève (K-L) expansion [37, 36]. We form the data matrix $X = [x^{(1)}, x^{(2)}, \cdots x^{(r)}] \in \mathbb{R}^{n\times r}$ and compute the "economy" SVD

$$X = U^X \Sigma (V^X)^T$$

where $U^X \in \mathbb{R}^{n\times r}$ and $V^X \in \mathbb{R}^{r\times r}$ are matrices with orthonormal columns and Σ is a diagonal matrix with entries $\sigma_1 \geq \cdots \geq \sigma_r \geq 0$. The best k-dimensional space for approximating X under the 2-norm error is the range of $U = \begin{bmatrix} u_1^X & \dots & u_k^X \end{bmatrix}$. The singular value σ_{k+1} bounds $\min_y \|Uy - x^{(j)}\|_2$ for each sample $x^{(j)}$. If σ_{k+1} is small, the space is probably adequate; if σ_{k+1} is large, the trial space may not yield good approximations.

3.2 Extracting Approximations

Given a trial space and an online query parameter w, we want an element of the trial space that approximates $x(w)$. That is, if $U = [u_1, u_2, \cdots, u_k] \in \mathbb{R}^{n\times k}$, we want an approximate PageRank vector of the form $\tilde{x}(w) = Uy(w)$ for some $y(w) \in \mathbb{R}^k$. All of the methods we describe can be expressed in the framework of Petrov-Galerkin methods; that is, we choose $y(w)$ to satisfy

$$W^T \left[M(w)Uy(w) - b\right] = 0$$

for some set of test functions W. We can also choose $y(w)$ to force the entries of $\tilde{x}(w)$ to sum to one, i.e.

$$\text{minimize} \|W^T \left[M(w)Uy(w) - b\right]\|^2 \text{ s.t. } \sum_j (Uy(w))_j = 1;$$

we solve this constrained linear least squares problem by standard methods [23, Chapter 6.2]. We consider two methods of choosing an approximate solution: *Bubnov-Galerkin* and the *Discrete Empirical Interpolation Method (DEIM)*.

In the Bubnov-Galerkin approach, we choose $W = U$; that is, the trial space and the test space are the same. Because U is a fixed matrix, we can precompute the system matrices that arise from the Bubnov-Galerkin approach for simple edge weight parameterizations. However, this approach is expensive for more complicated problems.

In the DEIM approach, we enforce a subset of the equations. That is, we choose y to mimimize the norm of a projected residual error:

$$\|\Pi \left[M(w)Uy(w) - b\right]\|^2,$$

where Π is a diagonal matrix that selects entries from some index set $\mathcal{I}$; that is,

$$\Pi_{ii} = \begin{cases} 1, & i \in \mathcal{I} \\ 0, & \text{otherwise.} \end{cases}$$

Equivalently, we write the objective in the minimization problem as

$$\|M_{\mathcal{I},:}(w)Uy(w) - b_{\mathcal{I}}\|^2$$

where $M_{\mathcal{I},:}(w)$ is the submatrix of $M(w)$ involving only those rows in the index set $\mathcal{I}$, and similarly with $b_{\mathcal{I}}$. If $|\mathcal{I}| = k$, we enforce k of the equations in the PageRank system; otherwise, for $|\mathcal{I}| > k$, we enforce equations in a least squares sense.

Minimizing this objective is equivalent to Petrov-Galerkin projection with $W = \Pi M(w)U$. Because W has few nonzero rows, only a few rows of $M(w)$ need ever be materialized, even when the parameterization is complicated. The key to this approach is a proper choice of the index set $\mathcal{I}$ of equations to be enforced.

Once the reduced system is solved and the coordinates in the reduced space $y(w)$ are located, we can reconstruct the approximate PageRank vector in the original space by $\tilde{x}(w) = Uy(w)$. This can be slow when the graph has millions of nodes, as it requires $O(kn)$ work. In practice, we usually do not require the whole PageRank vector, but only the PageRank values for a subset of the vertices, such as the nodes with high PageRank values, or the nodes associated with some keywords. Assuming the subset of vertices we are interested in is $\mathcal{S}$, we can compute the PageRank values for the vertices in $\mathcal{S}$ by $\tilde{x}_{\mathcal{S}}(w) = U_{\mathcal{S},:} \cdot y(w)$. The computation can be done with only $O(k|\mathcal{S}|)$ work.

3.3 Parameterization Forms

How $P(w)$ depends on w matters to how fast we can make different model reduction methods. For example, suppose $P(w) = A(w)D(w)^{-1}$, where the weighted adjacency matrix $A(w)$ has edge weights that are general nonlinear functions in w and $D(w)$ is the diagonal matrix of out-degree weights, as before. The DEIM approach would require that we compute the weight for all edges in $E' = \{(i,j) \in E : j \in \mathcal{I}\}$; and also all the edges $E'' = \{(i,j') \in E : \exists (i,j) \in E'\}$ in order to normalize. The expense of the Bubnov-Galerkin method in this case would be even worse: to form $U^T M(w) U$ we would need to multiply $P(w)$ by each of the k columns of U, at a cost comparable to running the PageRank iteration to convergence. We therefore consider two simple parameterizations for which the Bubnov-Galerkin and DEIM methods can be made more efficient.

In a *linear* parameterization, we assume

$$P(w) = \sum_{s=1}^m w_s P^{(s)}$$

where each $P^{(i)}$ is column stochastic and the weights w_i are non-negative and sum to one. For a linear parameterization, the Bubnov-Galerkin approach is fast, since

$$U^T M(w) U = U^T U - \alpha \sum_{s=1}^m w_s U^T P^{(s)} U,$$

and the (small) matrices U^TU and $U^TP^{(s)}U$ can be precomputed. The linear parameterization has been used several times in settings in which each edge has one of m different types [9, 42, 34]. In such settings, the parameterization corresponds to a generalized random walker model in which the walker first decides what type of edge to follow with probabilities given by the w_s, then decides between edges of that type. However, the model is limited in that it does not allow the probability of picking a particular edge type to vary across nodes.

In a *scaled linear* parameterization, $P(w) = A(w)D(w)^{-1}$ where the weighted adjacency $A(w)$ is

$$A(w) = \sum_{s=1}^{m} w_s A^{(s)}$$

and $D(w)$ is the diagonal matrix of outgoing weights

$$d_i(w) = \sum_{s=1}^{m} w_s d_i^{(s)}, \quad d_i^{(s)} = \sum_{j=1}^{n} A_{ji}^{(s)}.$$

The scaled linear parameterization corresponds to choosing each edge weight $A_{ji}(w)$ as a linear combination of edge features $A_{ji}^{(s)}$. For example, in a social network, the edge weights might depend on features like communication frequency, profile similarity, and the time since last communication [7]. Post topic similarity between users has also been adopted as an edge feature for sites such as Twitter [43]. Because of the normalization, $M(w)$ in the scaled linear case is not a linear function of w, and so the Bubnov-Galerkin system cannot be written as a linear combination of precomputed matrices. For DEIM in the scaled linear case, however, we can materialize only the subset of rows of $A(w)$ with indices in $\mathcal{I}$, since the out-degree weight vector needed for normalization is a linear combination of the weight vectors $d^{(i)}$, which we can precompute.

3.4 Choosing interpolation equations

As shown in Appendix A, if the columns of U are normalized to unit 1-norm, the key quantity in the error bound for DEIM is $\|(M_{\mathcal{I},:}U)^{\dagger}\|_1$. We do not want interpolation nodes that are always unimportant, since in this case $M_{\mathcal{I},:}U$ will have small elements, and this might suggest that we should choose "important" nodes according to the average of some randomly sampled PageRank vectors. However, this heuristic sometimes fails, as $M_{\mathcal{I},:}U$ can be nearly singular even if the elements of $M_{\mathcal{I},:}U$ are not nearly zero.

In the case $|\mathcal{I}| = k$, we want rows $M_{\mathcal{I},:}U$ that are maximally linearly independent. While an optimal choice is hard in general, this type of *subset selection* problem is standard in linear algebra, and a standard approach to selecting the most important rows is to apply the pivoted QR algorithm and use the pivot order as a ranking [23, Chapter 5]. However, if we were to apply pivoted QR to the rows of $M(w)U$ in order to choose the interpolation set, we would first have to compute $M(w)U$, which is again comparable in cost to computing PageRank directly. As an alternative, we evaluate $M(\tilde{w}^{(j)})U$ for one or more test parameters $\tilde{w}^{(j)}$, then apply pivoted QR to the rows of $Z = \begin{bmatrix} M(\tilde{w}^{(1)})U & \dots M(\tilde{w}^{(q)})U \end{bmatrix}$. By using more than one test parameter in the selection process, not only do we ensure that we are taking into account the behavior of M at more than one point, but we can choose as many as kq independent rows of the matrix Z.

Algorithm 1 Row subset selection via QR

```
In: $Z$ : $n \times m$ matrix
Out: $\mathcal{I}$: list of $m$ indices
def FIND-I($Z$)
  Initialize $I \leftarrow \varnothing$,   $Q \leftarrow 0^{n\times m}$,   $r \leftarrow 0^m$
  norm2 $\leftarrow$ squared norms of rows of $Z$
  for $k = 1$ to $m$ do
    $s \leftarrow \mathrm{argmax}_{1\le i\le n}\ \mathrm{norm}(i)$
    $\mathcal{I} \leftarrow \mathcal{I} \cup \{s\}$,   $Q_{k,:} \leftarrow Z_{n,:}$,   $r_k \leftarrow \sqrt{\mathrm{norm2}(s)}$
    $Q_{k,:} \leftarrow (Q_{k,:} - \sum_{l=1}^{k-1}\langle Q_{l,:}, Z_{n,:}\rangle Q_{l,:})/r_k$
    for $i = 1$ to $n$ do
      norm2$(i) \leftarrow$ norm2$(i) - \langle Z_{i,:}, Q_{k,:}\rangle^2$
    end for
  end for
  return $\mathcal{I}$
```

For scaled linear and nonlinear parameterizations, the cost of forming the Bubnov-Galerkin matrix $U^TM(w)U$ is linear in the size of the graph, and may even exceed the cost of an ordinary PageRank computation. In contrast, for DEIM methods, the cost of forming the reduced system is proportional to the number of incoming edges for the nodes with indices in $\mathcal{I}$ (in the scaled linear case) or those nodes plus all outgoing edges from those nodes (in the fully nonlinear case). Hence, there is a performance advantage to choosing low-degree nodes for the interpolation set $\mathcal{I}$. At the same time, choosing nodes with high in-degree for the set $\mathcal{I}$ may improve the accuracy of the reconstruction.

Note that the pivoted QR algorithm defines "utility" for the nodes, and it chooses the nodes with the highest utilities. The utility for vertex i is $\sqrt{\mathrm{norm2}(i)}$ described in Algorithm 1. We write the utility of vertex i as util(i) and the incoming degree as cost(i). We propose two methods to manage the trade-off between util(i) and cost(i): the cost-bounded pivot method and the threshold pivot method.

In the cost-bounded pivot method, a parameter C indicates the average cost we are willing to pay for each node. A straightforward heuristic would be choosing the nodes with highest util(i) with cost$(i) \le C$. However, this heuristic ignores all the nodes that exceed the cost budget. We can soften the budget constraint by choosing the nodes with highest util$(i)/\max(\mathrm{cost}(i), C)$. Alternatively, in the threshold pivot method, we require the node selected in each step to have utility higher than $\max_i \mathrm{util}(i)/F$. Among these nodes, the one with smallest cost is selected. The parameter F indicates the maximum ratio allowed between the highest utility and the utility of the selected node in each step. We will discuss the trade-offs of the two methods in the experiment section.

4. LEARNING TO RANK

We now describe an application enabled by fast PageRank approximations: learning to rank. In learning to rank, the goal is to learn the best values of the parameters that determine edge weights, based on training data such as user feedback or historic activities, as discussed in [7, 3]. Using surrogates, learning to rank becomes not only an interesting offline computation, but an online computation that could be done on a per-query basis.

As a concrete example, consider training data $T = \{(i_q, j_q)\}$, where each pair $(i, j) \in T$ indicates that i should be ranked higher than j. The optimization problem is

$$\min_w L(w) = \sum_{(i,j)\in T} l(x_i(w) - x_j(w)) + \lambda\|w\|^2, \qquad (4)$$

where $x_i(w)$ and $x_j(w)$ indicate the PageRank of nodes i and j for a parameter vector w. The loss $l(\cdot)$ penalizes violations of the ranking preference; popular choices include squared loss and Huber loss [7]. To minimize L by gradient-based methods, one needs both L and the derivatives

$$\frac{\partial L}{\partial w_s} = \sum_{(i,j)\in T} l'(x_i - x_j)\frac{\partial(x_i - x_j)}{\partial w_s} + 2\lambda w_s, \qquad (5)$$

To compute the derivatives of x, one uses the relation

$$M\frac{\partial x}{\partial w_s} = -\frac{\partial M}{\partial w_s}x; \qquad (6)$$

that is, each partial derivative requires solving a linear system involving the PageRank matrix $M(w)$.

We replace the PageRank vector $x(w)$ with an approximation $\hat{x}(w) = Uy(w)$, and seek to minimize the modified objective

$$\min_w \hat{L}(w) = \sum_{(i,j)\in T} l(\hat{x}_i(w) - \hat{x}_j(w)) + \lambda\|w\|^2. \qquad (7)$$

We write the component differences $\hat{x}_i(w) - \hat{x}_j(w)$ as $z_{ij}y(w)$ where $z_{ij} = U_{i,:} - U_{j,:}$ is the difference between two rows of U. The derivatives of $\hat{L}$ are then

$$\frac{\partial \hat{L}}{\partial w_s} = \sum_{(i,j)\in T} l'(z_{ij}y(w))z_{ij}\frac{\partial y(w)}{\partial w_s} + 2\lambda w_s. \qquad (8)$$

Depending on the size of the training set, one might form the z_{ij} vectors at the start of the optimization.

Once $y(w)$ has been evaluated for a given parameter vector w, the subsequent derivative computations can re-use factorizations, and so require less computation. For example, in the case of a linear parameterization and the Bubnov-Galerkin method, the derivatives of y satisfy

$$\tilde{M}\frac{\partial y}{\partial w_s} = -\frac{\partial \tilde{M}}{\partial w_s}y = \alpha\tilde{P}^{(s)}y, \qquad (9)$$

where $\tilde{M}(w) = U^T M(w) U$ is the same matrix that appears in the linear system for $y(w)$, while $\tilde{P}^{(s)} = U^T P^{(s)} U$ is one of the matrices that was precomputed in the offline phase. Though computing $y(w)$ requires $O(k^3)$ work to factor $\tilde{M}$ after it has been formed, the derivative computations can re-use this work, and involve only $O(k^2)$ additional work to form the right hand side and solve the resulting system.

For a scaled linear parameterization or nonlinear parameterization with the DEIM method, y satisfies the normal equations

$$\tilde{M}^T\tilde{M}y = \tilde{M}^T b_{\mathcal{I}} \qquad (10)$$

where $\tilde{M} = M_{\mathcal{I},:}U$. Differentiating (10) gives the formula

$$\tilde{M}^T\tilde{M}\frac{\partial y}{\partial w_s} = -\tilde{M}^T\left(\frac{\partial \tilde{M}}{\partial w_s}\right)y - \left(\frac{\partial \tilde{M}}{\partial w_s}\right)^T\tilde{r} \qquad (11)$$

where $\tilde{r} = \tilde{M}y - b_{\mathcal{I}}$ is the residual in the reduced least squares problem. Once we have computed $\tilde{M}$ and its derivatives, computing y takes $O(k^2|\mathcal{I}|)$ time, while additional solves to evaluate partial derivatives take $O(k|\mathcal{I}|)$ time. In many graphs, though, the dominant cost is the time to form $\tilde{M}$ and its derivatives.

5. EXPERIMENTS

Our experimental evaluation has three goals: First, we want to show that *global edge-weighted personalized PageRank can be computed interactively with model reduction.* As reported in Section 5.3, our model reduction methods can answer such queries in half a second, while the standard power method takes 20 seconds (DBLP graph) to 1 minute (Weibo graph). Second, we want to verify that *model reduction improves the performance of localized PageRank.* As reported in Section 5.4, our methods are usually 1-2 orders of magnitudes faster than BCA. Third, we want to demonstrate that model reduction enables *learning-to-rank at interactive speeds.* As we report in Section 5.5, our model reduction methods are up to nearly five orders of magnitudes faster than the full computation on a learning-to-rank application. We discuss our experimental setup and preprocessing in Sections 5.1 and 5.2.

5.1 Setup

Environment. We run all experiments on a computer with two Intel Xeon X5650 2.67GHz processors and 48GB RAM. We have implemented most of our methods in C++ using Armadillo [38] for matrix computations. For the LP solver required by the ScaleRank algorithm, we follow the original implementation and use the GLPK package [1]. A few routines are currently prototyped with SciPy [2].

Datasets. We run experiments on two graphs: the DBLP citation graph and the Weibo social graph. Table 1 shows their basic statistics. The DBLP graph, provided by ArnetMiner [40], has four types of vertices (paper, author, conference and venue) and seven edge types. We adopt the linear parameterization used in ObjectRank to assign edge weights [9, 28]. This graph is used for global PageRank and parameter learning.

The Weibo graph, released as part of the KDD Cup 2012 competition,[1] is a microblog dataset with subscription edges between users and text. In prior work [43], personalized PageRank with edge weights derived from topic analysis over user posts was used to rank topical authorities. In our experiments, we apply LDA to analyze the topics represented in user posts, following the approach in [12], with the number of topics set to 5, 10, and 20. We use both scaled-linear and linear parameterizations based on the user topic distributions. For the scaled-linear parameterization, we adopt a weighted cosine as the edge weight:

$$A_{ji}(w) = \sum_{s=1}^{m} w_s \cdot (\varphi_i)_s \cdot (\varphi_j)_s = \sum_{s=1}^{m} w_s A_{ji}^{(s)},$$

where φ_i denotes the topic distribution for user i and $A_{ji}^{(s)} \equiv (\varphi_i)_s \cdot (\varphi_j)_s$. We can also normalize the weighted adjacency matrix $A^{(s)}$ for a linear parameterization:

$$P^{(s)} = A^{(s)}(D^{(s)})^{-1}.$$

We use the Weibo graph in experiments for both global and localized PageRank. For localized PageRank, we run the experiments on 1000 randomly selected nodes, as different nodes can have slightly different running time and accuracy for both our methods and the baseline.

Baselines. We compare our methods to ScaleRank [28] for global PageRank and the Bookmark Coloring Algorithm

[1]`https://www.kddcup2012.org/c/kddcup2012-track1`

Table 1: Basic Statistics of Datasets

Dataset	# Vertices	# Edges	# Params
DBLP	3,494,258	18,515,718	7
Weibo	1,944,589	50,655,130	5, 10, 20

(BCA) [11, 4] for localized PageRank. We discuss these methods in more detail in Section 6. For global PageRank, ScaleRank [28] is the only previous work we know to efficiently compute edge-weighted personalized PageRank. Like our approach, ScaleRank has an offline phase that samples the parameter space, and an online phase in which queries are processed interactively. We only report the performance of ScaleRank on the DBLP graph, as it is ill-suited for general graphs. We set the parameters according to the original paper. For localized PageRank, prior methods approximate the PageRank of the top-k nodes in sub-second time without preprocessing. We compare our model reduction methods with a variant of BCA proposed in [4]. For the BCA termination threshold ϵ, which controls the trade-off between running time and accuracy, we use $\epsilon = 10^{-8}$; this leads to Kendall τ values of around 0.01.

Metrics. We evaluate both how well our approximate PageRank values match the true PageRank values and the distance between the induced rankings. We denote the exact and approximate PageRank vectors as x and $\tilde{x}$, respectively. We measure accuracy of the approximate PageRank values by the normalized L1 distance. The normalized L1 distance on the evaluation set $\mathcal{S}$ is defined as

$$NL_1(x, \tilde{x}, \mathcal{S}) = \|x_{\mathcal{S}} - \tilde{x_{\mathcal{S}}}\|_1 / \|x_{\mathcal{S}}\|_1.$$

For global PageRank, $\mathcal{S}$ contains all the nodes, while for localized PageRank, $\mathcal{S}$ is the exact top-100 set. To evaluate ranking, we adopt Kendall's τ, the *de facto* standard for ranking measurement [31]. Denote the number of concordant pairs and discordant pairs as n_C and n_D, respectively; the normalized Kendall distance is defined as the percentage of pairs that are out of order, i.e.

$$\tau' = n_D/(n_C + n_D).$$

More specifically, we compute τ' over the union of the exact and the approximated top-100 sets.

For both metrics, we report average results over 100 random test parameters.

5.2 Preprocessing

Reduced Space Construction. For all experiments, we construct the reduced space from PageRank vectors computed at 1000 uniformly sampled parameters.

We report the singular values of the global PageRank data matrix (see Section 3) for both DBLP and the Weibo graph in Figure 1(a). We examine the singular values of the localized PageRank data matrix for 10 randomly selected nodes and report them in Figures 1(b) and 1(c). For the scaled-linear parameterization, we found that for a given number of parameters, the singular values for most of our sample nodes decay at a similar rate, though one or two nodes show singular values with slower decay. We show the singular values for two representative nodes, denoted as v_4 and v_7, in Figure 1(b). For the linear parameterization, the singular values for all the nodes decay at a similar speed; thus we pick one node to report in Figure 1(c). Unsurprisingly, the singular values decay more slowly when there are more parameters on the Weibo graph. With the same number of parameters, the singular values decay more rapidly for the scaled-linear than for the linear parameterization.

Table 2: Preprocessing Time on Different Datasets

Procedure	DBLP	Weibo-G	Weibo-L
Prepare Samples	6 hours	17 hours	3-7 min
Get Basis U	0.8 hours	0.4 hours	1-2 min
Prepare $U^T P^{(i)} U$ [3]	2 min	N/A	<2 min
Choose $\mathcal{I}$ [4]	11 min	12-18min	<1 min

For most experiments, we set the reduced dimension to $k = 100$, as the singular values either decay slowly after this point, or they are already small enough. The only exception is localized PageRank with the scaled-linear parameterization, where we set $k = 50$ for 5 and 10 parameters, as the singular values are already quite small there.

Interpolation Set Selection. We use the pivoted QR method discussed in Section 3 in our experiments. For graphs with skewed incoming degree distributions, we need to restrict the total number of incoming edges for the nodes in $\mathcal{I}$ for performance. For both cost-bounded pivot and threshold pivot methods, we compute the interpolation set with different values for the parameter C or F, and select the $\mathcal{I}$ that works best on a small validation set of personalized PageRank parameters[2]. As the incoming degree distribution for DBLP is not heavy-tailed, we use the unconstrained pivoted QR method. For the Weibo graph, we use cost-bounded pivot with $C = 100$ and threshold pivot with $F = 10$ for localized PageRank, and cost-bounded pivot with $C = 1000$ for global PageRank.

With $|\mathcal{I}| = k$, the reduced system is nearly singular for some parameters. We can avoid this issue by slightly increasing the size of $|\mathcal{I}|$ (i.e. $1.2k$). This suggests that there is no single best interpolation set with size k for all parameters, and the more robust choice for DEIM is to choose $|\mathcal{I}| > k$. Further increasing the size of the interpolation set in DEIM produces more accurate results, with some increase in running time. We set $|\mathcal{I}| = 2k$ for all the experiments; the extra cost over choosing k nodes is modest, and we usually see little improvement after this point.

Preprocessing Time. We show preprocessing times in Table 2. For global PageRank, we compute sample PageRank vectors using the standard power method with stopping criterion $\|x^{(t+1)} - x^{(t)}\|_1 < 10^{-10}$. For localized PageRank, we use BCA with $\epsilon = 10^{-9}$; in this case, the preprocessing time slightly varies with different numbers of parameters and parameterization forms. In general, global PageRank takes hours of CPU time for preparation while localized PageRank takes minutes. In either case, the computation can be easily parallelized across samples.

5.3 Global PageRank

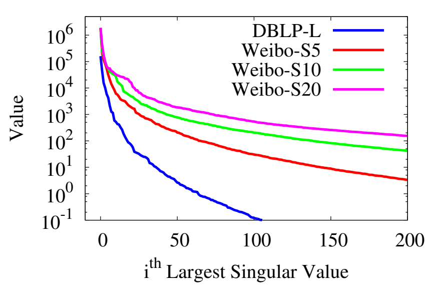
In this section, we discuss the results for global PageRank. The standard power iteration is ill-suited for interactive global PageRank – it takes about 20 seconds on the DBLP graph and a minute on the Weibo graph.

DBLP Graph. We conduct experiments on the DBLP graph to demonstrate the performance of model reduction

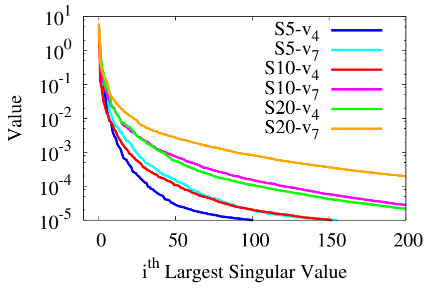
[2] We try $C = 10^0, 10^1, \ldots, 10^5$ and $F = 5, 10, 20, 40, 80$.

[3] Used by Bubnov-Galerkin.

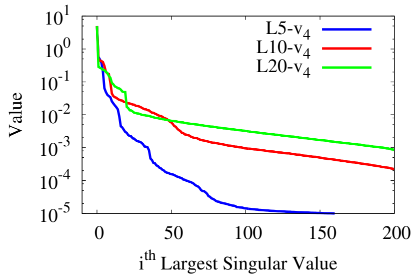
[4] Used by DEIM.



(a) Global PageRank



(b) Localized PageRank, Scaled Linear



(c) Localized PageRank, Linear

Figure 1: Singular Values

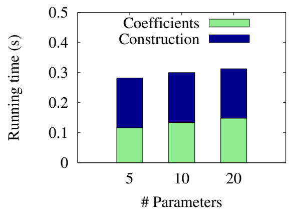
methods for linearly parameterized problems. Figure 2 and Figure 3 show the running time and accuracy of different methods. The running time has two parts: finding the coefficients in the reduced space, and constructing the PageRank vector as a linear combination of reduced basis vectors. Both model reduction approaches are accurate. For DEIM with $|\mathcal{I}| = 200$, the Kendall distance for the top 100 results is around 3×10^{-5}, while the normalized L1 distance is around 0.0005. The Bubnov-Galerkin method has even lower error. In contrast, ScaleRank has both Kendall distance and normalized L1 distance around 0.07.

All the methods produce results with interactive latency, and the model reduction methods run slightly faster than ScaleRank. Most of the time for model reduction is spent on PageRank vector construction. As discussed in Section 3.2, this time can be largely reduced by materializing the PageRank values only for a subset of the nodes. In contrast, the ScaleRank method spends much more time in obtaining the coefficients, and this cost cannot be easily reduced. This is because ScaleRank requires solving several linear programs and is much slower than solving a linear system.

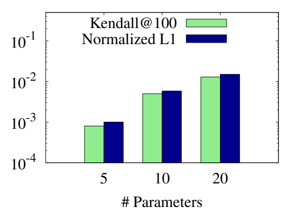
Weibo Graph. We run experiments on the Weibo graph to see how model reduction methods work for social graphs with scaled-linear parameterization. We use DEIM in this case, since the Bubnov-Galerkin method is slow for scaled-linear parameterization. Because the DEIM approximation vector usually does not sum to one for this problem, we add the constraint discussed in Section 3. In Figure 5(a) and Figure 5(b), we report the running time and accuracy for different numbers of parameters. The reduced model can answer queries with interactive latency, and the running time would be much less if we only reconstructed the PageRank values for a small subset of vertices. The error increases with the number of parameters, as increasing the number of parameters increases the error intrinsic in approximating solutions from a low-dimensional space. The Kendall distance is about 0.005 with 10 parameters, but it increases to about 0.013 when there are 20 parameters.

5.4 Localized PageRank

In this section, we discuss localized PageRank on the Weibo graph. Due to space limitations, we only report the Kendall distance for accuracy, but the approximations computed by model reduction always have much lower L1 distance than BCA. With fewer parameters, we see better performance for our methods. Thus we only report the result with 10 and 20 parameters, and not the easier case with 5 parameters.



(a) Running Time



(b) Accuracy

Figure 5: Global PageRank on Weibo Graph

In most applications of localized PageRank, we are only interested in a relatively small number of top-ranked nodes. To find the top 100 nodes, it generally suffices to compute PageRank values for the most promising 10000 nodes as indicated by average PageRank values over the sampled parameters. To construct the part of the PageRank vector associated with these 10000 nodes requires less than a millisecond. The running time for model reduction method reported in this section assumes constructing the PageRank values only for top vertices.

Linear Parameterization. For a linear parameterization, the only work in forming the Bubnov-Galerkin system is adding m matrices of size $k \times k$. In our experiments, with $k = 100$ and $m = 20$, we are able to form and solve the reduced system within a millisecond.

For the problem to retrieve top 100 nodes, the Bubnov-Galerkin approach is nearly two orders of magnitudes faster than BCA, with better accuracy. We report the running time and accuracy for $m = 20$ in Figure 6(a) and 7(a). The Bubnov-Galerkin method has much better accuracy with $m = 5, 10$ than for $m = 20$. For $m = 20$, the error increases as the 100-dimensional trial space is too small to contain highly accurate approximations to the PageRank vector. Still, the Bubnov-Galerkin method produces more accurate results compared with BCA.

Scaled-Linear Parameterization. For the scaled-linear parameterization, the Bubnov-Galerkin approach is expensive, and we turn instead to DEIM. The running time and accuracy for $m = 10$ are reported in Figures 6(b) and 7(b). Overall, DEIM answers the query nearly one order of magnitude faster than BCA with better accuracy.

We show the running time and accuracy for $m = 20$ in Figures 6(c) and 7(c). DEIM is much slower for $m = 20$ than for $m = 10$ for two reasons. First, because the edge weight computation is proportional to the number of parameters, it

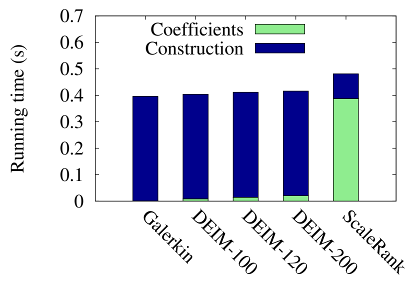


Figure 2: Running Time on DBLP Graph

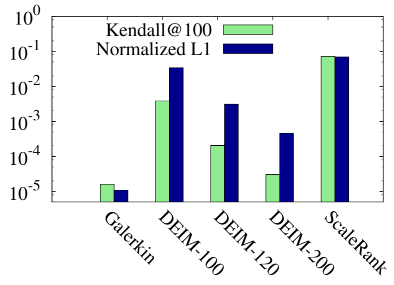


Figure 3: Accuracy on DBLP Graph

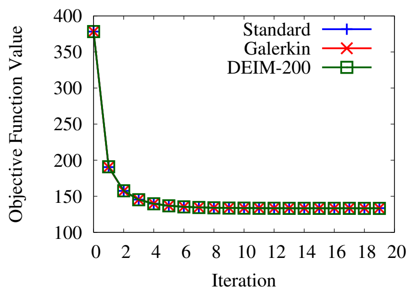


Figure 4: Objective Function Value for Parameter Learning

takes more time to form $M_{\mathcal{I},:}(w)$. Second, we need a larger reduced space ($k = 100$) to achieve acceptable accuracy.

In all cases, while the interpolation sets selected by the cost-bounded pivot method result in shorter running time, it produces less accurate results for a tiny fraction of outlier nodes. The threshold pivot method selects the interpolation sets in a more robust way, but it also requires longer time for each PageRank computation.

5.5 Parameter Learning

In this section, we demonstrate that parameter learning for edge-weighted personalized PageRank can be done at interactive rates via model reduction. For our experiments, we considered the DBLP graph with linear parameterization over the edge types, similar to the setting of PopRank [34]. A partial ranking list with eight papers was used in our experiments to represent the feedback received from the user. Although the original PopRank model is based on a non-differentiable objective function, we use a differentiable objective suggested in later work [3, 7] as it achieves the same goal with much less training time. Following the parameter learning framework described in Section 4, we choose the squared loss with margin $b = 0.2$ and set $\lambda = 1000$ for the regularizer. That is, we minimize the loss function

$$\sum_{(i,j)\in T} \max\left(x_j(w) - x_i(w) + 0.2, 0\right)^2 + 1000\|w - w_0\|_2^2.$$

Here T is the set of pairwise ranking preferences based on the partial ranking list, and w_0 is the default parameter $(0.34, 0.33, 1.0, 0.33, 0.75, 0.25, 1.0)$.

Figure 4 shows the value of the objective function after each iteration with different methods, and all the methods result in almost the same objective function value. As expected, the parameters after each iteration with different methods are also very similar.

The average running time for each iteration is reported in Table 3. For applications involving interactive learning, user feedback usually just slightly adjusts the parameters, and 5-10 optimization iterations should be enough to produce the adjusted parameters to reflect the user's personal preference. In fact, as shown in Figure 4, the objective function value does not change much after eight iterations. Assuming ten iterations are required to produce the new ranking result after taking user feedback, for linear parameterization, we are able to finish the learning procedure in 0.02 seconds by the Bubnov-Galerkin method. Otherwise, DEIM is required to give the adjusted ranking within a second. In comparison, the original method takes minutes for each optimization iteration, as a separate PageRank computation is required for the objective and for each component of the gradient.

Table 3: Avg. Running Time per Opt. Iteration

Method	Standard	Bubnov-Galerkin	DEIM-200
Time(sec)	159.3	0.002	0.033

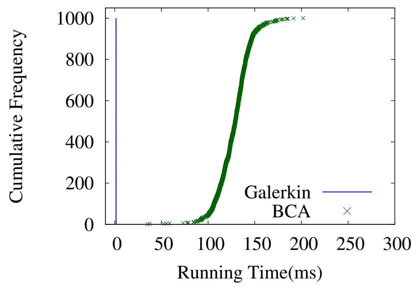
6. RELATED WORK

Personalized PageRank Computation. There has been a lot of work on fast personalized PageRank computation. A recent review can be found in [8]. However, as discussed in Section 2, most previous work focuses on node-weighted personalized PageRank and does not apply to edge-weighted personalized PageRank.

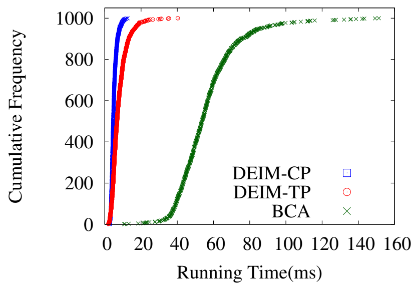
To the best of our knowledge, ScaleRank [28] is the only other work that answers edge-weighted personalized PageRank queries interactively. Like our approach, this method computes sample PageRank vectors $\{x^{(j)}\}_{j=1}^{r}$ in the offline phase, and the approximation to the online query is a linear combination of these samples. However, extracting the linear combination in ScaleRank is slower, as a series of linear programming problems must be solved. It is also unclear how to support learning-to-rank application in this framework. Furthermore, it can only find approximate solutions efficiently on typed graphs, as general graphs would introduce too many constraints in the linear programs.

For localized PageRank, the Monte Carlo method [8] and the Bookmark Coloring Algorithm (BCA) [4] have been proposed to efficiently find the approximate PageRank vector without preprocessing. These methods only explore a localized region close to the target vertex, as the nodes far from the target node usually have negligible PageRank value. The Monte Carlo method simply simulates random walks starting from the target node. BCA maintains a running estimate of the PageRank vector and the residual vector, and repeatedly picks a node and "pushes" its residual to the outgoing nodes until all the residuals are smaller than a predefined threshold.

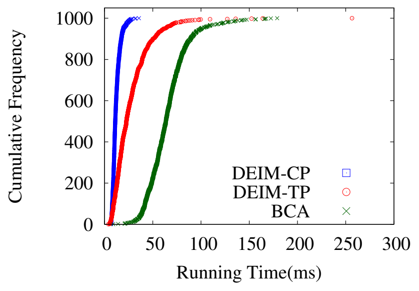
Surrogate Model. Another approach is to approximate $x(w) = M(w)^{-1}b$ by a fast *surrogate model* $\hat{x}(w) \approx x(w)$. One could build surrogates by interpolation or regression, whether via polynomials, radial basis functions, Gaussian processes, or neural networks. In these methods, PageRank is a black box: the methods sample the PageRank vector during model construction, but do not use the structure of the underlying linear system during model evaluation. An example of this approach is [16, 17, 18, 19], in which high-



(a) Linear, 20 Params

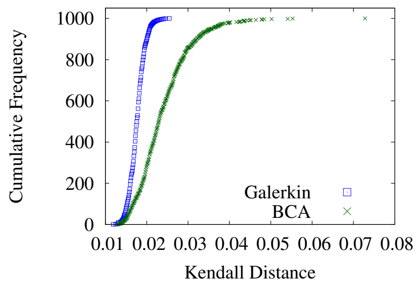


(b) Scaled-Linear, 10 Params

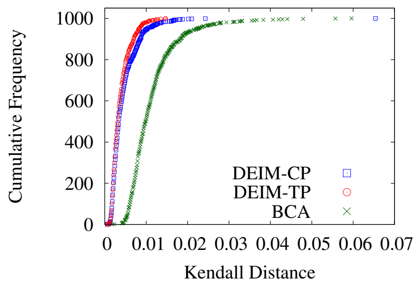


(c) Scaled-Linear, 20 Params

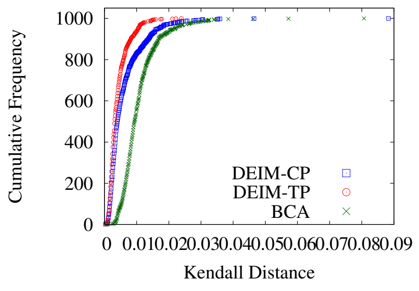
Figure 6: Running Time for Localized PageRank



(a) Linear, 20 Params



(b) Scaled-Linear, 10 Params



(c) Scaled-Linear, 20 Params

Figure 7: Accuracy for Localized PageRank

degree polynomial interpolation is used to approximate the dependence of the PageRank vector on the teleportation parameter α. In contrast, we pursue a *model reduction* strategy that uses the PageRank equations.

Model Reduction in Simulation. In physical simulations, one uses model reduction for tasks that involve repeated model evaluation, such as design, optimization, control, and uncertainty quantification. Our work is inspired by the work of Patera and co-workers on the use of reduced basis methods for elliptic PDEs with uncertain (or stochastic) coefficients [24], and by work on model reduction of dynamical systems in the time or frequency domain [5, 39, 15, 6].

7. CONCLUSIONS

In this paper, we present the first general scalable method for edge-weighted personalized PageRank based on model reduction. We discuss optimizations for common parameterizations and cost/accuracy tradeoffs when applying model reduction to power-law graphs. For applications such as learning to rank, our model reduction methods are nearly *five orders of magnitudes* faster than the standard approach.

In future work, we plan to investigate whether the PageRank computations can be pushed to the client side by sending the reduced model. We would also like to investigate how to update the reduced model efficiently as the graph evolves over time.

Acknowledgements. This work was funded by NSF Grants IIS-0911036 and IIS-1012593, and by the iAd Project from the National Research Council of Norway. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

8. REFERENCES

[1] GNU Linear Programming Kit. `http://www.gnu.org/software/glpk/`.
[2] SciPy. `http://www.scipy.org/index.html`.
[3] A. Agarwal, S. Chakrabarti, and S. Aggarwal. Learning to rank networked entities. In *KDD*, 2006.
[4] R. Andersen, F. Chung, and K. Lang. Local graph partitioning using PageRank vectors. In *FOCS*, 2006.
[5] A. C. Antoulas. *Approximation of Large-Scale Dynamical Systems*. SIAM, 2005.
[6] A. C. Antoulas, C. A. Beattie, and S. Gugercin. Interpolatory model reduction of large-scale dynamical systems. In *Efficient modeling and control of large-scale systems*, pages 3–58. Springer, 2010.
[7] L. Backstrom and J. Leskovec. Supervised random walks: predicting and recommending links in social networks. In *WSDM*, 2011.
[8] B. Bahmani, A. Chowdhury, and A. Goel. Fast incremental and personalized pagerank. *PVLDB*, 4(3):173–184, 2010.
[9] A. Balmin, V. Hristidis, and Y. Papakonstantinou. Objectrank: Authority-based keyword search in databases. In *VLDB*, 2004.
[10] P. Berkhin. A survey on PageRank computing. *Internet Mathematics*, 2(1):73–120, 2005.
[11] P. Berkhin. Bookmark-Coloring Algorithm for Personalized PageRank Computing. *Internet Mathematics*, 3:41–62, 2006.
[12] B. Bi, Y. Tian, Y. Sismanis, A. Balmin, and J. Cho. Scalable topic-specific influence analysis on microblogs. In *WSDM*, 2014.
[13] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1):107–117, 1998.
[14] H.-J. Bungartz and M. Griebel. Sparse grids. *Acta numerica*, 13:147–269, 2004.

[15] S. Chaturantabut and D. C. Sorensen. Nonlinear model reduction via discrete empirical interpolation. *SIAM J. Sci. Comput.*, 32(5):2737–2764, 2010.

[16] P. G. Constantine and D. F. Gleich. Using polynomial chaos to compute the influence of multiple random surfers in the pagerank model. In *Algorithms and Models for the Web-Graph*, pages 82–95. Springer, 2007.

[17] P. G. Constantine and D. F. Gleich. Random alpha PageRank. *Internet Mathematics*, 6(2):189–236, 2009.

[18] P. G. Constantine, D. F. Gleich, and G. Iaccarino. Spectral methods for parameterized matrix equations. *SIAM J. Matrix Anal. Appl.*, 31(5):2681–2699, 2010.

[19] P. G. Constantine, D. F. Gleich, and G. Iaccarino. A factorization of the spectral Galerkin system for parameterized matrix equations: Derivation and applications. *SIAM J. Sci. Comput.*, 33(5):2995–3009, 2011.

[20] W. Feng and J. Wang. Incorporating heterogeneous information for personalized tag recommendation in social tagging systems. In *KDD*, 2012.

[21] B. Gao, T.-Y. Liu, W. Wei, T. Wang, and H. Li. Semi-supervised ranking on very large graphs with rich metadata. In *KDD*, 2011.

[22] D. F. Gleich. PageRank beyond the web. *arXiv*, cs.SI:1407.5107, 2014.

[23] G. Golub and C. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, fourth edition, 2012.

[24] M. A. Grepl, Y. Maday, N. C. Nguyen, and A. T. Patera. Efficient reduced-basis treatment of nonaffine and nonlinear partial differential equations. *ESAIM: Mathematical Modelling and Numerical Analysis*, 41(03):575–605, 2007.

[25] W. Hager. Condition estimates. *SIAM J. Sci. Stat. Comput.*, 5(2):311–316, 1984.

[26] T. H. Haveliwala. Topic-sensitive PageRank. In *WWW*, 2002.

[27] N. J. Higham and F. Tisseur. A block algorithm for matrix 1-norm estimation with an application to 1-norm pseudospectra. *SIAM J. Matrix Anal. Appl.*, 21:1185–1201, 2000.

[28] V. Hristidis, Y. Wu, and L. Raschid. Efficient ranking on entity graphs with personalized relationships. *IEEE Trans. Knowl. Data Eng.*, 26(4):850–863, 2014.

[29] G. Jeh and J. Widom. Scaling personalized web search. In *WWW*, 2003.

[30] B. Jiang. Ranking spaces for predicting human movement in an urban environment. *International Journal of Geographical Information Science*, 23(7):823–837, 2009.

[31] R. Kumar and S. Vassilvitskii. Generalized distances between rankings. In *WWW*, 2010.

[32] A. N. Langville and C. D. Meyer. Deeper inside PageRank. *Internet Mathematics*, 1(3):335–380, 2004.

[33] W. J. Morokoff and R. E. Caflisch. Quasi-random sequences and their discrepancies. *SIAM J. Sci. Comput.*, 15(6):1251–1279, 1994.

[34] Z. Nie, Y. Zhang, J.-R. Wen, and W.-Y. Ma. Object-level ranking: bringing order to web objects. In *WWW*, 2005.

[35] A. Paul-Duboise-Taine and D. Amsallem. An adaptive and efficient greedy procedure for the optimal training of parametric reduced-order models. *Int. J. Numer. Meth. Engng.*, 2014.

[36] R. Pinnau. Model reduction via proper orthogonal decomposition. In *Model Order Reduction: Theory, Research Aspects and Applications*, pages 95–109. Springer, 2008.

[37] M. Rathinam and L. R. Petzold. A new look at proper orthogonal decomposition. *SIAM J. Num. Anal.*, 41(5):1893–1925, 2003.

[38] C. Sanderson. Armadillo: An open source C++ linear algebra library for fast prototyping and computationally intensive experiments. Technical report, NICTA, Australia, October 2010.

[39] W. Schilders. *Model Order Reduction: Theory, Research Aspects and Applications*, volume 13 of *Mathematics in Industry*. Springer, Berlin, 2008.

[40] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su. ArnetMiner: extraction and mining of academic social networks. In *KDD*, 2008.

[41] R. Varadarajan, V. Hristidis, and L. Raschid. Explaining and reformulating authority flow queries. In *ICDE*, 2008.

[42] R. Varadarajan, V. Hristidis, L. Raschid, M. Vidal, L. D. Ibáñez, and H. Rodríguez-Drumond. Flexible and efficient querying and ranking on hyperlinked data sources. In *EDBT*, 2009.

[43] J. Weng, E.-P. Lim, J. Jiang, and Q. He. TwitterRank: finding topic-sensitive influential twitterers. In *WSDM*, 2010.

[44] W. Xing and A. Ghorbani. Weighted PageRank algorithm. In *CNSR*, 2004.

APPENDIX

A. QUASI-OPTIMALITY

Petrov-Galerkin methods are *quasi-optimal*: the error in the Petrov-Galerkin approximation is within some factor of the best error possible in the space. We summarize with the following theorem.

THEOREM 1. *Suppose* $\|e_*\| = \min_{y_*} \|Uy_* - x\|$ *is the error in the best approximation from* $\mathcal{R}(U)$ *in some norm. The error* $\|e\| = \|Uy - x\|$ *for the Petrov-Galerkin approximation is bounded by* $\|e\| \le (1+\kappa)\|e_*\|$ *for* $\kappa = \|U\|\|(W^T MU)^{-1}\|\|W^T M\|$. *For the DEIM method, the error is bounded by* $\|e\| \le (1 + \kappa_{\text{DEIM}})\|e_*\|$ *for* $\kappa_{\text{DEIM}} = \|U\|\|(M_{\mathcal{I},:}U)^{\dagger}\|\|M_{\mathcal{I},:}\|$ *where* $(M_{\mathcal{I},:}U)^{\dagger}$ *indicates the Moore-Penrose pseudoinverse, i.e. the least-squares solution operator.*

PROOF. Suppose Uy_* is the best approximation in the space to $x = M^{-1}b$ under some norm. Let $e = x - Uy$ and $e_* = x - Uy_*$ be the error in the chosen approximation and the best approximation, respectively. Substituting $b = M(Uy_* + e_*)$ into the Petrov-Galerkin ansatz yields

$$W^T MU(y - y_*) = W^T Me_*,$$

and therefore

$$e = e_* - U(y - y_*) = \Big[I - U(W^T MU)^{-1} W^T M\Big] e_*.$$

Taking norms, we have

$$\|e\| \le \Big(1 + \|U\|\|(W^T MU)^{-1}\|\|W^T M\|\Big)\|e_*\|.$$

In the DEIM case, a similar argument yields

$$e = \Big[I - U(M_{\mathcal{I},:}U)^{\dagger} M_{\mathcal{I},:})\Big] e_*$$

and taking norms as before yields the final result. □

The most significant term in this bound is the norm of the inverse of the projected system, i.e. $\|(W^T MU)^{-1}\|$ or $\|(M_{\mathcal{I},:}U)^{-1}\|$. In particular, if the columns of U are normalized to have absolute sums equal to zero, then $\|U\|_1 = 1$ and $\|M_{\mathcal{I},:}\|_1 \le 1 + \alpha$, so that for the one-norm the quasi-optimality constant is bounded by

$$\kappa_{\text{DEIM}} \le (1 + \alpha)\|(M_{\mathcal{I},:}U)^{\dagger}\|_1.$$

That is, the key quantity in this case is $\|(M_{\mathcal{I},:}U)^{\dagger}\|_1$, which measures how close the projected system is to being singular. When evaluating the reduced model, we can bound this quantity with little extra cost via Hager's algorithm or variants [25, 27].