Fast Query Execution for Retrieval Models based on Path Constrained Random Walks

Ni Lao
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
nlao@cs.cmu.edu

William Cohen
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
wcohen@cs.cmu.edu

ABSTRACT

Many recommendation and retrieval tasks can be represented as proximity queries on a labeled directed graph, with typed nodes representing documents, terms, and metadata, and labeled edges representing the relationships between them. Recent work has shown that the accuracy of the widely-used random-walk-based proximity measures can be improved by supervised learning - in particular, one especially effective learning technique is based on Path Constrained Random Walks (PCRW), in which similarity is defined by a learned combination of constrained random walkers, each constrained to follow only a particular sequence of edge labels away from the query nodes. The PCRW based method significantly outperformed unsupervised random walk based queries, and models with learned edge weights. Unfortunately, PCRW query systems are expensive to evaluate. In this study we evaluate the use of approximations to the computation of the PCRW distributions, including fingerprint, particle filtering, and truncation strategies. In experiments on several recommendation and retrieval problems using two large scientific publications corpora we show speedups of factors of 2 to 100 with little loss in accuracy.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms

Algorithms,Experimentation

Keywords

learning to rank, Filtering and Recommending, genomics IR

1. INTRODUCTION

In many natural IR settings, documents are associated with metadata (e.g., author names and citations in corpora of technical papers or patents). A corpus of such documents can be viewed as a labeled directed graph, with typed nodes representing documents, terms, and metadata, and labeled edges representing the relationships between them (e.g., “authorOf”, “datePublished”, etc). This data representation suggests the problem of *relational retrieval*, in which a user’s query is a weighted set of graph nodes, and a response is a ranked set of answer nodes ordered by some measure of proximity to the query nodes. Traditional keyword-based ranked retrieval of documents is a special case of this, as is collaborative filtering.

One very common and successful set of proximity measures are based on random walks on graphs, for instance lazy random walks [19] or personalized PageRank [6, 8] or Random Walk with Restart [24]. However, regular graph-walk based similarity measure is naive in the sense that the random walker does not distinguish the importance of different paths. Relational data is usually annotated with a rich set of entity and relation types, and there could be many different relation paths by which the query entities can reach the target entity. Naturally these paths have different importance to the retrieval task. For example, in the TREC-CHEM Prior Art Search Task, instead of directly searching for patent with the query words, people found it much more effective to first find papers with similar topic, then aggregate these papers citations [17]. The path associated with this strategy can be expressed as “$\textit{query word} \xrightarrow{ContainedBy} \textit{patent} \xrightarrow{Cite} \textit{patent}$”.

In our previous work [15], we developed an effective learning technique that is based on Path Constrained Random Walks (PCRW), in which similarity is defined by a learned combination of constrained random walkers, each constrained to follow only a particular sequence of edge labels away from the query nodes. Each of these paths can be seen as a little expert of identifying relevant entities, and we combine their decisions by a supervised log-linear model. The PCRW based mode can discover (and appropriately weight) fairly complex relation paths, like the one above.

Unfortunately, PCRW query systems are expensive to evaluate. Number of nodes with non-zero probability grows very fast on a well connected graph. However, one impotant fact about random walks is that, in general, we expect that the random walk will lead to very uneven distributions over all the entities: high probability on a few entities, usually entities of high in-degrees, and low probability on the remainder. For example, this kind of uneven distribution (called power law distribution) has been observed [22] on PageRank scores of web pages, which is also a type of random walk model. As a consequence, a few nodes have a large influence on the retrieval result, and most nodes have very small influence, and it is plausibly acceptable to ignore, or approximate, the weight of most nodes. In past work, a sampling based random walk strategy has been shown to give inaccurate estimations for low ranked nodes; in spite of this, however, Fogaras et al. [11] showed that using a Monte Carlo algorithm and a small number of trials is sufficient to distinguish between the high, medium and low ranked nodes accurately in Personalized PageRank scores; as well, Chakrabarti [7] used a dynamic pruning strategy to the calculation of Personalized Pageranks, in which weights smaller than a threshold are pruned, and showed that this operation has minimal effect on accuracy.

Therefore, we can expect that keeping the distribution of random walk to be sparse may significantly reduce the amount of time and memory spent on query execution. In this study, we investigate the trade-off between inference efficiency and retrieval quality for supervised learning of PCRW models. We compare the query execution speedups by different strategies that help maintain sparsity of random walk, including finger prints, particle filtering, and truncation strategies. Our experiment on several recommendation and retrieval tasks involving scientific publications show that appropriate sparsity strategies can improve retrieval efficiency by up to two orders of magnitude without noticeably effecting retrieval quality.

The rest of this paper is organized as follows. Section 2 briefly reviews related works. Section 3 defines the relational retrieval task, and the path ranking algorithm. Section 4 describes four strategies to maintain sparsity of random walks. Section 5 analyzes experiment result on several tasks in biogenetics domain. Section 6 concludes the work.

2. RELATED WORKS

Many early approaches to the problem of retrieval on entity-relation data graph are keyword-based database systems [2][14][6]. They are designed mainly for ad-hoc queries, thus not trainable for specific IR tasks. Another branch of work is using random walk on graphs as proximity measures, notably the PageRank [21] and the Personalized PageRank algorithm [12]. There have been much follow up work in supervised learning of random walk models. Nie et al.[20] use simulated annealing to perform local search over each edge type, which is only applicable when the number of parameters is very small. Diligenti et al. [10] optimize relation weights using back-propagation, which has linear convergence, therefore requires many iterations to reach convergence. Furthermore, the use of regression loss function is not appropriate, since retrieval is actually a classification problem. Notably, Agarwal et al. [1] applied efficient second order optimization procedure with preference labels of pairs of entities. However, instead of using real relevance data, document orderings generated from artificially manipulated random walk models are used. In this study we use (orthant-wise L-BFGS [3]), and binomial log-likelihood loss functions. We situate our algorithm in several realistic retrieval and recommendation tasks each having thousands of training and testing queries.

One limitation to these models is that by using one parameter per edge type, these models cannot leverage complex path features of relational data. Recently, there has been systems that learn to do graph retrieval with richer feature sets. Minkov et al. [19] showed that using n-grams of relation paths as reranking features can significantly improve the retrieval quality of a random walk based model. More recently, Minkov & Cohen[18] proposed a random walk based generative learning model that favors paths which are more likely to reach relevant entities.

In our previous study, we developed a discriminative version of this learning technique that is based on Path Constrained Random Walks (PCRW), in which similarity is defined by a learned combination of constrained random walkers [15]. This PCRW based model has shown significant improvement of retrieval quality over the edge parameterized random walk models. We will describe this model in detail in section 3.

The efficiency of keyword search on graph has been the concern for many previous systems. Most of them [23][13][9] build two-level representations of the graphs offline, which can speed up query execution. Tong et al.[24] studied fast RWR methods based on low-rank matrix approximation, and graph partitioning. More recently, Chakrabarti [7] developed the HubRank algorithm, which precompute offline the Personalized Pagerank Vectors (PPVs) for a small fraction of nodes, carefully chosen using query log statistics. It is not immediately apparent how to adapt these methods to path-constrained random-walk distance measures; hence, in this study, we will focus on methods for maintaining a sparse representation of random walk distributions at query time. .

3. THE LEARNING METHOD

For completeness, this section summarizes the PCRW learning algorithm, which we call the path ranking algorithm.

3.1 The Relational Retrieval Problem

Before describing our model, let's define the relational retrieval problem. An *schema* $S = (\mathbf{T}, \mathbf{R})$ consists of a set of *entities types* $\mathbf{T} = \{T\}$, and a set of relations $\mathbf{R} = \{R\}$. An *entity-relation graph* $G = (\mathbf{E}, S)$ is an instantiation of the schema S to a set of *entities* $\mathbf{E} = \{e\}$. Each entity has type $e.T \in \mathbf{T}$. The *instantiation* of a type T is $I(T) = \{e|e.T = T\}$. Each relation R has a pair of entity types $R.T_1$, $R.T_2$. We use $R(e_1, e_2) = 1/0$ to denote $e_1 \in T_1$, $e_2 \in T_1$ having relation R or not. We use $R(e, \cdot) = \{e'|R(e, e') = 1\}$ to denote the set of entities that have relation R with e. As examples, Figures 2 and 3 show schemas for two entity-relation graphs.

We define a general task called *Relational Retrieval*: given a set of query entities $\mathbf{E}_q = \{e'\}$, predict the relevance of each entity e of the target entity type T_q . We call $q = (\mathbf{E}_q, T_q)$ a *query*. In other words, the task is to model $p(Rel(e)|q)$, $e \in I(T_q)$, where $Rel(e)$ is the relevance of e.

3.2 The Retrieval Model

This subsection describes how to automatically generate features of the target entities with the concept of relation trees. Let's start with the concept of relation paths. Consider the simpler case where $\mathbf{E}_q$ only contains entities of a single type

T_0. Given a ER graph $G = (\mathbf{E}, S = (\mathbf{T}, \mathbf{R}))$ and a query $q = (\mathbf{E}_q, T_q)$. We define a *relation path* $P = (R_1, ..., R_n)$ be a sequence of relations, with Constrained that $R_1.T_1 = T_0$, and $R_i.T_2 = R_{i+1}.T_1$.

For each relation path we define a series of probability distributions $h_i(e)$, $e \in I(T_i)$ on the sets of entities on the path

$$h_0(e) = \begin{cases} 1/|\mathbf{E}_q|, & e \in E_q \\ 0, & otherwise \end{cases} \quad (1)$$

$$h_i(e) = \sum_{e' \in I(T_{i-1})} h_{i-1}(e') \frac{R_i(e', e)}{|R_i(e', \cdot)|}, \quad (2)$$

where T_i is a short hand for $R_i.T_2$.These distributions correspond to a path constrained random walk. Note that these probability distributions might be deficient ($\sum_e h_i(e) < 1$). Since some entities might not have any out link of the required relation type, their probability mass do not pass on.

Let $\mathbf{P}(q, l)$ be the set of relation paths that start with T, end with T_q, and have length l. The distributions defined by all the relation paths can be summarized as a prefix tree (Figure 1), with each node corresponds to a distribution $h_P(e)$ over the entities. The PCWR model (G, l, θ) ranks

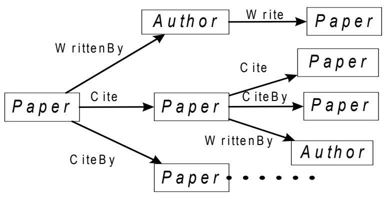


Figure 1: A 2-level relation tree for a simple schema of paper and author

$e \in I(T_q)$ by the scoring function

$$s(e; \theta) = \sum_{P \in \mathbf{P}(q,l), P.T_{last} = T_q} h_P^T(e) \theta_P, \quad (3)$$

or in matrix form $s = A\theta$, where s is a sparse column vector of scores, and θ is a column vector of weights for the relation paths. We call A the *feature matrix*, of which each column is the distribution $h_P(e)$ associated with type path P. We refer the i-th row of A as A_i.

We also note that some paths give exactly the same distribution over the target entities. For example the following three paths are equivalent (where $Before^{-1}$ is the inverse of relation $Before$).

$year \xrightarrow{Before^{-1}} year \xrightarrow{Before} year \xrightarrow{Before} year$
$year \xrightarrow{Before} year \xrightarrow{Before^{-1}} year \xrightarrow{Before} year$
$year \xrightarrow{Before} year \xrightarrow{Before} year \xrightarrow{Before^{-1}} year$
To avoid creating many of these uninteresting paths, we add constraints to some types of relations that they cannot be immediately followed by their inverse.

Generally, the query entity set $\mathbf{E}_q$ may contain multiple entities of multiple types $T^{(1)}, ..., T^{(n)}$. Here we simply assume that their effects can be linearly combined. We create a relation tree for each query entity type, and then linearly combine all their features. The score to the target entities can be written as $s = A\theta$, where $A = [A(T^{(1)})...A(T^{(n)})]$ is a concatenation of distribution matrices of all relation trees, and $\theta = [\theta(T^{(1)}); ...; \theta(T^{(n)})]$ is a concatenation of weight vectors of all relation trees.

3.3 Parameter Estimation

Given a set of training data $D = \{(q^{(m)}, y^{(m)})\}$ where $m = 1...M$, $y^{(m)}$ is a binary vector, each of its elements $y_e^{(m)} = 1/0$ indicates whether entity e is relevant to the query $q^{(m)}$, parameter estimation can be formulated as maximizing a regularized objective function

$$O(\theta) = \sum_{m=1..M} o_m(\theta) - \lambda_2 ||\theta||_2^2 / 2$$

where λ_2 controls $l2$-regularization to prevent over fitting, which is especially important when there are strongly correlated features. The per-instance objective function $o^{(m)}(\theta)$ can be in various forms. Like the binomial log-likelihood (of logistic regression), negative hinge loss (of SVM), negative exponential loss (of boosting), and etc. Here we use binomial log-likelihood, which is easy to optimize and does not penalize too harshly outlier samples, as exponential loss does.

For a training instance $(q^{(m)}, y^{(m)})$, let $A^{(m)}$ be the feature matrix, $P^{(m)}$ be the index set of the relevant entities, and $N^{(m)}$ the index set of the irrelevant entities. We use the average log-likelihood of positive and negative entities as the objective

$$o_m(\theta) = \sum_{i \in P_m} \frac{\ln p_i^{(m)}}{|P_m|} + \sum_{i \in N_m} \frac{\ln(1 - p_i^{(m)})}{|N_m|}$$

where $p_i^{(m)} = p(Y_i^{(m)} = 1; \theta) = \sigma(\theta^T A_i^{(m)})$, and $\sigma(x) = \exp(x)/(1 + \exp(x))$ is the "logistic" function. It's gradient is

$$\frac{\partial o_m(\theta)}{\partial \theta} = \sum_{i \in P_m} \frac{(1 - p_i^{(m)}) A_i^{(m)}}{|P_m|} - \sum_{i \in N_m} \frac{p_i^{(m)} A_i^{(m)}}{|N_m|}.$$

We use efficient second order optimization procedure (orthant-wise L-BFGS [3]), to estimate the parameters.

3.4 Negative Document Selection Strategy

In most retrieval tasks, there are just a few positive entities but thousands (or millions) of negative ones. Reflecting all of them in the objective function would lead to very expensive training. Aslam et al. [5] compared the effect of different document selection methods for effective and efficient learning to rank. However, most of the tested methods assume the existence of multiple retrieval systems and are thus not applicable to our setting.

Here we take a simple method which we call *quadratic sampling*. Generally, non-relevant entities highly ranked by some weak (but decent) ranking function are more important than lower ranked ones. Therefore, we first sort all the negative entities with a PCWR based model without training (all path weights are set to 1.0). Then entities at the $k(k + 1)/2$-th positions in the retrieved result are selected as negative samples, where $k = 0, 1, 2, 3, ...$ For example, if we have 100k target entities in a corpus, then around $\sqrt{2 * 100k} \approx 500$ negative samples would be selected.

In preliminary experiments, we compared quadratic sampling with cubic sampling $(k(k+1)(k+2)/6)$, exponential sampling (2^k), and top-k sampling (taking top-k negative entities), but none of them gives better result. Actually, exponential and top-k sampling produce very poor results. This confirms experience with the LETOR [16] judgment collection, which uses BM25 as weak ranker and uses top-k pooling. If not enough negative samples from low score range are sampled, the learnt model becomes negatively correlated to the weak ranker, and demotes overall performance.

4. MAINTAINING DISTRIBUTION SPARSITY OF RANDOM WALKS

In this section, we describe four strategies to maintain sparsity of random walks. All of them approximate the exact estimation of random walk distribution defined by equation (1). They use different ways of generating or sparsifying $h_{i+1}(e)$ at each step of random walk.

4.1 The Fingerprint Strategy

Fogaras et al. [11] suggest a Monte Carlo algorithm to compute approximate values of personalized PageRank, where for each node u, K independent random walks are simulate starting from u. The Personalized PageRank vector for u is approximated by the normalized counts of number of times the random walkers visit a node. The authors show that using only a relatively small number of random walker is sufficient to distinguish between the high, medium and low ranked nodes in the fully computed Personalized PageRank scores. Although the order of the low ranked nodes are usually not as accurate using sampling, it is most often the high ranked nodes that determine the quality of retrieval.

In this study, we test the effectiveness of this sampling strategy in the context of path constrained random walk models. The distribution $h_{i+1}(e)$ can be approximated by the normalized count of number of times the walkers visit a node e after one step walks starting from their position of the previous step i.

$$h_{i+1}(e) = \frac{\text{\#times the walkers visit } e}{\text{\#walkers}},$$

We can easily control the amount of computation by changing K.

4.2 Weighted Particle Filtering

One possible downside for the fingerprint strategy is the waste of computation when the number of walkers is larger than the number of possible paths. For example, if we start 30k walkers from a node, which only has three out links, the fingerprint strategy will draw 10k random numbers, and assign each of the walker to follow a specific out link. However, with knowledge of probabilities, we know that it is expected to have around 10k walkers following each of the out links. Conceptually we can treat the initial 30k walkers as a single particle, and after the first step of random walk it splits into three equal-sized particles, each containing 10k walkers and follows a different out link. If we let the particles keep splitting to arbitrarily small sizes, then we just get the exact probability distribution defined by equation (1) (with proper normalization). Therefore, in order to keep the distribution sparse and speedup random walk, we can set a threshold to the minimum size of the particles. If a potential split can break a particle to sizes smaller than the threshold, we just let the particle follow one of the out links of current node, without splitting the particle. We call this procedure *weighted particle filtering*, and it is shown in Algorithm 1.

Algorithm 1 Weighted particle filtering

```
Input: distribution $h_i(e)$, relation $R$, minimum particle size $\varepsilon_{min}$
Output: $h_{i+1}(e)$
Set $h_{i+1}(e) = 0$ (should not take any time)
for each $e$ with $h_i(e) \neq 0$ do
  $size = h_i(e)/|R(e,\cdot)|$
  if $size_{new} > \varepsilon_{min}$ then
    for each $e' \in R(e,\cdot)$ do
      $h_{i+1}(e') += size_{new}$
    end for
  else
    $h_{i+1}(e') += h_i(e)$ , where $e'$ is randomly selected from $R(e,\cdot)$
  end if
end for
```

4.3 Truncation Strategies

In general, we expect that the random walk have uneven distribution over all the entities: high probability on a few entities of high in-degrees, and low probability on many noisy entities of low in-degrees. For example, this kind of uneven distribution (called power law distribution) has been observed [22] on PageRank scores of web pages, which is also a type of random walk model. Therefore, we hypothesis that removing the probability on those less important entities will not significantly affect the random walk's ability to identify important entities, but on the other hand may significantly reduce the amount of time and memory spent on random walk. Recently, Chakrabarti [7] apply a dynamic pruning strategy to the calculation of Personalized Pageranks, for which elements in the fingerprint vectors smaller than a threshold are pruned. They discover that this operation has dramatic effect on keeping the fingerprint vectors sparse, while minimal effect on accuracy.

In this study, we test the effectiveness of dropping nodes with small probabilities in the context of path constrained random walk models. At each step of random walk, we add a further truncation step to the probability distribution estimated by equation (1) as the following

$$h_{i+1}(e) = max(0, h_{i+1}(e) - \varepsilon),$$

where ε is a parameter to control harshness of the truncation. This truncation procedure also has the effect of putting more regularization on longer paths. Since path constrained random walks of longer paths are generally reduced more harshly by this truncation procedure, their weights need to be larger than the short paths in order to achieve the same effect in the ranking function. We call this approach *fixed truncation.*

One possible disadvantage of fixed truncation is that the truncation parameter ε is not directly related to the sparsity of probability distributions. Therefore, we design an adaptive truncation strategy called *beam truncation*, which explicitly constrains the random walk to the desired sparseness. At each step of random walk, we add a further truncation step to the probability distribution estimated by equation (1) as the following

$$h_{i+1}(e) = max(0, h_{i+1}(e) - \varepsilon_W),$$

where ε_W is the W-th highest probability among all the entities, and W is called the width of the beam.

5. EXPERIMENT

Biologists currently spend a lot of time in searching for relevant information about specific bioentities (e.g. a set of genes). In this section, we compare different sparsity strategies with exact calculation of random walk distributions by their retrieval qualities on several recommendation and retrieval tasks involving scientific publications.

5.1 Data Sets

We use two publication data sets (Yeast, and Fly) in the biogenetic domain. These publications are tagged with biological entities. We generate the corpora from the following resources. *PubMed* (www.ncbi.nlm.nih.gov/pubmed) is a free on-line archive of over 18 million biological abstracts for papers published since 1948. *PubMed Central (PMC, www.ncbi.nlm.nih.gov/pmc)* contains full-text copies of over one million of these papers. *Saccharomyces Genome Database* (SGD, www.yeastgenome.org) is a database of various types of information concerning the yeast organism Saccharomyces cerevisiae, including about 50K papers manually tagged with the genes they mention. *Gene Ontology (GO, www.geneontology.org)* is a large ontology describing the *properties* of and *relationships* between various biological entities across numerous organisms. *Flymine* (www.flymine.org) is an integrated database for Drosophila and Anopheles genomics, which contains about 130K papers tagged with genes and proteins.

Figure 2 shows the schema of the yeast corpus. Each paper

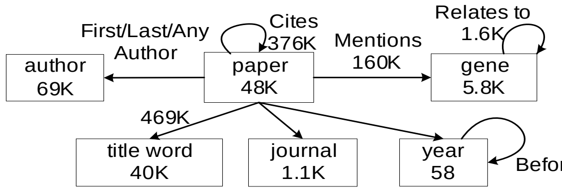


Figure 2: Schema of the yeast data.

is annotated with the genes it mentions. The title words are filtered by a stop word list of size 429. The *RelatesTo* relations from genes to other genes are extracted from their GO description. The *Authorship* relations are further distinguish into three sub-types: any author, first author, and last author.

Figure 3 shows the schema of the fly corpus. The schema is similar to that of the yeast data, except a new entity type

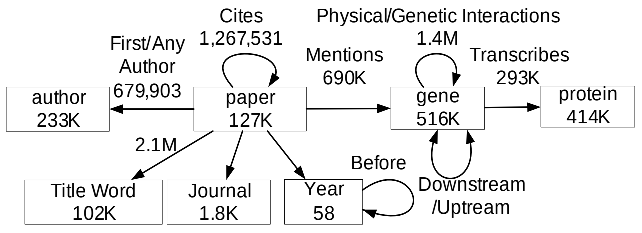


Figure 3: Schema of the fly data.

Table 1: Corpus statistics

	paper	node	edge	train	dev	test
Yeast	48K	164K	2.8M	2K	2K	2K
Fly	127K	770K	3.5M	2K	2K	2K

Protein, and several relations among genes. *Downstream* and *Upstream* relations connect a gene to its closest neighbors on the DNA strand.

Table 1shows statistics of the two corpora. For both corpora, we randomly hold out 2000 queries for development, and another 2000 queries for testing. We evaluate models by Mean Average Precision (MAP). All experiments are averaged over 10 runs. We randomly pick 2000 queries (not overlapping with the development and testing queries) 10 times as the training sets.

5.2 The "Paper Completion" Tasks

We simulate several retrieval and recommendation tasks for which relevance judgements can be automatically generated from publication data. Given a paper, we treat entities of some of its fields (e.g. title) as query, and entities of one other fields as true relevant entities. Depending on the types of query entities $\mathbf{E}_q$ and target entities T_q, different application scenarios can be fit into this framework. We design these tasks to be actually helpful for biologists in their daily work:

The **Reference Recommendation** task help literature review of specific topic, or help author to cite previous works properly. The query is based on genes, title words, proteins (for fly data) or year (for yeast data).

The **Expert Finding** task helps finding expert of a specific topic. The query is based on genes, title words, proteins (for fly data) or year (for yeast data).

The **Gene Recommendation** task suggests to researchers genes that they might be interested in. The query is based on who the author is and the year of query. This task is useful to intelligently guide biologists' search [4].

Because the fly data is larger than the yeast data, and has an extra entity type protein, we do not use the publication year as part of the query for fly data. To simulate the actual retrieval scenario, we need to prevent the system from using information obtained later than the query's date. Therefore, we define a *time variant graph* in which each edge is tagged with a time stamp (year). When doing random walk for a query generated from a particular paper, we only consider edges that are earlier than the publication date of that paper. For generalized PageRank, one set of features is calculated offline at each year.

5.3 Parameter Tuning on Development Data

Figure 4 shows the effect of L2 regularization and different path length l for reference recommendation task on yeast data. We can see that a small amount of L2 regularization can slightly improve MAP, and longer path lengths give better performances, but only to a certain level. For the rest of the experiments, we pick for each task λ_2 and l that produces relative good performance on the development data.

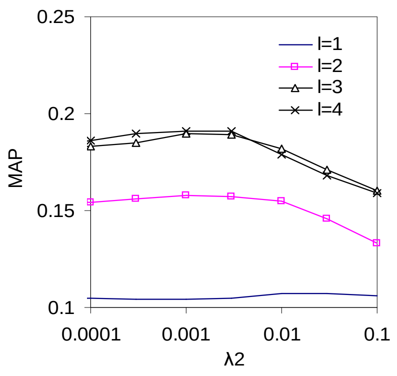


Figure 4: Compare different regularization parameter λ_2 and different path length l for reference recommendation task on yeast data.

5.4 Main Results

In order to investigate the trade-off between the speedup of query execution and retrieval qualities, we varies for each sparsity method its sparsity parameter– from most inefficient to most efficient setting, and see how the retrieval speed and quality are affected. We provide three baselines as references to the quality of retrieval: exact calculation of random walk distributions, exact random walk with unsupervised model (weights of all paths are uniformly set of 1.0), and exact random walk with edge parameter learning [1, 19]. For the estimation of the edge parameters, we use the same binomial log-likelihood objective function and LBFGS optimization procedure as the PCRW model.

Figure 5 and 6 compare speedup of query execution verses MAP and MRR for different methods. From left to right we order the tasks in increasing order of random walk complexity. The average single query execution time ranges from 0.15 seconds to 5.6 seconds when using exact calculation of random walk distributions. We can see that all four strategies manage to speedup query execution to certain extend without significantly sacrifice the quality of retrieval. The sparsity strategies are relative more effective on complex task like gene recommendation and reference recommendation.

The two truncation strategies (fixed and beam truncation) have relatively limited ability to speedup query execution (ranging from 2 fold to 10 fold). This is because, at each step of random walk, they need to do the full distribution before truncation, therefore limit their ability to speedup random walk. Fixed truncation generally has slightly better retrieval quality than beam truncation. After close inspection we found that although fixed and beam truncation can produce random walk distribution with the same sparsity when their parameters are properly set, fixed truncation has an extra effect of demoting longer paths. Since random walks of longer paths are reduced more harshly by this truncation procedure, their weights need to be larger than the short paths in order to achieve the same effect in the ranking function. The beam truncation on the other hand might not truncation the distribution at all if the number of non-zero nodes is smaller than the beam size, which is very likely to happen in the first few steps of random walk. This is evident in figure 7, where we compare sparsity and learned weights for fixed and beam truncation model. Parameters are set so as both strategy gives 2.7 fold speedup over the exact random walk baseline. However, fixed truncation has statistically significantly better performance (MRR=0.407, MAP=0.205) than beam truncation (MRR=0.398, MAP=0.200). We can see that although the sparsity of these two strategies correlates with each other very well, the learnt weights of beam truncations is generally much smaller than that of fixed truncation. Since the number of long relation path is much larger than the number of short relation paths, beam search is more pruned to overfitting.

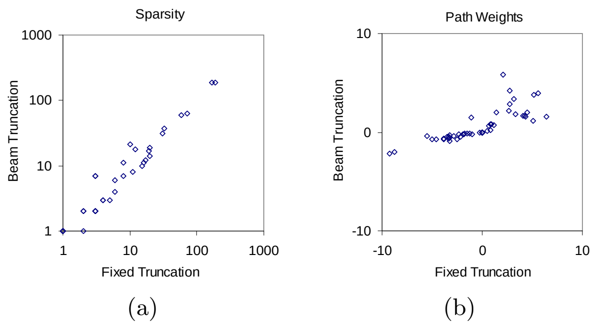


Figure 7: Compare sparsity & model weights for fixed and beam truncation. Experiment conducted for the reference recommendation task on Yeast data. Each point represents a relation path. (a) shows the number of nodes with non-zero probability for each path averaged over all queries. (b) shows the learned weights for each path

The two sampling based strategies (fingerprint and particle filtering) have relatively larger speedup (ranging from 10 fold to 100 fold on various tasks). Compared to particle filtering, fingerprint has slightly worse retrieval quality at low speedup range, but significantly better quality at high speedup range.

More interestingly, in several tasks (expert finding and gene recommendation), moderately applying the sparsity strategies not only speed up the query execution but produces better retrieval quality. Table 2 shows statistics of average number of relevant entities per query for different tasks and corpora. We can see that, these two tasks have relatively less

Table 2: Average number of relevant entities per query for different tasks and corpora

	Yeast	Fly
Expert Finding	4.1	5.9
Gene Recommendation	4.1	19.7
Reference Recommendation	23.2	35.4

number of relevant entities than the reference recommendation task. In these cases, ignoring low importance node (by truncation or sampling) and focusing on high ranked nodes are likely to improve retrieval quality.

6. CONCLUSION

In this study we evaluate the use of approximations to the computation of the path constrained random-walk distributions, including fingerprint, particle filtering, and truncation strategies. In experiments on several recommendation and retrieval problems using two large scientific publications corpora we show speedups of factors of 2 to 100 with little loss in accuracy. More interestingly, we found that in several tasks moderately applying the sparsity strategies not only speed up the query execution but produces better retrieval quality.

7. REFERENCES

[1] A. Agarwal, S. Chakrabarti, and S. Aggarwal. Learning to rank networked entities. pages 14–23, 2006.

[2] S. Agrawal, S. Chaudhuri, and G. Das. Dbxplorer: A system for keyword-based search over relational databases. *ICDE*, pages 5–16, 2002.

[3] G. Andrew and J. Gao. Scalable training of l^1-regularized log-linear models. *ICML*, pages 33–40, 2007.

[4] A. Arnold and W. W. Cohen. Information extraction as link prediction: Using curated citation networks to improve gene detection. *WASA*, pages 541–550, 2009.

[5] J. A. Aslam, E. Kanoulas, V. Pavlu, S. Savev, and E. Yilmaz. Document selection methodologies for efficient and effective learning-to-rank. *SIGIR*, pages 468–475, 2009.

[6] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. *ICDE*, pages 431–440, 2002.

[7] S. Chakrabarti. Dynamic personalized pagerank in entity-relation graphs. pages 571–580, 2007.

[8] S. Chakrabarti and A. Agarwal. Learning parameters in entity relationship graphs from ranking preferences. *PKDD*, pages 91–102, 2006.

[9] B. B. Dalvi, M. Kshirsagar, and S. Sudarshan. Keyword search on external memory data graphs. *PVLDB*, 1(1):1189–1204, 2008.

[10] M. Diligenti, M. Gori, and M. Maggini. Learning web page scores by error back-propagation. In *IJCAI*, pages 684–689, 2005.

[11] D. Fogaras and B. Rácz. Towards fully personalizing PageRank. In *Proceedings of the 3^{rd} Workshop on Algorithms and Models for the Web-Graph (WAW2004), in conjunction with FOCS 2004.*, 2004.

[12] T. H. Haveliwala. Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search. *IEEE Trans. Knowl. Data Eng.*, 15(4):784–796, 2003.

[13] H. He, H. Wang, J. Y. 0001, and P. S. Yu. Blinks: ranked keyword searches on graphs. *SIGMOD*, pages 305–316, 2007.

[14] V. Hristidis and Y. Papakonstantinou. Discover: Keyword search in relational databases. *VLDB*, pages 670–681, 2002.

[15] N. Lao and W. Cohen. Relational retrieval using an combination of path-constrained random walks. *in submission.*

[16] T. Y. Liu, J. Xu, T. Qin, W. Xiong, and H. Li. Letor: Benchmark dataset for research on learning to rank for information retrieval. *SIGIR*, 2007.

[17] M. Lupu, F. Piroi, X. Huang, J. Zhu, and J. Tait. Overview of the trec 2009 chemical ir track. *TREC-18*, 2009.

[18] E. Minkov and W. W. Cohen. Learning graph walk based similarity measures for parsed text. *EMNLP*, pages 907–916, 2008.

[19] E. Minkov, W. W. Cohen, and A. Y. Ng. Contextual search and name disambiguation in email using graphs. *SIGIR*, pages 27–34, 2006.

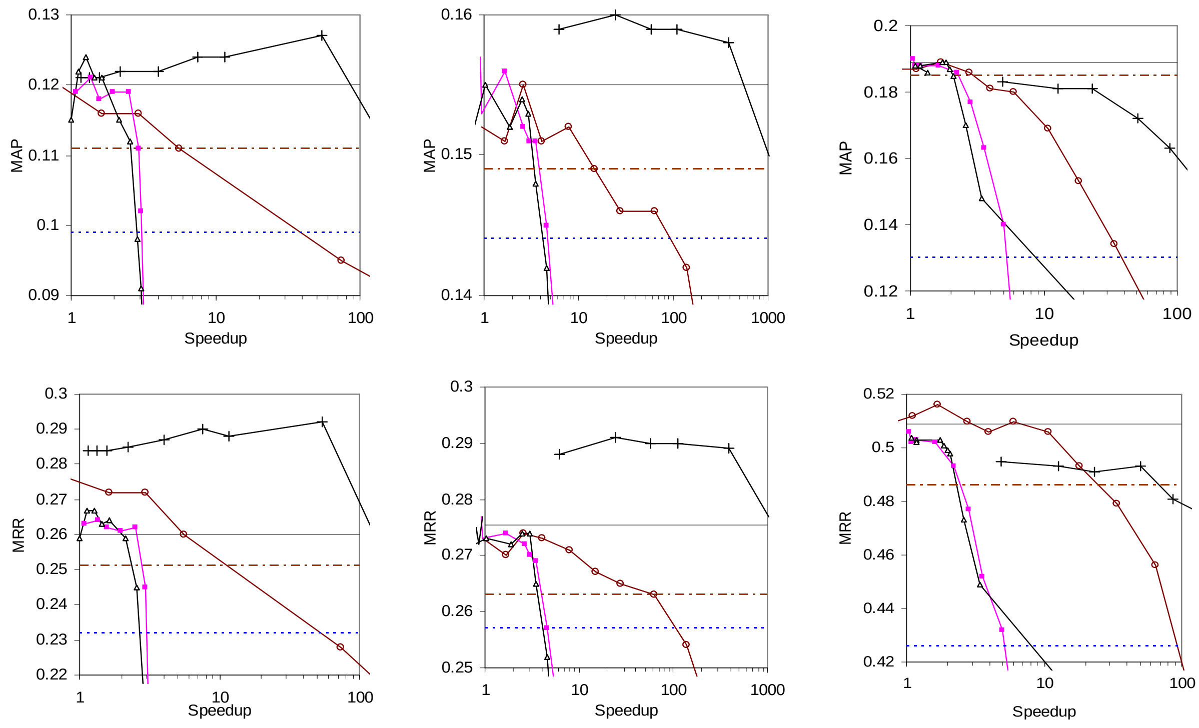
[20] Z. Nie, Y. Zhang, J.-R. Wen, and W.-Y. Ma. Object-level ranking: bringing order to web objects. *WWW*, pages 567–574, 2005.

[21] L. Page, S. Brin, R. Motwani, and T. Winograd.

[22] G. Pandurangan, P. Raghavan, and E. Upfal. Using pagerank to characterize web structure. *COCOON*, pages 330–339, 2002.

[23] S. Raghavan and H. Garcia-Molina. Representing web graphs. pages 405–416, 2003.

[24] H. Tong, C. Faloutsos, and J.-Y. Pan. Fast random walk with restart and its applications. *ICDM*, pages 613–622, 2006.

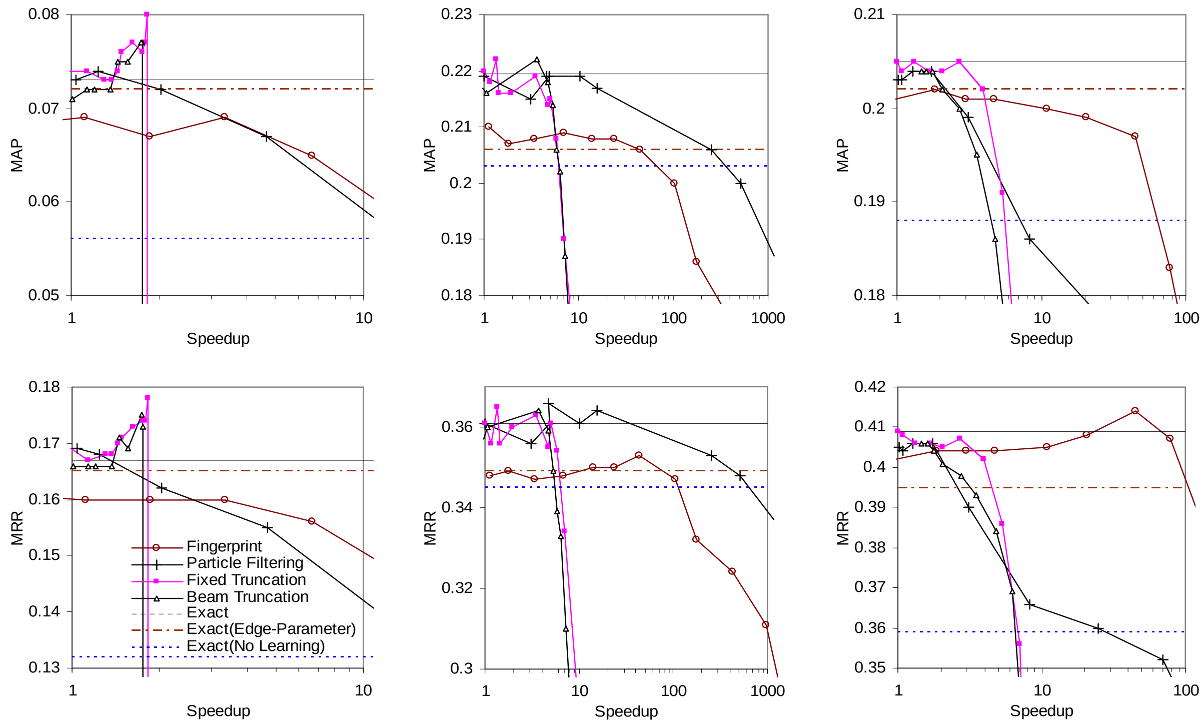


Expert Finding
$T_0 = 0.17s, l = 3$

Gene Recommendation
$T_0 = 1.6s, l = 4$

Reference Recommendation
$T_0 = 2.3s, l = 3$

Figure 5: Speedup verses retrieval performance for different strategies on the yeast data. T_0 is the average query execution time using exact random walk. l is the maximum length of random walk paths



Expert Finding
$T_0 = 0.15s, l = 3$

Gene Recommendation
$T_0 = 1.7, l = 4$

Reference Recommendation
$T_0 = 5.6s, l = 3$

Figure 6: Speedup verses retrieval performance for different strategies on the fly data.