Learning to Rank Networked Entities

Alekh Agarwal
IIT Bombay
alekh@cse.iitb.ac.in

Soumen Chakrabarti *
IIT Bombay
soumen@cse.iitb.ac.in

Sunny Aggarwal
IIT Bombay
sunny@it.iitb.ac.in

ABSTRACT

Several algorithms have been proposed to learn to rank entities modeled as feature vectors, based on relevance feedback. However, these algorithms do not model network connections or relations between entities. Meanwhile, Pagerank and variants find the stationary distribution of a reasonable but arbitrary Markov walk over a network, but do not learn from relevance feedback. We present a framework for ranking networked entities based on Markov walks with parameterized conductance values associated with the network edges. We propose two flavors of conductance learning problems in our framework. In the first setting, relevance feedback comparing node-pairs hints that the user has one or more hidden preferred communities with large edge conductance, and the algorithm must discover these communities. We present a constrained maximum entropy network flow formulation whose dual can be solved efficiently using a cutting-plane approach and a quasi-Newton optimizer. In the second setting, edges have types, and relevance feedback hints that each edge type has a potentially different conductance, but this is fixed across the whole network. Our algorithm learns the conductances using an approximate Newton method.

Categories and Subject Descriptors

H.3.3 [**INFORMATION STORAGE AND RETRIEVAL**]: Information Search and Retrieval[Retrieval models; Relevance feedback]; I.5.1 [**PATTERN RECOGNITION**]: Models[Statistical]

General Terms

Algorithms, Experimentation, Measurement

Keywords

Pagerank, conductance, network flow, maximum entropy

1. INTRODUCTION

Consider a set V of entities (such as documents) that can be returned by a search engine in response to queries. Each entity $v \in V$ may be represented by a feature vector $x_v \in \mathbb{R}^d$. E.g., if the entities are documents, they can be represented in the vector space model used in Information Retrieval (IR) [21]. In standard IR, given a query vector $q \in \mathbb{R}^d$, responses are presented in decreasing order of the dot product $q'x_v$.

In fact, any vector $w \in \mathbb{R}^d$ defines a scoring function $w'x$ and thus (in general) a total order over V. A series of papers [14, 16, 1] explore how to learn w, given a partial order "$\prec$" involving some of the entities. If $u \prec v$, we want $w'x_u \leq w'x_v$. (Throughout, we will use $\prec$ both as an operator, as in "$a \prec b$," and as a set of preferences, as in "$(a, b) \in \prec$." Also, $a \prec b$ means $b \succ a$.) We will review some of this work in Section 2.1. None of this family of algorithms models entities as nodes in a graph.

Increasingly, documents are not isolated sequences of words, but are interconnected through a network. This is true not only of the Web, where hyperlinks greatly assist ranking [6, 17], but also of entity-relationship (ER) graphs [5, 3], XML data [12], and Semantic Web networks [2] where nodes represent entities with textual attributes and edges represent diverse relations.

In these networked data models, ranking is often achieved by some generalization of Pagerank [6] or HITS [17]. A Markovian random walk is defined on the graph, and the score of a node is defined as its steady-state visit probability. The random walk, while usually intuitive and reasonable, is arbitrarily designed, in the sense that no preference $\prec$ is automatically incorporated to improve its design. We produce several examples below, and will review the techniques in Sections 2.3 and 2.4.

In standard Pagerank [6], all edges are considered the same. In ObjectRank [3], the Intelligent Surfer [20], and XRank [12], the random walk favors nodes containing query keywords in a fixed, arbitrary manner. In topic-sensitive Pagerank [13], the random walk preferentially moves to nodes (Web pages) on a specified topic. In personalized Pagerank [15] the random walk preferentially moves to pages visited by the user in the past. Only very few projects [8, 24, 10, 19] attempt to *learn* the parameters of the random walk.

1.1 Our contributions

Our primary contribution is to bring together the power of Markovian walk-based scoring functions and the flexibility of improving the scoring function using relevance judgments. (We focus on the Pagerank family, but it should be possible to extend our work to some members of the HITS family as well.)

We propose a framework for learning certain edge parameters of Markovian walks on graphs, given preference orders over nodes. Within this framework, we consider two learning problems that have to estimate conditional and absolute transition probabilities—$\Pr(v|u)$ and $\Pr(u \to v)$—on each edge (u, v) of the graph (see Section 1.2 for details).

*Contact author.

KDD'06, August 20–23, 2006, Philadelphia, Pennsylvania, USA.

The difference between the two settings is that in one, we must estimate the transition probability on each edge separately (Section 3) but in the other, edges are associated with a few types (person *wrote* book, company *located-in* city etc.) and each type of edge has a globally fixed (but unknown) transition probability, which our algorithm must discover (Section 4).

We evaluate the proposed algorithms on synthetic data generated by state-of-the-art graph generators, using broad statistics measured from real graphs collected from DBLP (`http://dblp.uni-trier.de/`) and CiteSeer (`http://citeseer.ist.psu.edu/`). We show that the algorithms are scalable and that they compute Markovian walk parameters that lead to good prediction of unseen user preferences.

1.2 Classes of formulations

Throughout this paper, our "null hypothesis" or "parsimonious belief" is that standard Pagerank is the ideal ranking mechanism unless $\prec$ provides contrary evidence. Based on $\prec$, our learners must pick up an "ideal" Markovian walk from a larger hypothesis space. Here we consider two spaces, the first containing the second.

1.2.1 Hidden favored communities

In this setting, $\prec$ is non-empty because the user has one or few favorite communities. Not only is the "ideal" random walk disproportionately likely to transit to nodes in these communities, but the edges within these communities may have large transition probability $\Pr(v|u)$ compared to the rest of the graph. E.g., to a computer vision researcher, papers and citations related to computer vision in DBLP are more significant than other papers and citations, which are mere distractions.

With rare exceptions [24], existing personalization literature has proposed arbitrary biases in the Markov procedure "by force" [13, 15] and not discovered a modified Markov walk parameters from preference data. In contrast, we propose an efficient and scalable procedure to estimate transitions modeled as a *network flow* p with $p_{uv} = \Pr(u \to v)$ on each edge (u, v) of the graph, such that the total inflows into the nodes satisfy $\prec$ as far as possible.

1.2.2 Type-specific edge weights

In the second setting the graph represents entity-relationship (ER) connectivity with multiple kinds of relationship edges. Graph-structured databases are becoming a "lowest common denominator" representation not only for XML [12, 2], but also for relational data [5, 3].

Each edge (u, v) in an ER graph adheres to a schema, i.e., has an associated type $t(u, v) \in \{1, \ldots, T\}$, a fixed and typically small set of edge types. E.g., the edge connecting a paper to an author has a type different from a paper-to-paper citation edge, and the "ideal" random walk is likely to transit along edges of different types with different probabilities. Our assumption is that $\prec$ is generated because of these differences between the ideal and baseline random walks.

Our algorithm sees the graph G and preferences $\prec$, and knows the baseline walk, and has to discover the ideal walk by estimating the relative conductance of each type of edge. In contrast, many systems [12, 3] that use Pagerank-like Markovian walks over typed graphs associate each edge type t with an arbitrarily fixed weight $\beta(t)$ (or two weights if the edge is bidirectional) which then determines its conductance.

2. BACKGROUND AND RELATED WORK

We review two kinds of prior work: those that we build upon, and those that we enhance or generalize.

2.1 Scoring feature vectors

Most algorithms that learn to order items model them as feature vectors $x \in \mathbb{R}^d$ [9, 14, 16]. The quest is for a model vector $w^* \in \mathbb{R}^d$ so that the score of item x is $w'x \in \mathbb{R}$, and items are ranked in decreasing order of this score. A preference $i \prec j$ means we want w to be such that $w'x_i \leq w'x_j$. A max-margin search for w introduces a set of slack variables $s^*_{ij} \geq 0$ and solves the quadratic optimization

$$\min_{s \geq \mathbf{0}; w \in \mathbb{R}^d} w'w + B \sum_{(i,j): i \prec j} s_{ij} \quad \text{subject to}$$
$$w'x_j - w'x_i + s_{ij} \geq 1 \quad \forall i \prec j \qquad \text{(RankSVM)}$$

Note that if $w'x_j \geq 1 + w'x_i$ then $i \prec j$ is satisfied, $s_{ij} = 0$ and no penalty is paid. As with support vector classifiers, B is a tuned parameter that trades off the model complexity $w'w = \|w\|_2$ against the penalty for violating preferences. Note that no graphical connection is modeled between any x_i and x_j; they remain independent feature vectors.

2.2 Pagerank

Pagerank [6] is a total order on nodes in a graph $G = (V, E)$ imposed via a "random surfer" model. The random surfer performs a Markovian walk on G, and is at node j with probability $p_j = \sum_i p_i p(j|i)$. If we write $p(j|i)$ as a $|V| \times |V|$ transition matrix C, the column vector p solves $p = Cp$, where C is designed as

$$C(j,i) = \begin{cases} \alpha \frac{[\![(i,j) \in E]\!]}{\text{OutDegree}(i)} + (1-\alpha) r_j, & i \in V_o \\ r_j, & \text{otherwise} \end{cases} \qquad \text{(UnweightedPagerank)}$$

Here $[\![I]\!] = 1$ if boolean condition I is true, and 0 otherwise. $V_o \subseteq V$ is the set of nodes which are not dead-ends, i.e., have at least one out-link. The two design variables are α, the probability of walking to a neighbor instead of jumping to a random node; and $r = (r_j)$, the *teleport* or *personalization* vector, which, in ordinary Pagerank, is set uniformly to $(1/n, \ldots, 1/n)$ where $n = |V|$. With r set thus, p depends only on the structure of G and the value of α.

2.3 Teleport optimization

Follow-up work on Pagerank has attempted to modify the teleport vector r to "personalize" the scores heuristically, based on topics [13], words [20, 3], or user preferences on graph nodes [15].

We will compare our work with that of Tsoi *et al.* [24]. They propose a quadratic programming (QP) approach to

optimizing r given preferences $\prec$. For simplicity, assume $V_o = V$, i.e., that there are no dead-end nodes in G. (We can add new edges to connect any dead-end node u to itself or all other nodes.) Let A be the node adjacency matrix of G with each row scaled to add up to 1. Given teleport vector $r \in \mathbb{R}^{|V|\times 1}$, the Pagerank vector satisfies

$$\begin{aligned} p &= \alpha A' p + (1-\alpha) r, \quad \text{and therefore} \\ p &= (1-\alpha)(\mathbb{I} - \alpha A')^{-1} r = Mr, \quad \text{say.} \end{aligned} \tag{1}$$

Here $\mathbb{I}$ is the identity matrix. The inverse in (1) always exists, but we will not be concerned with the complications of computing it. We are looking for a r so that elements of the resulting p satisfies inequalities given by $\prec$. These preferences are easily encoded in a matrix $\Pi \in \{-1, 0, 1\}^{|\prec|\times|V|}$ and written as $\Pi p \geq \mathbf{0}^{|\prec|\times 1}$. Each row of Π represents one preference $u \prec v$ and has one -1 (in the u column) and one 1 (in the v column) and the other columns are zeros. If for r we used the uniform teleport r^U, we would get the standard Pagerank vector $p^U = Mr^U$. Tsoi *et al.* propose to minimize $\|p - p^U\|_2$ while making p satisfy the constraints given by Π. This leads to the "hard constraint" QP:

$$\begin{aligned} &\min_{r\in\mathbb{R}^{|V|}} (Mr - Mr^U)'(Mr - Mr^U) \\ &\text{s.t.} \quad \Pi Mr \geq \mathbf{0}, \quad r \geq \mathbf{0}, \quad \mathbb{1}'r = 1. \end{aligned} \tag{2}$$

Here $\mathbb{1}$ is a vector of 1s of suitable size. (We also need $Mr \geq \mathbf{0}$ but that is guaranteed by $r \geq \mathbf{0}$.) Surprisingly, Tsoi *et al.* do not enforce $\mathbb{1}'r = 1$, i.e., $\|r\|_1 = 1$, which is essential to keep r meaningful as a teleport probability vector, and which is generally violated unless enforced. Tsoi *et al.* note that (even without the $\mathbb{1}'r = 1$ constraint) (2) may not be feasible, and propose a "soft constraint" QP in which they replace the one-sided constraint $\Pi Mr \geq \mathbf{0}$ with an additional symmetric quadratic penalty in the objective:

$$\begin{aligned} \min_{r\in\mathbb{R}^{|V|}} &(Mr - Mr^U)'(Mr - Mr^U) + \underline{B\, r'(M'\Pi'\Pi M)r} \\ &\text{s.t.} \quad r \geq \mathbf{0}, \quad \mathbb{1}'r = 1. \end{aligned} \tag{3}$$

Here, too, enforcing $\Pi Mr \geq \mathbf{0}$ leads to infeasibility and not enforcing it generally leads to violation. Also, it is unclear why $\Pi Mr > \mathbf{0}$ is being penalized. One simple fix is to introduce slack variables and rewrite the optimization as

$$\begin{aligned} \min_{r\in\mathbb{R}^{|V|},\, s\geq\mathbb{0}} &(Mr - Mr^U)'(Mr - Mr^U) + \underline{B\mathbb{1}'s} \\ &\text{s.t.} \quad r \geq \mathbf{0}, \quad \mathbb{1}'r = 1, \quad \Pi Mr + s \geq \mathbb{0}, \end{aligned} \tag{4}$$

but the resulting QP optimizer turns out to be much slower than Tsoi *et al.*'s formulation. As we shall see in Section 3.4.3, these are serious limitations from which our proposals do not suffer.

2.4 Tuning edge weights

Equation (UnweightedPagerank) can be generalized to incorporate edge weights. Each edge e has an associated *edge type* $t(e)$ taken from a flat set of edge types T. Any edge e with type $t(e)$ has a strictly positive weight $\beta(t(e)) > 0$. A nonexistent edge has weight zero. The modified Pagerank equation is

$$C(j,i) = \begin{cases} \alpha \frac{\beta(t(i,j))}{\text{OutWeight}(i)} + (1-\alpha) r_j, & i \in V_o \\ r_j, & \text{otherwise} \end{cases} \tag{WeightedPagerank}$$

where $\text{OutWeight}(i) = \sum_j \beta(t(i,j))$. C is a function of the weights β, and we are looking for β such that the p that solves $p = Cp$ also satisfies $\prec$. Unlike (1) where M is a constant, we will now face quadratic equality constraints, which poses more difficulty than quadratic objectives with linear constraints.

There have been various attempts to approximate this optimization via gradient descent [8], error backpropagation [10] or simulated annealing [19]. Unfortunately the objective is not well-behaved, and the search procedures are complex and time-consuming. Usually, the search routine has to effectively call Pagerank a large number of times with various weight choices. We propose a simple and efficient way to search for $\beta(t)$s approximately in Section 4.

3. LEARNING CONSTRAINED FLOWS

We now give a different formulation that not only captures teleport learning, but generalizes to learning a network flow throughout G, from which node ranks can then be derived naturally. In Pagerank, since $p_j = \sum_i p_{ij} = \sum_i p_i p(j|i)$, we can cast our transition process in terms of *flows* p_{ij} along each edge (i,j), with $\sum_{i,j} p_{ij} = 1$.

A Markov process must also satisfy the flow balance property: $\sum_i p_{iu} = \sum_j p_{uj}$ for each node u. Any Pagerank, biased or unbiased, with uniform or non-uniform teleport, satisfies the above two properties. But there are other classes of solutions as well. In particular, Tomlins [22] advocates maximizing the entropy $H(p)$ of $\{p_{ij}\}$, i.e., $-\sum_{i,j} p_{ij} \log p_{ij}$ while enforcing the above constraints. In this Section we will combine Tomlin's view of Pagerank as a flow system together with Joachims and others' notions of max-margin scoring/ranking.

3.1 Primal formulation

Before we get to our formulations, we provide a uniform device to handle teleport. We add a special dummy node d, and directed edges (v,d) and (d,v) for all $v \in V$. The augmented graph is called $G' = (V', E')$.

If in the original graph G, u had no outlinks, the entire inflow into u has to pass out through (u,d). If u had at least one outlink in G, a fraction $1-\alpha$ of the net inflow into u passes out through (u,d) and the remaining fraction α is apportioned into the original outlinks (u,v) in G.

The outflows from d back to other nodes along (d,v) edges are variables included in our optimization; i.e., the search for a good teleport vector is embedded in our formulation.

The "hard constraint" optimization can be cast as follows:

$$\min_{\{0\le p_{uv}\le 1\}} \sum_{(u,v)\in E'} p_{uv}\log p_{uv} \quad \text{(HardObjective)}$$

$$\text{such that} \sum_{(u,v)\in E'} p_{uv} - 1 = 0 \quad \text{(Total)}$$

$$\forall v \in V' : -\sum_{(u,v)\in E'} p_{uv} + \sum_{(v,w)\in E'} p_{vw} = 0 \quad \text{(Balance)}$$

$$\forall v \in V_o : -\alpha p_{vd} + (1-\alpha)\sum_{(v,w)\in E} p_{vw} = 0 \quad \text{(Teleport)}$$

$$\forall u \prec v : \sum_{(w,u)\in E'} p_{wu} - \sum_{(w,v)\in E'} p_{wv} \le 0 \,\text{(Preference)}$$

Why no margin in (Preference)***?*** Some traditional classifiers use the notion of a margin to make the system more robust to minor perturbations of training points on either side of the decision boundary, analogous to the margin of "1" in (RankSVM). An arbitrary margin can be asserted because any margin can be satisfied by suitably scaling the model (β in case of (RankSVM)). However, in our case, there is no such scaling capability: all $p_{uv} \in [0,1]$ and indeed $\sum_{u,v} p_{uv} = 1$. Therefore, a margin would represent an arbitrary decision and will simply add more parameters to the system. Also, given that we are dealing with extremely small numbers (a typical flow could be $O(1/|E|)$, say), too large a choice of the margin may easily lead to infeasibility.

Soft constraints and slack variables: As in SVMs, the "soft margin" counterpart introduces and penalizes slacks s_{uv} with a penalty function $\mathcal{L}(s)$ weighted with a magic penalty parameter B. Some common choices for $\mathcal{L}(s)$ are the L1 penalty $\sum_{u\prec v} s_{uv}$ and the L2 penalty $\sum_{u\prec v} s_{uv}^2$. Because $0 \le s_{uv} \le 1$, L2 downplays violations and so L1 is usually more suitable; therefore we focus on L1. (Preference) changes to

$$\forall u \prec v : \sum_{(w,u)\in E'} p_{wu} \le s_{uv} + \sum_{(w,v)\in E'} p_{wv} \quad \text{(SoftPreference)}$$

Minimizing distance to a parsimonious model: Maximizing the entropy of flow $\{p_{uv}\}$ seeks to make all edge flows equal. A more meaningful "null hypothesis" or "parsimonious belief" is that all edges are functionally identical and the teleport follows a uniform distribution—this is just (UnweightedPagerank) and gives what we call a "reference" flow $\{q_{uv}\}$. Flow q may already satisfy some preferences. Our objective is to perturb q minimally to get a flow p that (largely) satisfies $\prec$, and the KL divergence $\mathrm{KL}(p||q)$ is a natural measure of perturbation. Based on the above discussion our final primal objective becomes

$$\min_{\substack{\{0\le p_{uv}\le 1\}\\ \{0\le s_{uv}:u\prec v\}}} \sum_{(u,v)\in E'} p_{uv}\log\frac{p_{uv}}{q_{uv}} + B\sum_{u\prec v} s_{uv} \quad \text{(SoftObjective)}$$

Why not include α in the optimization? We avoid including α in the optimization for two main reasons. First, this would result in quadratic constraints, making the optimization much more difficult. Also, if α were an optimization variable, the hypothesis space would include a degenerate solution: with α set to zero, and p_{dv}'s set to satisfy a total order extending $\prec$, the empirical risk reduces to zero. Even in the soft-constraint version, too large a B may drive us toward this solution, overriding the $\mathrm{KL}(p||q)$ term. Hence we felt that it is better in practice to do a grid search over a small range of "sensible" values of α rather than include α in the optimization.

3.2 Dual formulation

We propose to solve the *dual* of the above optimization, because the dual has some useful and interesting properties. Instead of $O(|E|)$ variables as in the primal problem, it has $O(|V|+|\prec|)$ dual variables. Each dual variable turns out to be either unconstrained, or bounded below and above by two constants (a so-called "box-constrained" variable). Each iteration of the dual optimizer is analogous in computational cost to an iteration of Pagerank. And, as we shall see in Section 3.3, we can induct only a carefully chosen subset of dual variables into the optimization, implicitly setting the rest to zeros, and considerably speed up the optimization.

Let $\{\beta_v : v \in V'\}$ ($|V|+1$ variables), $\{\tau_v : v \in V_o\}$ ($|V_o|$ variables) and $\{\pi_{uv} : u \prec v\}$ ($|\prec|$ variables) be the dual variables corresponding to constraints (Balance), (Teleport) and (SoftPreference) respectively. Let

$$\mathrm{bias}(v) = \sum_{r\prec v} \pi_{rv} - \sum_{v\prec s} \pi_{vs} \quad (5)$$

Using a standard Lagrangian procedure, we arrive at the following observations.

Proposition 1. *The primal flows can be expressed as*

$$\begin{aligned}
\forall v \in V \quad & p_{dv} = (1/Z)\, q_{dv}\, \exp(\beta_v - \beta_d + \mathrm{bias}(v)) \\
\forall v \in V_o \quad & p_{vd} = (1/Z)\, q_{vd}\, \exp(\beta_d - \beta_v + \alpha\tau_v) \\
\forall v \in V \setminus V_o \quad & p_{vd} = (1/Z)\, q_{vd}\, \exp(\beta_d - \beta_v) \\
\forall (u,v) \in E \quad & p_{uv} = (1/Z)\, q_{uv} \\
& \qquad \exp(\beta_v - \beta_u - (1-\alpha)\tau_u + \mathrm{bias}(v))
\end{aligned}$$

Here all β and τ are unconstrained, and each $\pi_{uv} \in [0,B]$. The dual objective to maximize is $-\log Z$, where

$$\begin{aligned}
Z = & \sum_{v\in V} q_{dv}\exp(\beta_v - \beta_d + \mathrm{bias}(v)) \\
& + \sum_{v\in V_o} q_{vd}\exp(\beta_d - \beta_v + \alpha\tau_v) + \sum_{v\in V\setminus V_o} q_{vd}\exp(\beta_d - \beta_v) \\
& \quad + \sum_{(u,v)\in E} q_{uv}\exp(\beta_v - \beta_u - (1-\alpha)\tau_u + \mathrm{bias}(v)),
\end{aligned}$$

so that $\sum_{(u,v)\in E'} p_{uv} = 1$.

Once we have routines to compute $\partial Z/\partial\beta_x$, $\partial Z/\partial\tau_x$ and $\partial Z/\partial\pi_{xy}$ (we omit the tedious expressions) the dual can be solved using the BLMVM optimizer [4].

3.3 Variable inclusion

Computing the dual objective and gradient takes time roughly proportional to $|V|+|E|$ and $|\prec|$, as we shall see in

Section 3.4.3. However, in a deployed search system, V, E and $\prec$ can be large, and $\prec$ can grow indefinitely with time.

Two features of our setting come to our rescue. First, while satisfying balance equalities exactly is mathematically appealing, it matters less in practice. Small imbalances near low-ranked nodes may not matter at all to the best-ranked nodes. Second, some pairs in $\prec$ may (approximately) subsume others.

Following the cutting-plane approach of Tsochantaridis *et al.* [23], we propose an approach to introduce dual variables gradually to the dual optimizer. We present some theoretical guarantees of progress and termination, and also provide experimental evidence that our approach can be effective.

1: Input: V', E', $\prec$ and tolerance ϵ
2: Let $\mathcal{B}, \mathcal{T}, \mathcal{P}$ be current sets of dual variables
3: $\mathcal{B} \leftarrow \varnothing$, $\mathcal{T} \leftarrow \varnothing$, $\mathcal{P} \leftarrow \varnothing$ (implicitly all $\beta, \pi, \tau = 0$)
4: **repeat**
5: {estimate violations}
6: $\mathcal{V}(\beta_v) = |\,\mathrm{OutFlow}(v) - \mathrm{InFlow}(v)|$
7: $\mathcal{V}(\tau_v) = |\alpha p_{vs} - (1-\alpha)\sum_{(v,w)\in E} p_{vw}|$
8: $\mathcal{V}(\pi_{u,v}) = \mathrm{InFlow}(v) - \mathrm{InFlow}(u)$
9: discard candidates with violation $\mathcal{V}$ less than ϵ
10: $\mathcal{B} \leftarrow \mathcal{B} \cup \arg\max^{(k)}_{v\in V'} \mathcal{V}(\beta_v)$
11: $\mathcal{T} \leftarrow \mathcal{T} \cup \arg\max^{(k)}_{v\in V'} \mathcal{V}(\tau_v)$
12: $\mathcal{P} \leftarrow \mathcal{P} \cup \arg\max^{(k)}_{(u,v)\in \prec} \mathcal{V}(\pi_{u,v})$
13: Run dual optimizer over variables in $\mathcal{B}, \mathcal{T}, \mathcal{P}$
14: **until** $\mathcal{B}, \mathcal{T}, \mathcal{P}$ stabilize or test accuracy saturates

Figure 1: Constraint inclusion heuristic. Here $\arg\max^{(k)}$ selects the arguments corresponding to top-k values.

Figure 1 shows the dual variable inclusion heuristic. Unlike StructSVM [23] we wish to include not one but several violators, because we do not have an exponential number of dual variables, and in comparison the relatively heavyweight optimizer needs to be run after every inclusion step. Note that the parameters k and ϵ which control the number of variables that will be included in an optimization step are crucial to the success of the algorithm. Too small a value of k will lead to a prohibitively large number of iterations to induct a sufficient number of constraints for an acceptable quality of solution. Too large a k can lead to the induction of an extremely large number of constraints, thereby defeating the purpose. Similar arguments hold for ϵ.

We adaptively tune k and ϵ so that, even if their initial values are not very good, we can quickly reach a reasonable value. Every time the number of violators found above the ϵ threshold is greater than k, we increment k. This allows us to start off with a conservatively small k. For adapting ϵ, when we see that the number of variables being inducted is extremely low for several consecutive iterations, we increase ϵ. This is based on our observation that towards the end, the optimizer drags on, adding very few violators per iteration, and hardly improving in the quality of solution. Hence, we increase ϵ so that only significant violators, if any left, are inducted and can make a perceptible change in the quality of solution. The exact formulae by which we set k and ϵ are deferred to an extended version of this paper [18].

Proposition 2. *The primal problem in Section 3.1 can be superficially rewritten to represent all dual variables β, τ and π collectively as a vector $\lambda = (\lambda_j)$ with j ranging over a suitable index space, and to express*

$$p_{uv} = \frac{q_{uv}}{Z_\lambda} \exp\left(\sum_j \lambda_j f_j(u,v)\right) = \frac{q_{uv}}{Z_\lambda} \exp(\lambda' \mathbf{f}(u,v)), \quad (6)$$

$$Z_\lambda = \sum_{(u,v)\in E'} q_{uv} \exp(\lambda' \mathbf{f}(u,v)) \quad (7)$$

where $f_j(u,v) \in [0,2]$ are features *that encode the contributions of various λ_js to p_{uv}. The modified dual objective to maximize is*

$$\max_{\{\lambda_j\}} \left\{ -\log Z + \textstyle\sum_j \nu_j \lambda_j \right\} = \max_\lambda \{ -\log Z + \nu \cdot \lambda \}$$

where each ν_j is a fixed small constant.

We can also show the following important guarantee.

Proposition 3. *Suppose vector $\lambda^{(\ell-1)}$ is updated to $\lambda^{(\ell)}$ in the ℓth step of the dual variable inclusion algorithm shown in Figure 1. Assume that $\lambda^{(\ell)}$ is the same as $\lambda^{(\ell-1)}$ except for newly-included dual variables, which are greedily set to values that maximize the dual objective. Then, for $\epsilon > 0$ and all $f_j(u,v) \in [0,2]$,*

$$-\log Z_{\lambda^{(\ell-1)}} + \nu' \lambda^{(\ell-1)} < -\log Z_{\lambda^{(\ell)}} + \nu' \lambda^{(\ell)},$$

i.e. the dual optimization makes monotonic progress.

The proofs can be found in the full version of this paper [18]. Therefore, the algorithm will terminate in a finite number of inclusion phases. We can also show that we will make a good progress when we are far away from the dual optimum, and make smaller progress when we approach close to it.

3.4 Experiments

For the problem we are studying there are no publicly available or widely-used benchmarks. Given the subtle interplay between E and $\prec$, a great deal of care is needed to generate these in a meaningful and realistic manner, so as to tease out the nature of the problem, the behavior of various algorithms, and the effects of different system parameters.

3.4.1 Graph generation using RMAT

Real social networks have many well-studied properties: degree and Pagerank distributions tend to be power-law [11], diameter is small (small-world phenomena), and there are clustered communities. To achieve these goals, we used the RMAT graph generator [7]. RMAT populates edges one by one, driven by four parameters b_{xy} with $x, y \in \{1,2\}$ and $\sum_{x,y} b_{xy} = 1$. Starting with source and destination node ranges $[1,n]$, RMAT bisects each range and picks quadrant (x,y) with probability b_{xy}, and then recurses until only one source and one destination node remain, at which point an edge is added. In all our experiments, we used $b_{11} = 0.48$, $b_{12} = b_{21} = 0.16$, and $b_{22} = 0.20$, giving us graphs with characteristic clustering and power-law degree distributions

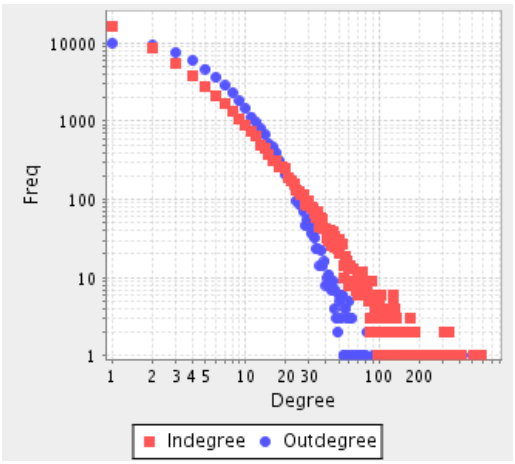

Figure 2: Characteristic near-power-law degree distribution of the DBLP+CiteSeer graph.

very similar to real data from DBLP+CiteSeer, shown in Figure 2.

We also experimented with multiple overlapping graphs, each created using an RMAT invocation (as described in Section 4.2) and the results were subjectively similar.

3.4.2 *Hidden teleport and sampling* $\prec$

Perhaps the simplest "hidden cause" for $\prec_{\text{train}}$ disagreeing with flow q is that the user has a personal preference for an unknown region of G. (Tsoi *et al.* [24] make basically the same assumption.) After computing reference flow q, we "secretly" picked a seed node $u^* \in V_o$ and sent it a relatively large flow from the dummy node d, say $r_{u^*} = p(u^*|d) = 0.1$. We divided the remaining teleport mass of 0.9 equally among other $v \in V$. This gave us our "hidden" flow p^*.

In applications, users are more likely to provide feedback on, and benefit from, the ranking of nodes near the top of the lists ordered by q and p^* scores, rather than low-scoring nodes. (For any flow p or q, the total inflow into a node v is its "score," written as p_v or q_v.) Accordingly, we prepared two sorted lists, and considered all distinct node pairs (u, v) drawn from a large prefix over each list. If $q_u \le q_v$ and $p^*_u \le p^*_v$ or $q_u \ge q_v$ and $p^*_u \ge p^*_v$, we called it an *agreement* between q and p^*; the other two cases are *disagreements*. Using reservoirs, we sampled a fixed number of agreements and (an equal number unless specified) of disagreements, which together constitute $\prec_{\text{train}}$. $\prec_{\text{test}}$ was collected similarly. This generally led to an overlap of the node set involved in $\prec_{\text{train}}$ and $\prec_{\text{test}}$ (we always ensured $\prec_{\text{train}} \cap \prec_{\text{test}} = \varnothing$), but if this was undesired, we partitioned the node set ahead of time (say odd and even node IDs) and sampled $\prec_{\text{train}}$ from one and $\prec_{\text{test}}$ from the other.

We also experimented with multiple hidden favored seeds, and also with hidden, well-connected communities having high-conductivity edges grown around the seeds. The results were qualitatively similar.

3.4.3 *Results*

Dual optimization dynamics: If we initialize all dual variables at zero, the initial primal flow p is equal to q, which satisfies all (Balance) and (Teleport) constraints. However, as the optimizer seeks to respect $\prec$, many primal constraints are abruptly violated, major flow readjustments take place, and the violations reduce. Gradually, egregious primal violations become rare, as shown in Figure 3. A meaningful primal solution can be read off only at this stage, and BLMVM termination has to take care to monitor primal violations over and above dual objective saturation.

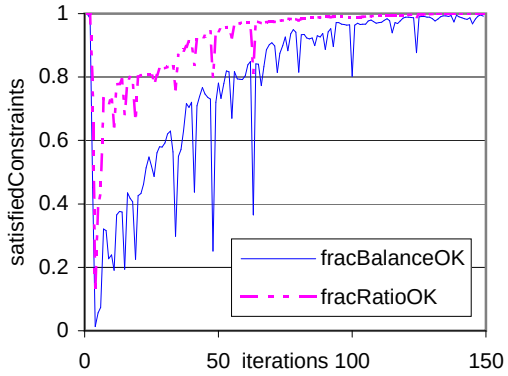

Figure 3: Satisfaction of primal feasibility constraints (Balance) **and** (Teleport) **as dual optimization progresses.**

Learning rate: We first sampled a fixed graph using RMAT, with $|V| = 1000$ and $|E| = 4644$. Then we created some 10 separate problem instances by picking 10 hidden seeds v^* at random to favor with a teleport of $r_{v^*} = 0.1$ as described before.

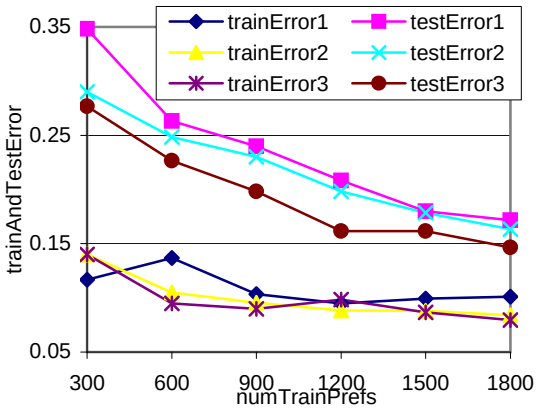

Figure 4: Reduction in test error as training $|\prec|$ is increased, for three random choices of the hidden teleport seed.

For each problem instance, we first selected a fixed $\prec_{\text{test}}$ of size 600 (pairs). Then we picked $\prec_{\text{train}}$ of sizes 300, 600, 900, 1200, 1500, and 1800 pairs, and plotted training and test error, as a fraction of the total number of pairs, in Figure 4 (only three representative instances are shown, but they give some idea of the observed variance).

Effect of node overlap: As we picked larger and larger $\prec_{\text{train}}$ in Figure 4, the set of nodes involved in $\prec_{\text{test}}$ started

overlapping with the set of nodes involved in $\prec_{\text{train}}$, although we obviously ensured $\prec_{\text{train}} \cap \prec_{\text{test}} = \varnothing$ at all times.

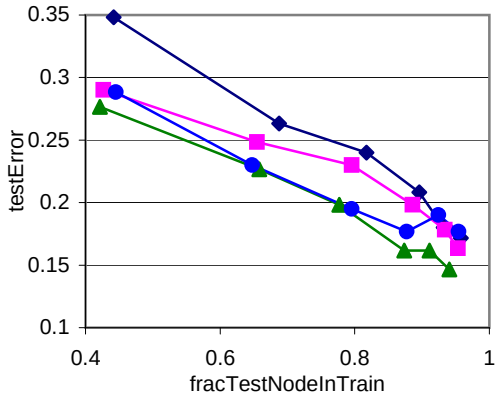

Figure 5: Effect of overlap between nodes involved in $\prec_{\text{train}}$ and $\prec_{\text{test}}$ on test error. Four random teleport seeds were used.

For several different hidden teleport seeds, we increased the size of $\prec_{\text{train}}$ and plotted, in Figure 5, the test error against the fraction of nodes involved in $\prec_{\text{test}}$ that also appeared in $\prec_{\text{train}}$. In search applications, users are typically focused on specific communities, and have no need to rank nodes far from and unrelated to nodes about which they already have ranking opinions.

Comparison with QP teleport optimization: In their experiments, Tsoi *et al.* [24] first computed (UnweightedPagerank), and then picked a pair of nodes (typically, one was highly ranked, the other not) and flipped their order to produce a $\prec_{\text{train}}$ with only one pair. Their goal was to study the effect of this inversion on various clusters of G.

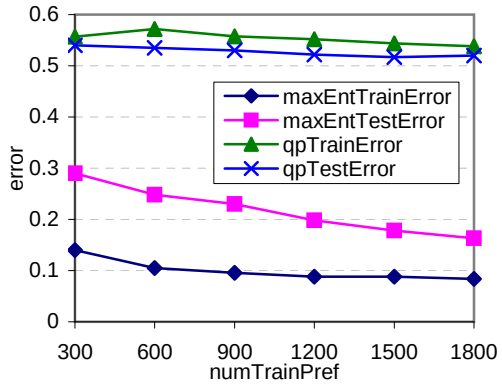

Figure 6: Comparison of maxent flow with QP teleport tuning.

Used in our setting, the QP formulation of Section 2.3 performs surprisingly poorly (Figure 6), with an error rate comparable to random guessing, even if node overlap between $\prec_{\text{train}}$ and $\prec_{\text{test}}$ is allowed. For five out of ten choices of the random favored teleport seed, the QP optimization assigned *zero* teleport to the secret favored node. In contrast, in all ten cases, our algorithm assigned a positive primal inflow into the secret favored node.

QP with slack variables: Anecdotally, our modified QP (4) with slack variables gives much better solutions, but is computationally very expensive because it has not $|V|$ but $|V|+|\prec|$ variables and the constraints are more challenging than a symmetric square loss. Compared to our two algorithms, the quadratic programming approach, which also involves a matrix inversion to get M, appear impractical.

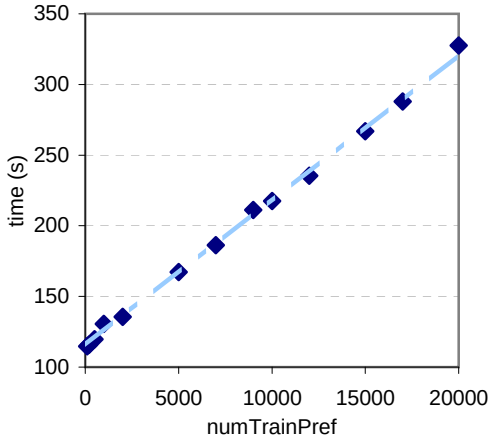

Figure 7: Flow optimization time scales linearly with $\prec_{\text{train}}$.

Performance scaling: Figures 7 and 8 show that a dual optimization involving all dual variables takes time roughly linear in $|\prec|$, $|V|$ and $|E|$. In Figure 7 G was fixed and $\prec_{\text{train}}$ was scaled. In Figure 8 $\prec_{\text{train}}$ was fixed and $|V|$ and $|E|$ scaled separately.

Savings from variable inclusion: We used a baseline graph with 21000 nodes and about 42000 edges, and scaled up $|V|$, $|E|$ and $|\prec|$ in tandem. Figure 9 shows the running time of the one-shot dual optimizer and the total time of the multi-round dual variable inclusion strategy given in Figure 1. As the problem size scales up, we get bigger and bigger gains from the variable selection strategy.

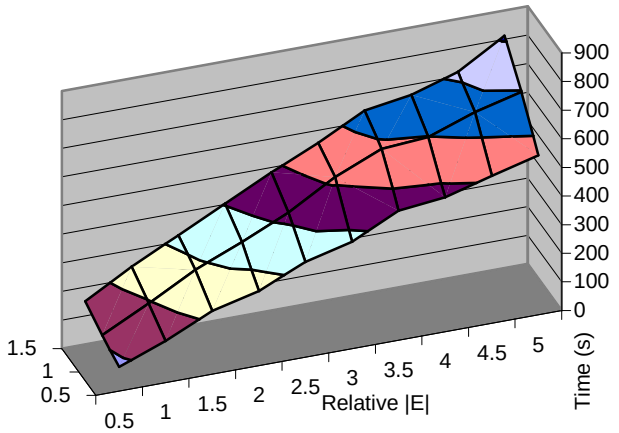

Figure 8: Flow optimization time scales roughly linearly with $|V|$ (relative sizes 0.5, 1, 1.5 shown) and with $|E|$ (relative sizes 0.5–5 shown).

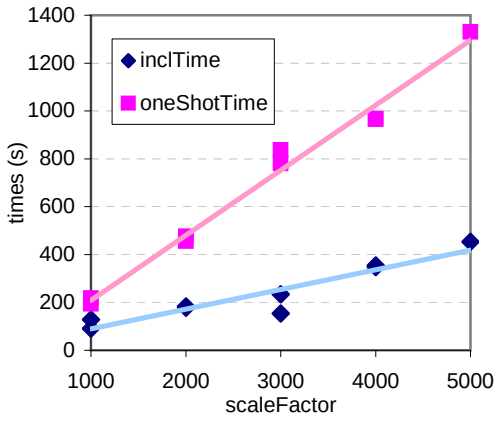

Figure 9: Running time of the one-shot dual optimizer vs. the gradual inclusion strategy. The x-axis is $|\prec|$; $|V|$ and $|E|$ are scaled up proportionately.

4. LEARNING EDGE CONDUCTANCES

In this Section we address the problem of learning weights for each edge type from $\prec$.

4.1 Approximate gradient descent

The conductance matrix C used in (UnweightedPagerank) is modified to reflect edge weights, as follows:

$$C(j,i) = \begin{cases} \frac{\alpha\beta(t(i,j))}{\sum_{j'}\beta(t(i,j'))}, & (i,j) \in E \\ 1-\alpha[\![i \in V_o]\!], & i \in V, j = d, \\ r_j & i = d, j \in V \\ 0, & \text{otherwise} \end{cases} \quad \text{(Conductance)}$$

Here d is the dummy node and $r = (r_j)$ is the teleport vector as before. Note that C is a function of β, and we seek a set of βs such the p solves $p = Cp$ and p satisfies $\prec$.

As in soft-margin approaches, we again turn the latter hard constraint into a part of the objective that penalizes violations of $\prec$. The transformation of (Preference) into (SoftPreference) and (SoftObjective) essentially adds a violation penalty

$$B\sum_{u\prec v}\text{loss}(p_u - p_v) = B\sum_{u\prec v}\max\{0, p_u - p_v\}; \quad (8)$$

note that if $p_u \le p_v$ as $\prec$ wants, no penalty is incurred. Two problems remain: the max function is not differentiable at zero, and p_u cannot be expressed easily in terms of β.

The first problem is common, and readily removed by approximating (8) with a everywhere-differentiable function such as the Huber penalty with window width W:

$$\text{loss}(y) = \begin{cases} 0, & y \le 0 \\ y^2/(2W), & y \in (0, W] \\ y - W/2, & W < y \end{cases} \quad (9)$$

Because we are searching for $\beta(t)$s, we will need to find the gradient of $\text{loss}(p_u - p_v)$ wrt $\beta(t)$ for each type t, which is $\text{loss}'(p_u - p_v)(\frac{\partial p_u}{\partial\beta(t)} - \frac{\partial p_u}{\partial\beta(t)})$, where $\text{loss}'(y)$ is the derivative of the rhs of (9). The only missing piece is $\partial p_u/\partial\beta(t)$ for each node u and type t. Let $g(u,t)$ be an approximation to $\partial p_u/\partial\beta(t)$.

1: Initialize all $p_v^{(0)} \leftarrow 1/|V'|$ and $g^{(0)}(v,t) = 0$ for all v, t
2: $\ell \leftarrow 0$
3: **while** any element of p or g changes significantly **do**
4: $\quad \ell \leftarrow \ell + 1$
5: $\quad$ for each u set $p_u^{(\ell)} \leftarrow \sum_v C(u,v)p_v^{(\ell-1)}$
6: $\quad$ **for** each node u and type t **do**
7: $\quad\quad g^{(\ell)}(u,t) \leftarrow \sum_v \frac{\partial C(u,v)}{\partial\beta(t)}p_v^{(\ell-1)} + C(u,v)g^{(\ell-1)}(v,t)$
8: $\quad$ **end for**
9: **end while**

Figure 10: Iterative approximation to $\partial p_u/\partial\beta(t)$.

We show in Figure 10 how to compute all the $g(u,t)$s by accompanying the regular Pagerank iterations with gradient-finding steps. This is just an application of chain rule iteration by iteration. $\frac{\partial C(u,v)}{\partial\beta(t)}$ is easily derived from (Conductance). Once we calculate p and g, we can evaluate the objective and gradient and use a Newton method like BLMVM [4].

From (Conductance) we see that scaling all βs by a fixed factor does not change C. To prevent any $C(i,j)$ from going to zero, we arbitrarily set the lower bound $\beta(t) \ge 1$ for all types t. We can also penalize large βs with a standard Ridge-penalty of the form $\beta'\beta$.

4.2 Experiments

4.2.1 *Generating realistic typed graphs*

Generating a synthetic graph through a single call to RMAT, and then randomly assigning types and weights, would lead to very unrealistic graphs that would look locally statistically homogeneous at all nodes wrt incident weights.

To generate natural graphs with typed nodes and edges, such as the DBLP or CiteSeer citation graphs, we first called RMAT with a single set of 10000 paper nodes, creating 86382 citation links between them. Then we created a separate set of 10000 author nodes, and called RMAT to connect papers and authors with 26280 edges. Similarly, we connected papers to 1000 venue nodes using 15930 edges. These numbers were derived from an informal study of the degree distribution of the DBLP and CiteSeer graphs (see Figure 2). We also experimented with a graph derived from IMDB (`http://imdb.com`) and the results were similar.

4.2.2 *Generating $\prec$ using hidden edge weights*

Edges connecting two node communities have a designated type, e.g., paper *written-by* author. As in several ER graph databases [5, 3] all edges logically exist in both directions. Another way to say this is that each edge has two types, e.g. an "author wrote paper" also has a "paper written-by author" in the reverse direction.

We first assigned all edges unit weights (all $\beta = 1$) and computed the reference flow q. Then we assigned the edges various hidden weights (default values were paper-author: 6, 10; paper-paper: 20; paper-venue: 1, 4), and computed the hidden flow p^*. Finally, as in Section 3.4, we sampled from the agreements and disagreements between q and p^* to get $\prec_{\text{train}}$ and $\prec_{\text{test}}$.

4.2.3 *Results*

In this section we give evidence that the approximate gradient-descent is very effective at recovering the hidden

parameters that led to $\prec$, in terms of both accuracy and speed. A direct comparison with Nie *et al.*'s system was not feasible because they use a sophisticated, highly-tuned simulated annealing approach whose code is not public, and their running times range into several hours [19, Figure 8] while our algorithm takes a few minutes.

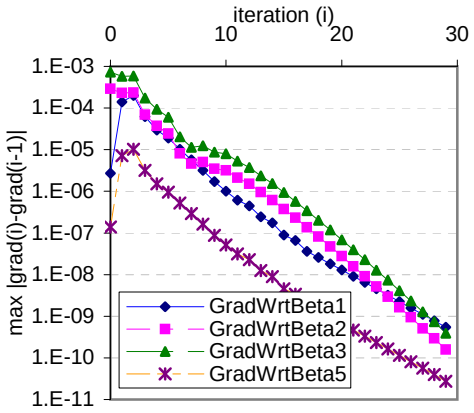

Figure 11: Like Pagerank itself, the gradients converge within very few iterations.

Gradient approximation: $\max_{v \in V} |g^{(\ell)}(v,t) - g^{(\ell-1)}(v,t)|$ is plotted against iterations ℓ for several edge types t in Figure 11. The difference between successive values decay exponentially, and convergence is achieved in practice between 30 and 50 iterations. We therefore feel confident to use these gradients in a gradient-descent procedure.

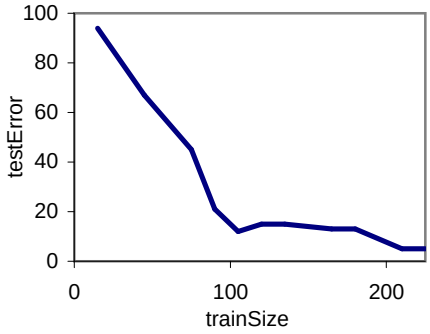

Figure 12: Reduction in test errors out of 2000 as $\prec_{\text{train}}$ is increased.

Learning rate: Figure 12 shows, for a fixed $\prec_{\text{test}}$ of size 2000, the test error as $\prec_{\text{train}}$ is increased. Unlike in the maxent flow approach, here node overlap between $\prec_{\text{train}}$ and $\prec_{\text{test}}$ had no systematic effect on test error, so we ensured zero node overlap between $\prec_{\text{train}}$ and $\prec_{\text{test}}$ throughout. Compared to the maxent flow setting, we are estimating only a handful of βs, so the size of $\prec_{\text{train}}$ needed to attain good test accuracy is much smaller.

Accuracy of estimating hidden βs: In another experiment, we varied 1–2 edges weights away from the defaults listed above, and saw if our algorithm can estimate values close to the hidden values. The results are shown in Figure 13. The prominent diagonal is reassuring. Thanks to the $\beta'\beta$ Ridge penalty, there is a downward pressure on some βs leading to the below-diagonal entries. However, we note that an infinite number of combinations of edge weights can lead to the same Pagerank ordering per (WeightedPagerank). Even where we underestimated a β, the effect on train or test error was negligible.

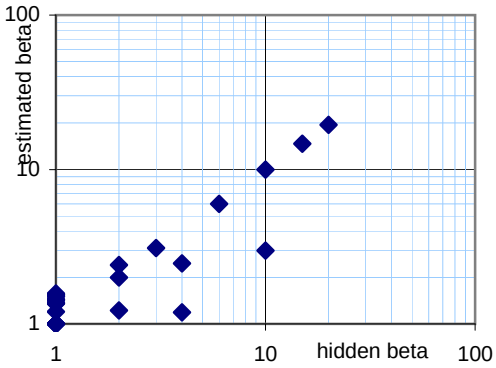

Figure 13: Accuracy of estimation of hidden βs.

Scalability: Figure 14 shows the increase of iterations and time per iteration as the graph size is scaled up. The time per iteration scales essentially linearly with $|V|$ and $|E|$, while the number of iterations is more erratic, but grows slowly with G. The overall result is that the training time is proportional to the scale factor raised to the power of about 1.34, which is mildly superlinear.

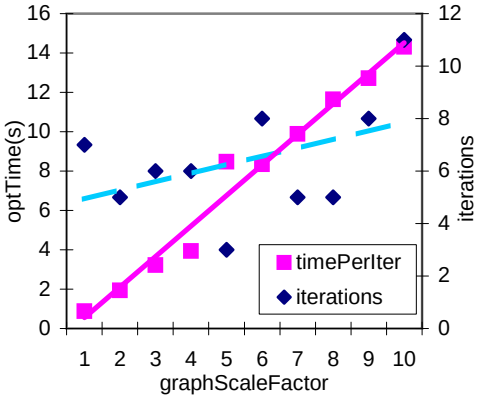

Figure 14: Scaling of running time with graph size. The x-axis represents the factor by which our synthetic DBLP graph's V and E were expanded.

5. CONCLUDING REMARKS

Most existing approaches to ranking entities involve learning weights for feature vectors, or Markovian walks with arbitrarily-designed conductance matrices. We have initiated the study of a uniform framework for learning the parameters of Markovian walks in graphs to satisfy pairwise preference constraints between nodes.

We presented two learning problems in this framework. In the first, the preferences hint at one or more favored communities that the learning algorithm must discover. We proposed a maximum entropy flow estimation algorithm for this

setting. In the second problem, edges have types that determine their conductance, and the learner must estimate these weights. We proposed an approximate gradient-descent algorithm for this setting. Our formulations enhance and generalize some previous approaches. We showed experimentally that our approaches are effective.

The flow approach has to estimate a large number of variables, scaling with G. The flow approach applies to settings where edges are not typed, and $\prec_{\text{train}}$ and $\prec_{\text{test}}$ are naturally clustered (as they would be in many relevance feedback or collaborative filtering applications). In contrast, the approximate gradient-descent approach estimates relatively few global weights, and can therefore generalize from $\prec_{\text{train}}$ to $\prec_{\text{test}}$ that involve completely different nodes, far away in G, with a much smaller number of examples. However, the second approach requires a notion of global edge types.

In ongoing work we are trying to go beyond just counting satisfied node-pairs to a more rank-aware objective that pays more importance to top-ranking nodes. We are also trying to extend the framework to integrate node feature vectors (e.g. text on Web pages) in an elegant manner.

6. REFERENCES

[1] S. Agarwal, C. Cortes, and R. Herbrich, editors. *Learning to Rank*, NIPS Workshop, 2005.

[2] K. Anywanwu, A. Maduko, and A. Sheth. SemRank: Ranking complex semantic relationship search results on the semantic Web. In *WWW Conference*, pages 117–127, Chiba, Japan, 2005.

[3] A. Balmin, V. Hristidis, and Y. Papakonstantinou. Authority-based keyword queries in databases using ObjectRank. In *VLDB*, Toronto, 2004.

[4] S. J. Benson and J. J. Moré. A limited memory variable metric method for bound constraint minimization. Technical Report ANL/MCS-P909-0901, Argonne National Laboratory, 2001.

[5] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *ICDE*. IEEE, 2002.

[6] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *WWW Conference*, 1998.

[7] D. Chakrabarti, Y. Zhan, and C. Faloutsos. R-MAT: A recursive model for graph mining. In *ICDM*. SIAM, 2004.

[8] H. Chang, D. Cohn, and A. McCallum. Creating customized authority lists. In *ICML*, 2000.

[9] W. W. Cohen, R. E. Shapire, and Y. Singer. Learning to order things. *JAIR*, 10:243–270, 1999.

[10] M. Diligenti, M. Gori, and M. Maggini. Learning Web page scores by error back-propagation. In *IJCAI*, 2005.

[11] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM*, pages 251–262, 1999.

[12] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: Ranked keyword search over XML documents. In *SIGMOD Conference*, pages 16–27, 2003.

[13] T. H. Haveliwala. Topic-sensitive PageRank. In *WWW*, pages 517–526, 2002.

[14] R. Herbrich, T. Graepel, and K. Obermayer. Support vector learning for ordinal regression. In *International Conference on Artificial Neural Networks*, pages 97–102, 1999.

[15] G. Jeh and J. Widom. Scaling personalized web search. In *WWW Conference*, pages 271–279, 2003.

[16] T. Joachims. Optimizing search engines using clickthrough data. In *SIGKDD Conference*. ACM, 2002.

[17] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *JACM*, 46(5):604–632, 1999.

[18] NetRank project home page. `http://www.cse.iitb.ac.in/~soumen/doc/netrank`, 2006.

[19] Z. Nie, Y. Zhang, J.-R. Wen, and W.-Y. Ma. Object-level ranking: Bringing order to Web objects. In *WWW Conference*, pages 567–574, 2005.

[20] M. Richardson and P. Domingos. The intelligent surfer: Probabilistic combination of link and content information in pagerank. In *NIPS 14*, pages 1441–1448, 2002.

[21] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.

[22] J. A. Tomlin. A new paradigm for ranking pages on the world wide Web. In *WWW Conference*, pages 350–355, 2003.

[23] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *JMLR*, 6(Sep):1453–1484, 2005.

[24] A. C. Tsoi, G. Morini, F. Scarselli, M. Hagenbuchner, and M. Maggini. Adaptive ranking of web pages. In *WWW Conference*, pages 356–365, 2003.