Learning Parameters in Entity Relationship Graphs from Ranking Preferences

Soumen Chakrabarti* and Alekh Agarwal

IIT Bombay
soumen@cse.iitb.ac.in

Abstract. Semi-structured entity-relation (ER) data graphs have diverse node and edge types representing entities (paper, person, company) and relations (wrote, works for). In addition, nodes contain text snippets. Extending from vector-space information retrieval, we wish to automatically learn ranking function for searching such typed graphs. User input is in the form of a partial preference order between pairs of nodes, associated with a query. We present a unified model for ranking in ER graphs, and propose an algorithm to learn the parameters of the model. Experiments with carefully-controlled synthetic data as well as real data (garnered using CiteSeer, DBLP and Google Scholar) show that our algorithm can satisfy training preferences and generalize to test preferences, and estimate meaningful model parameters that represent the relative importance of ER types.

1 Introduction

There is much recent interest [12,14,1] in learning to order entities represented by feature vectors, given a partial order $\prec$ involving some of the entities. The order is defined by a scoring function whose parameters have to be learned. A popular scoring function, suitable for ranking documents in Information Retrieval (IR), is an inner product $\beta' x_i$ between the feature vector x_i representing entity i and an estimated parameter vector β; if $i \prec j$, we want $\beta' x_i \leq \beta' x_j$.

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 [5,15], but also of entity-relationship (ER) graphs [4,2] and XML data [10] where nodes represent entities with textual attributes and edges represent diverse relations. A sample ER graph is shown in Fig. 1. Activation spreading or Pagerank-like Markovian random walks are often used to score nodes given a query. Typically, a database administrator has to assign and/or tune edge weights, which are used to bias the walks or activation propagation.

In Section 2 we will review a number of efforts to learn some of the parameters of the Markov walk system, most typically via heuristic search [17], quadratic programs [20] or local hill-climbing [7,9].

To our knowledge, no single existing approach covers the scenario we address in Section 3: User preference is provided as $\prec$ (not as absolute score targets

* Contact author.

J. Fürnkranz, T. Scheffer, and M. Spiliopoulou (Eds.): PKDD 2006, LNAI 4213, pp. 91–102, 2006.

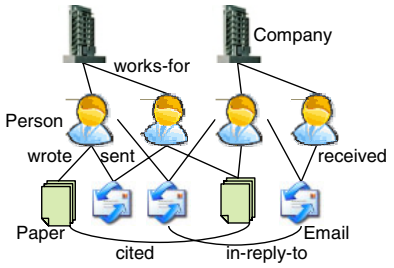

Fig. 1. ER graph with diverse node and edge types

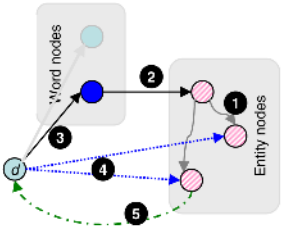

Fig. 2. Typed graph with word nodes

as in some previous work), and ranking is query-specific. We must learn to combine information from feature vectors associated with nodes as well as edge types to output a total order that agrees well with $\prec$ and generalizes to unseen preferences. As a by-product we learn a notion of relative conductivity of different edge types.

In Section 4 we carefully evaluate our proposed algorithm using synthetic data as well as real data from DBLP, CiteSeer and Google Scholar. Unlike some of the earlier work, we give a very detailed account of loss functions, constraints on parameters and model parsimony, the nature of the optimization surface and parameter search techniques. We will release our code and data in the public domain [16].

2 Related Work

Learning to rank feature vectors: Learning to rank items represented by feature vectors from partial orders or point-scale training input (ordinal regression) is well-explored in machine learning [12,14,1]. In RankSVM [14], a slack variable $s_{ij} \geq 0$ is introduced for each constraint $i \prec j$, and the preference is expressed as $\beta' x_j + s_{ij} \geq \beta' x_i + 1$, or, equivalently, $s_{ij} \geq \max\{0, 1 - (\beta' x_j - \beta' x_i)\} = \text{hinge}(1 + \beta' x_i - \beta' x_j)$, where $\text{hinge}(y) = \max\{0, y\}$ is the classic hinge loss. The objective is to minimize $\beta'\beta + B \sum_{i \prec j} s_{ij}$, where B is a magic parameter that trades off the norm of β against the severity of training set errors. Summarizing, RankSVM seeks to minimize $\beta'\beta + B \sum_{i \prec j} \text{hinge}(1 + \beta' x_i - \beta' x_j)$, which can be done using standard quadratic programming tools. However, observe that the training set error is just $\sum_{i \prec j} \text{step}(\beta' x_i - \beta' x_j)$, where $\text{step}(y) = [\![y > 0]\!]$. ($[\![I]\!]$ is 1 if condition I is true and 0 otherwise.) RankSVM upper bounds the training error with $\sum_{i \prec j} \text{hinge}(1 + \beta' x_i - \beta' x_j)$, which is more amenable to quadratic optimization.

Pagerank basics: We review the "random surfer" model of Pagerank [5] briefly. Items are now nodes in a graph $G = (V, E)$, not feature vectors. In the steady state, the random surfer is at node j with probability $p_j = \sum_i p_i p(j|i)$. If we write $p(j|i)$ as a transition or conductance matrix C, the row vector p solves $p = Cp$. C is designed as

$$C(j,i) = \begin{cases} \alpha \frac{[\![(i,j) \in E]\!]}{\text{OutDegree}(i)} + (1-\alpha) r_j, & i \notin \text{leaf}(V) \\ r_j, & \text{otherwise} \end{cases} \quad \text{(UnweightedConductance)}$$

leaf(V) is the set of dead-end nodes without outlinks. 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.

Follow-up work on Pagerank has attempted to modify the teleport vector r to "personalize" the scores heuristically, based on topics [11], words [18,2], or user preferences on graph nodes [13], but the transitions are designed by hand, not learnt from preference data.

Learning link-based ranking: Recently there have been efforts to learn r and even C automatically. Tsoi *et al.* [20] used a quadratic programming approach to optimize only the teleport vector r in (UnweightedConductance), but their formulation had only one kind of edges. Chang *et al.* [7] proposed tuning edge weights for HITS [15] using relevance feedback without a notion of edge types. Nie *et al.* [17] tried local search exhaustively over each edge type. Diligenti *et al.* [9] fit edge weights using back-propagation in a neural network, in case there are known absolute target scores for a few specific pages. In applications, it is easier to collect partial orders between nodes rather than absolute scores. None of the above approaches combine query and per-node feature vectors with link information.

Combining links and text in ranking: Pagerank [5] is precomputed on the entire Web graph and combined with text-based scores at query time in undocumented and proprietary ways. In HITS [15], query keywords drive a heuristic procedure for collecting a limited subgraph of the Web, which is then scored. We know of only a few attempts to combine link and text information for ranking (for the classification task much more is known) in a unified, principled manner. Cohn and Hofmann [8] propose an elegant joint generative model combining text and links, but leave the application to search and ranking unspecified. Silva *et al.* [19] extend Turtle and Croft's approach to IR scoring using Bayesian networks [21] to include link information, but as precomputed scores from standard HITS—the learning is limited to the Bayesian network and there is no learning associated with hyperlink edges.

3 Learning Markov Parameters from Preferences

We are given a directed graph $G = (V, E)$. Edge (i, j) has type $t(i, j)$ belonging to a set of types numbered $1, \ldots, T$. Each type t has an associated importance represented by a weight $\beta(t)$. Thus, edge (i, j) has weight $\beta(t(i, j))$. We will require that our learning algorithm keep these weights positive, to ensure that

the graph topology is not altered by effectively erasing these edges. We will revisit this issue in Section 3.4.

(UnweightedConductance) is modified to use weights as follows. As before, columns are source nodes and rows are destination nodes. Each column adds up to 1. Teleport is handled by adding a dummy node d to the graph, connecting each node $i \in V$ to d and back again, i.e., edges (i, d) and (d, i).

$$C(j,i) = \begin{cases} 0, & i \neq d, j \neq d, i \in \text{leaf}(V) \\ \alpha \frac{\beta(t(i,j))}{\sum_{j'} \beta(t(i,j'))}, & i \neq d, j \neq d, i \notin \text{leaf}(V) \\ 1, & i \neq d, j = d, i \in \text{leaf}(V) \\ 1-\alpha, & i \neq d, j = d, i \notin \text{leaf}(V) \\ r_j & i = d, j \neq d \\ 0, & i = d, j = d \end{cases} \quad \text{(WeightedConductance)}$$

Here "$i \in \text{leaf}(V)$" means i has no outlinks in G. The Pagerank vector $p \in \mathbb{R}^{(|V|+1)\times 1}$ for a given C satisfies $p = C\,p$. In addition, we wish p to satisfy $\prec$, i.e., for each $i \prec j$, we must have $p_i \leq p_j$. Unfortunately, searching over feasible values of both β and p together will introduce problematic quadratic constraints, thanks to the requirement $p = C\,p$.

3.1 Truncating the Recursion

To make the learning problem manageable, we truncate the recursion. In practice, the Pagerank vector is frequently computed via power iteration, by initializing $p = p^0 = \mathbf{1}^{(|V|+1)\times 1}/(|V|+1)$ and iterating $p \leftarrow Cp$ until convergence. The number of iterations needed, which we call the *horizon* H, depends strongly on α but is less sensitive to other aspects of C, and typically grows slowly with the size of the graph. As we shall see, $p = C^H p^0$ is often an excellent approximation, even for modest values of H (10–50). Therefore, the problem reduces to looking for β such that, for each $i \prec j$, $(C^H\,p^0)_i \leq (C^H\,p^0)_j$. Although we describe our learning procedure in terms of a fixed H, in an implementation we need not pick a fixed horizon, but iterate until a specific error tolerance is satisfied.

3.2 Choice of Loss Function

Next we will look at various choices of the loss function. We need to approximate $\sum_{i \prec j} \text{step}((C^H\,p^0)_i - (C^H\,p^0)_j)$. Because $\|p^0\|_1 = 1$ and columns of C add up to 1, $\|p\|_1 = 1$ as well. Therefore $-1 \leq (C^H\,p^0)_i - (C^H\,p^0)_j \leq 1$, and thus picking $\text{loss}(y) = \text{hinge}(1+y)$ is meaningless because we end up just minimizing $\sum_{i \prec j}(1+(C^H\,p^0)_i-(C^H\,p^0)_j)$ where a satisfied constraint contributes a *negative* amount to the sum.

(Note that this is a non-issue for RankSVM because there, $\|\beta\|_2$ can be inflated by the optimizer to prevail over these effects. Moreover, if the training input comprises absolute score targets [9] for nodes, rather than the more realistic partial order preferences, this problem does not arise.)

In optimizing conditional models, it is very common to replace the hinge loss with a "soft" hinge loss of the form $\log(1+e^y)$, which asymptotes to the hinge loss for large $|y|$. Yet another possibility is to directly approximate the step function $\text{step}(y)$ with the logit function $\text{logit}(y) = 1/(1+e^{-y})$. Unfortunately, neither soft hinge nor logit works for us. A detailed study showed that the reason is that they are both positive at $y = 0$ (and small negative values), adding a "noise floor" to the objective even for satisfied preferences, making it very hard for the optimizer to find a reliable gradient. It is very important that $\text{loss}(y)$ is exactly zero for $y \le 0$. After quite some experimentation we picked the Huber loss with window W given by

$$\text{huber}(y) = \begin{cases} 0, & y \le 0 \\ y^2/(2W), & y \in (0, W] \\ y - W/2, & W < y \end{cases}; \quad \text{huber}'(y) = \begin{cases} 0, & y \le 0 \\ y/W, & y \in (0, W] \\ 1, & W < y \end{cases}$$

3.3 Newton Method

All that remains to plug in our formulation into a Newton method is the computation of $(\partial/\partial\beta_t)\sum_{i \prec j} \text{loss}((C^H\, p^0)_i - (C^H\, p^0)_j)$, which is

$$\sum_{i \prec j} \text{loss}'\big((C^H\, p^0)_i - (C^H\, p^0)_j\big)\left(\partial(C^H\, p^0)_i/\partial\beta_t - \partial(C^H\, p^0)_j/\partial\beta_t\right).$$

We can compute this easily if we had a $(|V|+1)\times T$ matrix of Pagerank gradients $\frac{\partial}{\partial\beta_t}(C^H\, p^0)_i$ where $i = 1, \ldots, |V|+1$ and $t = 1, \ldots, T$. This matrix can be built up inductively over $h = 0, \ldots, H$ as follows:

$$\frac{\partial}{\partial\beta_t}(C^0 p^0)_i = 0 \quad \text{for all } t \text{ and } i, \tag{1}$$

and for $h = 1, \ldots, H$:

$$\frac{\partial}{\partial\beta_t}(C^h p^0)_i = \sum_j \left[\frac{\partial C(i,j)}{\partial\beta_t}(C^{h-1} p^0)_j + C(i,j)\frac{\partial}{\partial\beta_t}(C^{h-1} p^0)_j\right] \quad \text{(ChainRule)}$$

Finally we compute $\frac{\partial C(i,j)}{\partial\beta_\tau}$ from (WeightedConductance), where the only interesting case is $i \ne d, j \ne d, i \notin \text{leaf}(V)$:

$$\frac{\partial C(i,j)}{\partial\beta_\tau} = \begin{cases} -\alpha\dfrac{\beta(t(i,j))\sum_w [\![\tau = t(i,w)]\!]}{(\sum_w \beta(t(i,w)))^2} & \tau \ne t(i,j) \\ \alpha\dfrac{\sum_w \beta(t(i,w)) - \beta(t(i,j))\sum_w [\![\tau = t(i,w)]\!]}{(\sum_w \beta(t(i,w)))^2}, & \tau = t(i,j) \end{cases} \tag{2}$$

In case we wish to also make α a variable in the optimization, (ChainRule) carries over unchanged, and the nonzero derivatives of conductance are:

$$\frac{\partial C(i,j)}{\partial\alpha} = \begin{cases} \frac{\beta(t(i,j))}{\sum_w \beta(t(i,w))}, & i \ne d, j \ne d, i \notin \text{leaf}(V) \\ -1, & i \ne d, j = d, i \notin \text{leaf}(V) \end{cases}$$

Each iteration of the Newton update takes $O((|V| + |E|)\, H)$ floating point operations. $O(T\,|V|)$ main memory is adequate; the edge list E can be scanned sequentially from disk. We use the BLMVM optimizer [3].

3.4 Keeping the Model Parsimonious

In RankSVM, the model parameters are penalized by a $\beta'\beta$ terms in the objective, which is equivalent to imposing a Gaussian prior with zero mean on each element of β. Zero mean does not make sense for us. In fact, any β_t going to zero may change the topology of G and its Markov properties in serious ways. We can see at least two reasonable choices for penalizing model complexity.

Centered at 1: If we consider Equation (UnweightedConductance) as the most parsimonious model, we should center the β_ts at 1. E.g., we might assess a penalty of $\sum_t (\beta_t - 1)^2$, choosing square loss for computational simplicity. For the reasons above we also need to lower bound each β_t strictly away from zero, which involves yet another magic number that we do not like.

Scale-free floating: There is nothing special about 1 as center; given a solution, we can scale all β_ts by a factor and C in Equation (WeightedConductance) (and therefore p) will remain unchanged. We can therefore lower bound all $\beta_t \geq 1$ without loss of generality, and modify the penalty to try to keep all the β_ts close together without centering any of them: $\sum_{t,t'} (\beta_t - \beta_{t'})^2$. Suppose there are two solution vectors, one a constant multiple of the other. The violations and losses are the same, but the solution with smaller magnitude has lower model penalty, so we will naturally prefer that one.

3.5 Incorporating Queries and Node Features

Much recent work on algorithms to learn ranking functions have used an intrinsic feature vector representation of the objects being ranked, whereas we have, thus far, ignored the objects and considered only the graph that connect them. There was also no notion of a query.

We propose two ways to incorporate node features and queries into our framework. The first implements teleport through word nodes and resembles ObjectRank [2] and the intelligent surfer [18]. The second, more pedestrian approach fits a linear combination of Pagerank-induced and node feature-induced scores.

Teleport through word nodes: We assume each node has a set of associated words, and the query is also a set of words. As shown in Fig. 2, we introduce a node for every word. The dummy node d is connected only to nodes corresponding to query words (edge type "3") and other word nodes are deleted. Each matched word node is connected to all entity nodes where the word occurs (edge type "2"). Entity nodes are interconnected as in the rest of this paper (collectively marked as edge type "1" although there could be more than one type of edges here). d also connects to entities directly (edge type "4")—this sets up a competition

between text match and network prestige. Finally, entity nodes teleport back to d as usual (edge type "5"). Our algorithm has to be invoked on this graph for each query and query-specific preferences.

Linear score combination: The above approach retains a "pure Markovian flavor", but requires a query-time Pagerank computation, which is expensive. Practical implementations are more likely to adopt our second approach.

In this approach we first compute a match function $\mu(q, i) \in \mathbb{R}^+$ between the query and the node features. E.g., each node may be a document, and the query and documents may be represented in vector space and μ may be the commonly-used cosine similarity between q and the text of node i. For a fixed query, we will omit q and use μ_i in place of $\mu(q, i)$. For simplicity we will assume that the text score vector μ has been scaled so that $\|\mu\|_1 = 1$.

We use uniform teleport to compute $C^H p^0$, but integrate signal from μ by writing the score vector as $p = (1-\gamma)C^H p^0 + \gamma\mu$, where $\gamma \in [0, 1]$ is part of the optimization, with

$$\frac{\partial}{\partial\gamma} \sum_{i \prec j} \text{loss}(p_i - p_j) = \sum_{i \prec j} \text{loss}'(p_i - p_j) \left[\mu_i - \mu_j + (C^H p^0)_j - (C^H p^0)_i\right].$$

4 Experiments

Here are the main steps of our evaluation scheme:

1. Get G from real data or a synthetic graph generator.
2. Get $\prec$ from real data and do a test-train split, or, for synthetic generation:
   (a) Compute unweighted Pagerank p_{uw} on G using (UnweightedConductance).
   (b) Assign hidden parameters β (and possibly α and γ), and compute weighted Pagerank p_{w} using the hidden parameters and (WeightedConductance).
   (c) Draw stratified samples (typically 1:1 for us) from agreements and disagreements between scores of node pairs as per p_{uw} and p_{w}. This reflects the reasonable null hypothesis of (UnweightedConductance), and the belief that training data must expose the implausibility of the null hypothesis.
3. Give our algorithm G and $\prec_{\text{train}}$ but not the hidden weights.
4. Our algorithm estimates β^* (and possibly α^* and γ^*).
5. Compute weighted Pagerank p^*_{w} using these estimated parameters.
6. Evaluate what fraction of $\prec_{\text{test}}$ is satisfied by p^*_{w}.

Graphs: We experimented with synthetic and real-life graphs. SynthDBLP, a synthetic citation graph, had author, affiliation and paper nodes (total 21000) connected by "works-for", "wrote" and "cited" edges (total 128592). SynthIMDB, a synthetic graph modeling `http://imdb.com`, had actor, movie, and genre nodes (total 21000) with "acted-in" and "belongs-to" edges (total 97121). We used RMAT [6] to generate the synthetic graphs that satisfy typical properties of social networks. The real citation graph curated from DBLP and CiteSeer had author, paper, and venue (conference/journal) nodes (total 147870) and "cited", "wrote" and "appeared-in" edges (total 1145393).

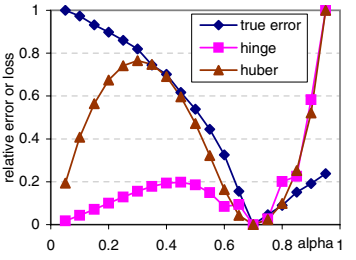

Fig. 3. Training error and losses vs. α ($\alpha_{\text{hidden}} = 0.7$)

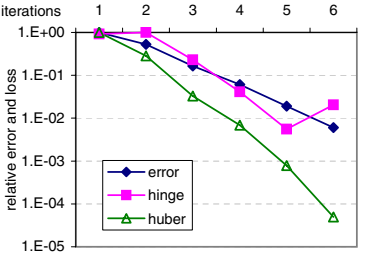

Fig. 4. Training error and losses vs. optimizer iterations

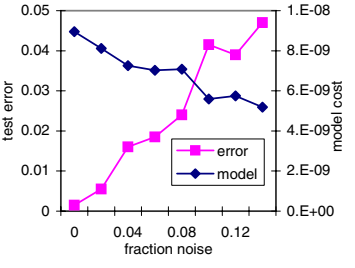

Fig. 5. Test error and model penalty vs. training noise

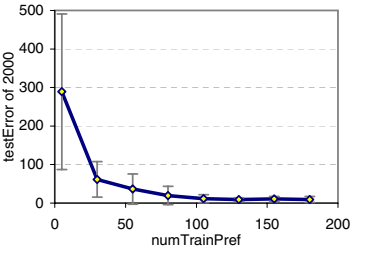

Fig. 6. Test error and its standard deviation (10 runs) vs. $|\prec_{\text{train}}|$

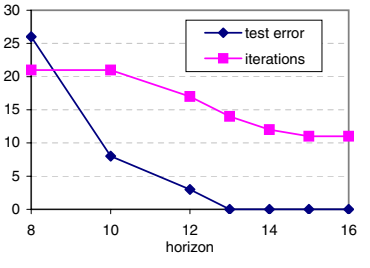

Fig. 7. Effect of H on convergence and test error

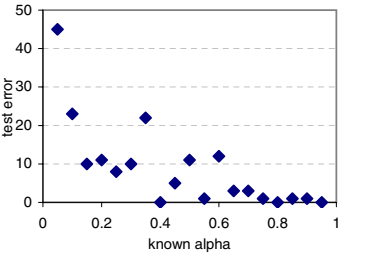

Fig. 8. Test error vs. α

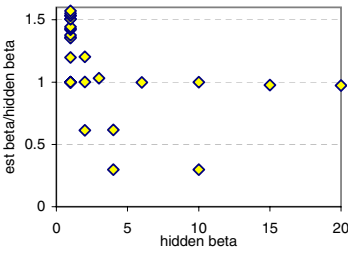

Fig. 9. β estimation accuracy

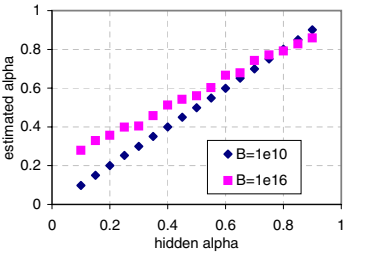

Fig. 10. Estimated vs. hidden α

Edges and weights: Except for experiments involving teleport through word nodes, each edge is made bidirectional, and "hidden" weights are associated with each direction to reflect heuristic values used in the literature [4,2]. Part of our goal is to see if the algorithm can discover these hidden weights given preference data. (Details about data generation can be found in an extended version of this paper [16].)

Error, loss, and convergence: The success of the optimizer depends on how closely our loss approximation tracks training loss. Our general experience is that Huber loss works well for large α (say, larger than 0.4) typically used in Pagerank, but deteriorates at small α. This is not a problem with Huber loss *per se*, because hinge loss (without a margin, defined as $\sum_{i \prec j} \max\{0, p_i - p_j\}$) is an even poorer approximation. Fig. 3 shows training error, hinge loss and Huber loss for variate α. For values of α commonly used in Pagerank algorithms, Huber loss gave reliable convergence (Fig. 4).

Robustness to noisy training data: In real life, our hypothesis that ranking is determined by weighted edges may not hold, and relevance feedback may not even be consistent. Our algorithm seems very robust to random flipping ($i \prec j$ replaced by $j \prec i$) of training pairs (Fig. 5). Even when over 20% of $\prec$ has been corrupted, the error on (clean) test data is less than 6%. It is also reassuring to note that as noise increases, the algorithm cuts back on investments in model complexity (measured as the floating penalty).

Learning rate and validation: To check if the model generalizes well, we generated $\prec_{\text{train}}$ and $\prec_{\text{test}}$ to be *node disjoint*, so that our algorithm cannot, e.g., benefit from discovering transitivity. We increased $\mid \prec_{\text{train}} \mid$ and plotted the test error in Fig. 6. Just a few hundred preference pairs appear adequate to learn a model that generalizes well.

Effect of horizon choice H: Is the truncation of iterations in Section 3.1 reasonable for both objective and gradient? Fig. 7 shows the effects of varying horizon H on the number of iterations to convergence and the error rate (out of 4000 node pairs). As H is increased, the objective and gradient estimates become more accurate (but computationally expensive) and the Newton method converges in fewer iterations. Furthermore, the edge model learnt is more accurate and therefore test error reduces. Even for larger real-life graphs, it appeared that $H > 30$ is always sufficient.

Effect of known α: As α goes to zero, edge weight tuning struggles harder and test error goes up, although, even at $\alpha = 0.05$, test error is less than 5%. Note that in Fig. 8, the algorithm knows α and estimates only the βs.

Edge weight (β) discovery: Fixing G, we varied 2–3 hidden β_ts at a time and ran our algorithm. Recall that all β_ts can be scaled arbitrarily without changing conductance C. In principle, our model penalty should force an automatic scaling

down, but the complex optimization surface can prevent the scaling down once training error is minimized. Therefore we scaled all β_ts by their minimum value. In Fig. 9 we plot the ratio of estimated to hidden β_ts against the hidden value.

Typically the two largest β_ts are estimated very well, but, thanks to our regularization scheme, there is a slight upward pressure on small values and a downward pressure on large values. However, note that the optimization problem is fundamentally degenerate in that, while we start from a specific hidden β, the same Pagerank ordering can be achieved by an infinite number of β vectors, so the deviations below the diagonal hurt neither training nor test error.

Combined α, β search: Fig. 3 shows that the approximate loss surface has local minima. Therefore, a Newton method will need multiple restarts to locate the global optimum. We varied α between 0 and 1, but this was hidden (as was β) from the algorithm, which had to estimate α and β together.

Fig. 10 shows that α is reconstructed with surprising perfection, despite our ignorance of both α and β_ts, (the latter were all initialized to 2, which always led us to the global optimum for any fixed α—this, and the much better quality of reconstructing α, merit a future study). Compared to the fixed α case, more care was needed with B to avoid overfitting.

We performed all our experiments above using synthetic $\prec$ on both synthetic graphs and the real graph culled from DBLP and CiteSeer, and all the results were very similar and consistent.

Integrating queries and text match: We collected queries in the areas of databases and XML (Fig. 11). First, for each query, we pinned down all edge weights except for dummy-to-word edges, which we varied to inspect the rankings obtained. Fig. 12 shows the results. For small dummy-to-word weights, text match is ignored and generic classic papers are listed at the top, whereas at larger weights, the query gets more attention (but citations are still important). Given there are only a handful of query words and about 80000 papers, naturally the dummy-to-word edges need to be quite heavy to have an effect.

Second, we set a hidden weight for dummy-to-word edges and fixed all other edge weights at 1, and tested if our algorithm can discover the hidden weight. The results broadly paralleled our study on other $\beta(t)$s and are omitted. The accuracy was not as good as in Fig. 9. Overestimates like $\beta_{\text{hidden}} = 100$ and $\beta_{\text{estimated}} \approx 20000$ were seen, but training and test errors went down reliably as before. Therefore, the "teleport through word nodes" model works as intended.

Third, we sent the queries to Google Scholar (`http://scholar.google.com`) and sampled the (prefix of the) total order returned to derive $\prec_{\text{train}}$ and $\prec_{\text{test}}$.

```
1: database xml structure index inverted, 2: "data streams" "query processing",
3: database concurrency control deadlock handling, 4: recovery shadow paging,
5: relation nested subquery optimization, 6: transaction serializability,
7: query processing sensor networks, 8: set "similarity join", 9: xml twig join,
10: heterogeneous schema integration "machine learning"
```

Fig. 11. Queries for DBLP and CiteSeer

`transaction serializability`, $\beta(\text{dummy} \rightarrow \text{word}) = 1$
Graph based algorithms for boolean function manipulation (506)
Scheduling algorithms for multiprogramming in a hard real time environment (413)
A method for obtaining digital signatures and public key cryptosystems (312)
Rewrite systems (265)
Tcl and the Tk toolkit (242)
`transaction serializability`, $\beta(\text{dummy} \rightarrow \textit{word}) = 10^6$
On serializability of multidatabase transactions through forced local conflicts (38)
Autonomous transaction execution with epsilon serializability (6)
The serializability of concurrent database updates (104)
Serializability a correctness criterion for global concurrency control in interbase (41)
Using tickets to enforce the serializability of multidatabase transactions (12)

Fig. 12. β(dummy→word) gives a learnable trade-off between word match and citation popularity. Top paper nodes shown with number of citations.

We injected this $\prec$ into the "teleport through word nodes" model with only one variable edge weight, β(dummy → word); there was no known "ground truth". Estimates were between 1 and 2 in 4 of 10 cases, and 21, 127, 172, 509, 650, and 6686 for the others. Training error was typically around 25% but test error was higher, around 35–40%; see comments at the end of this Section.

Finally, to test the "linear score combination" model, we used the IR TFIDF cosine score between the query and the paper titles to obtain μ, and used $\prec$ from Google Scholar. We set a few values of α by hand and estimated γ and β. For small α, G asserts little effect, the Pagerank distribution is very flat, so the optimizer can afford and prefers very small γs. For large α, G provides some valuable signal (absolute error goes down nicely), but it becomes more important to emphasize text: γ rises substantially (Fig. 13).

$\alpha = 0.05$

Q#	$\lvert\prec_{\text{test}}\rvert$	Errors	γ
q1	945	390	0.0272911
q3	771	310	0.0272881
q5	1008	406	0.0209949
q4	993	409	0.0272896

$\alpha = 0.7$

Q#	$\lvert\prec_{\text{test}}\rvert$	Errors	γ
q1	945	326	0.9888147
q3	771	219	0.9364531
q5	1008	382	0.6016729
q4	993	343	0.9833497

Fig. 13. Fitting γ with fixed α

The results involving $\prec$ from Google Scholar are preliminary and come with an important caveat: Google Scholar uses a. paper body text and b. a much larger and different graph compared to our sample of DBLP and CiteSeer, so its ranking function is using information not accessible to us.

5 Conclusion

We have presented models and numerical methods for learning Markov and other parameters for ranking nodes in ER graphs from partial order preferences. The optimization surfaces involved are not always benign, but they must be searched satisfactorily, given the widespread and increasing applicability of such models. We initiate an in-depth treatment of the choice of loss functions, optimization surfaces, search procedures, and parameter settings. In ongoing work we are

investigating text-match models more deeply and seeking to extend the loss framework to become rank-sensitive.

References

1. S. Agarwal, C. Cortes, and R. Herbrich, editors. *Learning to Rank*, NIPS Workshop, 2005.
2. A. Balmin, V. Hristidis, and Y. Papakonstantinou. Authority-based keyword queries in databases using ObjectRank. In *VLDB*, Toronto, 2004.
3. 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.
4. G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *ICDE*. IEEE, 2002.
5. S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *WWW Conference*, 1998.
6. D. Chakrabarti, Y. Zhan, and C. Faloutsos. R-MAT: A recursive model for graph mining. In *ICDM*. SIAM, 2004.
7. H. Chang, D. Cohn, and A. McCallum. Creating customized authority lists. In *ICML*, 2000.
8. D. Cohn and T. Hofmann. The missing link — a probabilistic model of document content and hypertext connectivity. In *NIPS*, 2001.
9. M. Diligenti, M. Gori, and M. Maggini. Learning Web page scores by error back-propagation. In *IJCAI*, 2005.
10. L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: Ranked keyword search over XML documents. In *SIGMOD Conference*, pages 16–27, 2003.
11. T. H. Haveliwala. Topic-sensitive PageRank. In *WWW*, pages 517–526, 2002.
12. R. Herbrich, T. Graepel, and K. Obermayer. Support vector learning for ordinal regression. In *International Conference on Artificial Neural Networks*, pages 97–102, 1999.
13. G. Jeh and J. Widom. Scaling personalized web search. In *WWW Conference*, pages 271–279, 2003.
14. T. Joachims. Optimizing search engines using clickthrough data. In *SIGKDD Conference*. ACM, 2002.
15. J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *JACM*, 46(5):604–632, 1999.
16. NetRank project. `http://www.cse.iitb.ac.in/~soumen/doc/netrank`, 2006.
17. 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.
18. M. Richardson and P. Domingos. The intelligent surfer: Probabilistic combination of link and content information in pagerank. In *NIPS 14*, pages 1441–1448, 2002.
19. I. Silva, B. Ribeiro-Neto, P. Calado, E. Moura, and N. Ziviani. Link-based and content-based evidential information in a belief network model. In *SIGIR Conference*, pages 96–103, 2000.
20. A. C. Tsoi, G. Morini, F. Scarselli, M. Hagenbuchner, and M. Maggini. Adaptive ranking of web pages. In *WWW Conference*, pages 356–365, 2003.
21. H. R. Turtle and W. B. Croft. Evaluation of an inference network-based retrieval model. *Transactions on Information Systems*, 9(3):187–222, 1991.