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 W. 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 fingerprinting, 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

Relational Retrieval, Path-Constrained Random Walks, Learning to Rank, Filtering and Recommending

KDD'10, July 25–28, 2010, Washington, DC, USA.

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 [25]. 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 "*query word* $\xrightarrow{ContainedBy}$ *patent* $\xrightarrow{Cite}$ *patent*".

In our previous work [16], 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 for 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. The number of nodes with non-zero probability grows very fast on a well connected graph. However, one important 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; PageRank 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 for 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 reached by random walks 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 fingerprinting, particle filtering, and truncation strategies. Our experiments 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.

In the remainder of the paper, we first briefly reviews related work. We next present the path ranking algorithm, describing first the way in which path experts are enumerated, then the learning algorithm. In the next section, we describe four strategies to maintain the sparsity of random walks. We then analyze experimental result on several tasks in biogenetics domain and conclude.

2. RELATED WORK

Many early approaches to the problem of retrieval on entity-relation data graph are keyword-based database systems [2][15][6]. They are designed mainly for ad-hoc queries, thus are 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. 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 were used. One limitation to all 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 work on 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, in which similarity is defined by a learned combination of constrained random walkers [16]. 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 [24][13][9] build two-level representations of the graphs offline. Tong et al.[25] 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 PCRW-BASED MODEL

For completeness, this section summarizes the PCRW learning algorithm, which we call Path Ranking Algorithm (PRA).

3.1 The Retrieval Model

One-parameter-per-edge label RWR proximity measures are limited because the *context* in which an edge label appears is ignored. For example, in the reference recommendation task, one of the query nodes is a year. There are two ways in which one might use a year y to find candidate papers to cite: (H1) find papers published in year y, or (H2) find papers frequently cited by papers published in year y. Intuitively, the second heuristic seems more plausible than the first; however, a system that insists on a using a single parameter for the "importance" of the edge label *PublishedIn* cannot easily encode this intuition.

To define heuristics of this sort more precisely, let R be a typed binary relation. We write $R(e, e')$ if e and e' are related by R, and define $R(e) \equiv \{e' : R(e, e')\}$. We will use $dom(R)$ to denote the domain type of R, and let $range(R)$ for its range. A *relation path* P is a sequence of relations $R_1 \dots R_\ell$. We say that a relation path is *type-correct* if the domains and ranges of adjacent relations are compatible– i.e., if $\forall i : 1 < i < \ell - 1, range(R_i) = dom(R_{i+1})$. In this paper we will consider only type-correct relation paths. For a type-correct path, we define $dom(R_1 \dots R_\ell) \equiv dom(R_1)$ and $range(R_1 \dots R_\ell) \equiv range(R_\ell)$, and when we wish to emphasize the types associated with each step in a path, we will write the path $P = R_1 \dots R_\ell$ as

$$T_0 \xrightarrow{R_1} \dots \xrightarrow{R_\ell} T_\ell$$

where $T_0 = dom(R_1) = dom(P), T_1 = range(R_1) = dom(R_2)$ and so on. In this notation,the two heuristics suggested above would be written as:

$$H1: \quad year \xrightarrow{PublishedIn^{-1}} paper$$

$$H2: \quad year \xrightarrow{PublishedIn^{-1}} paper \xrightarrow{Cite} paper$$

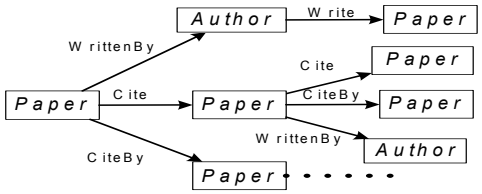


Figure 1: A 2-level relation tree for a simple schema of paper and author

This notation makes it clear that the range of each relation path is *paper*, the desired type for reference recommendation. We use $^{-1}$ to denote the inverse of a relation, which is considered a different relation: for instance, $PublishedIn$ and $PublishedIn^{-1}$ are considered as different relations.

For any relation path $P = R_1 \dots R_\ell$ and set of query entities $E_q \subset dom(P)$, we define a probability distribution $h_{E_q,P}$ as follows. If P is the empty path, then define

$$h_{E_q,P}(e) = \begin{cases} 1/|E_q|, & \text{if } e \in E_q \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

If $P = R_1 \dots R_\ell$ is nonempty, then let $P' = R_1 \dots R_{\ell-1}$, and define

$$h_{E_q,P}(e) = \sum_{e' \in range(P')} h_{E_q,P'}(e') \cdot \frac{I(R_\ell(e',e))}{|R_\ell(e')|}, \quad (2)$$

where $I(R(e',e))$ is an indicator function that takes value 1 if $R(e',e)$ and 0 otherwise. If we assume as well that $I(R(e',e)) = 0$ when e' is not in the domain of R, then the definition natural extends to the case where E_q is not a subset of $dom(P)$.

Given these definitions, the intuition that "heuristic H1 is less useful than H2" could be formalized as follows: for reference recommendation queries E_q, T_q, where E_q is a set of title words, gene-protein entities, and a year y, entities e_1 with high weight in $h_{E_q,PublishedIn^{-1}}$ are not likely to be good citations, where as entities e_2 with high weight in $h_{E_q,PublishedIn^{-1}.Cite}$ are likely to be good citations. More generally, given a set of paths $P_1, \dots, P_n$, one could treat these paths as features for a linear model and rank answers e to the query E_q by

$$\theta_1 h_{E_q,P_1}(e) + \theta_2 h_{E_q,P_2}(e) + \dots \theta_n h_{E_q,P_n}(e)$$

where the θ_i are appropriate weights for the paths.

In this paper, we consider learning such linear weighting schemes over all relation paths of bounded length ℓ. For small ℓ (e.g., $\ell \leq 4$), one can easily generate $\mathcal{P}(q,l) = \{P\}$, the set of all type-correct relation paths with range T_q and length $\leq l$. The distributions defined by all the relation paths can be summarized as a prefix tree (Figure 1), where each node corresponds to a distribution $h_P(e)$ over the entities. A PRA model ranks $e \in I(T_q)$ by the scoring function

$$s(e;\theta) = \sum_{P \in \mathcal{P}(q,l)} h_{E_q,P}(e)\theta_P, \quad (3)$$

In matrix form this could be written $s = A\theta$, where s is a sparse column vector of scores, and θ is a column vector of weights for the corresponding paths P. We will call A the *feature matrix*, and denote the i-th row of A as A_i.

We found that, because some of the relations reflect one-to-one mappings, there are groups of paths that give exactly the same distribution over the target entities. For example, the following three paths among years are actually equivalent:

$year \xrightarrow{Before^{-1}} year \xrightarrow{Before} year \xrightarrow{Before} year$

$year \xrightarrow{Before} year \xrightarrow{Before^{-1}} year \xrightarrow{Before} year$

$year \xrightarrow{Before} year \xrightarrow{Before} year \xrightarrow{Before^{-1}} year$

To avoid creating these uninteresting paths, we add constraint to the following relations that they cannot be immediately preceded by their inverse: $Before$, $Before^{-1}$, $PublishedBy(journal)$, $PublishedIn(year)$.

3.2 Parameter Estimation

There have been much previous work in supervised learning of random walk models. Nie et al.[20] use exhaustive local search over each edge type, which is only applicable when the number of parameters is very small. Diligenti et al. [10] and its follow up [19] optimize weights on the relations using back-propagation, which has linear convergence, therefore requires many iterations to reach convergence. Recent work [1][8] uses more efficient second order optimization procedure like BLMVM for numerical optimization. In this study, we use L-BFGS [3], a second order optimization procedure used for many machine learning problems, and binomial log-likelihood loss functions.

The training data can be represented as $\mathcal{D} = \{(q^{(m)}, r^{(m)})\}$, $m = 1...M$, where $r^{(m)}$ is a binary vector. $r_e^{(m)} = 1$ if entity e is relevant to the query $q^{(m)}$, and $r_e^{(m)} = 0$ otherwise. Given the training data, parameter estimation can be formulated as maximizing a regularized objective function

$$O(\theta) = \sum_{m=1..M} o^{(m)}(\theta) - \lambda|\theta|_2/2 \quad (4)$$

where λ is a regularizer, and $o^{(m)}(\theta)$ is a per-instance objective function. In this paper we use binomial log-likelihood (the loss function for logistic regression); however, negative hinge loss (for SVM), negative exponential loss (for boosting), and many other functions could be used instead. Binomial log-likelihood has the advantage of being easy to optimize, and also does not penalize outlier samples too harshly, as exponential loss does. For a training instance $(q^{(m)}, r^{(m)})$, let $A^{(m)}$ be its corresponding feature matrix, $\mathcal{R}^{(m)}$ be the index set of the relevant entities, and $\mathcal{N}^{(m)}$ the index set of the irrelevant entities. In order to balance the uneven number of positive and negative entities, we use the average log-likelihood of positive and negative entities as the objective

$$o^{(m)}(\theta) = \sum_{i \in \mathcal{R}^{(m)}} \frac{\ln p_i^{(m)}}{|\mathcal{R}^{(m)}|} + \sum_{i \in \mathcal{N}^{(m)}} \frac{\ln(1-p_i^{(m)})}{|\mathcal{N}^{(m)}|} \quad (5)$$

where $p_i^{(m)} = p(r_i^{(m)} = 1;\theta) = \sigma(\theta^T A_i^{(m)})$, σ is the sigmoid function $\sigma(x) = \exp(x)/(1+\exp(x))$, and the gradient is

$$\frac{\partial o^{(m)}(\theta)}{\partial \theta} = \sum_{i \in \mathcal{R}^{(m)}} \frac{(1-p_i^{(m)})A_i^{(m)}}{|\mathcal{R}^{(m)}|} - \sum_{i \in \mathcal{N}^{(m)}} \frac{p_i^{(m)} A_i^{(m)}}{|\mathcal{N}^{(m)}|}. \quad (6)$$

In most retrieval tasks, there are just a few positive entities but thousands (or millions) of negative ones. Therefore using all of them in the objective function is expensive. Here we used a simple strategy similar to stratified random sampling [23]. First, we sort all the negative entities using a PRA model without training (i.e., all feature weights are

set to 1.0). Then, entities at the $k(k+1)/2$-th positions are selected as negative samples, where $k = 0, 1, 2, 3, ...$ This is useful because generally, non-relevant entities which are highly ranked by the untrained ranking function are more important than lower ranked ones: for in-depth comparisons of different selection strategies we refer the reader to Aslam et al.'s work [5].

For parameter estimation of the one-weight-per-edge-label RWR model, we use the same log-likelihood objective function and LBFGS optimization procedure as for PRA. Since RWR can be seem as the combination of all the PCRWs with each path having its weight set to the product of all the edge weights along the path, we can calculate the gradient of edge weights by first calculating the gradient w.r.t. the paths, and then applying the chain rule of derivative.

4. SPARSE RANDOM WALKS

In this section, we describe four strategies to maintain the sparsity of random walks. They all approximate the exact random walk distribution defined by equation (2), but with different ways of generating or sparsifying $h(e)$ at each step of random walk. Each of these strategies will be used in both parameter estimation at training time and query execution at test time.

4.1 The Fingerprinting Strategy

Fogaras et al. [11] suggested a Monte Carlo algorithm to approximate the distributions of personalized PageRank, where K independent random walks are simulated starting from the query node. The probability of a node u is approximated by the normalized count of number of times it is visited by the random walkers, and the amount of computation can be easily controlled by varying K. The authors showed that using only a relatively small number of random walkers is sufficient to distinguish between the high, medium and low ranked nodes in the fully computed Personalized PageRank scores. Although the orders 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 PCRW. The distribution $h_{i+1}(e)$ at the $i+1$-th step can be approximated by the normalized count of the number of walkers visiting a node e after a one step walk starting from their positions in the previous step i

$$h_{i+1}(e) = \frac{\#\text{walkers visiting } e}{\#\text{walkers}}.$$

4.2 Weighted Particle Filtering

One possible downside for the fingerprinting strategy is the waste of computation when the number of walkers is much larger than the number of links. For example, if we start with 30k walkers from a node which only has three outlinks, the fingerprinting strategy will draw 30k random numbers, and assign each of the walkers to follow a specific outlink. However, with knowledge of probabilities, we know that it is expected to have around 10k walkers following each of the outlinks.

Here we describe a *weighted particle filtering* procedure (Algorithm 1), which is an combination of exact random walk and sampling. Conceptually, we can treat the initial 30k walkers as a single particle, and at the first step of random walk it splits into three equal-sized particles, each containing 10k walkers and following a different link. If we let the particles split to arbitrarily small sizes, then we just get the exact probability distribution defined by equation (2) (with proper normalization). In order to keep the distribution sparse and speedup random walk, we set a threshold ε_{min} on the minimum size of the particles. If a potential split breaks a particle to particles with sizes smaller than the threshold, we switch from exact calculation to sampling strategy. We let the particle splits to less number of child particles, each having the same size as the threshold ε_{min}. Each of these child particles randomly picks one of the outlinks to follow.

Algorithm 1 Weighted Particle Filtering

Input: distribution $h_i(e)$, relation R, threshold ε_{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**
  $size_{new} = h_i(e)/|R(e)|$
  if $size_{new} > \varepsilon_{min}$ **then**
    for each $e' \in R(e)$ **do**
      $h_{i+1}(e') += size_{new}$
    end for
  else
    for k=1..$floor(h_i(e)/\varepsilon_{min})$ **do**
      randomly pick $e' \in R(e)$
      $h_{i+1}(e') += \varepsilon_{min}$
    end for
  end if
end for

4.3 Truncation Strategies

As we explained before, random walks usually have uneven distribution over all the entities: high probability on a few important entities, and low probability on many noisy entities. Therefore, we hypothesis that zeroing the weights on the low-weight 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] applied 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 a dramatic effect on keeping the fingerprint vectors sparse, while having a minimal effect on accuracy.

Here we test the effectiveness of this strategy in the context of PCRW. At each step of the random walk, we add a truncation step to the distribution estimated by equation (2):

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

where ε is a parameter to control the harshness of truncation. This procedure also has the effect of putting more regularization on longer paths. Since 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 sparsities of probability distributions. Therefore, we design an adaptive truncation strategy called *beam truncation*, which explicitly constrains the random walk to the desired sparseness. The truncation step is defined as

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

where $\varepsilon_W(h_{i+1})$ is the W-th highest probability in distribution h_{i+1}, and W is called the width of the beam.

5. EXPERIMENT

In this section, we compare different sparsity strategies by their retrieval qualities on several recommendation and retrieval tasks involving scientific publications. The baseline for these strategies is the exact calculation of random walk distributions.

5.1 Tasks

We consider here three tasks that are well-suited to solution by typed proximity queries.

Reference (or citation) recommendation is the problem of finding relevant citations for a new paper. The query is, as in venue recommendation, the title terms and relevant entities for the new paper, the current year, and the answer type is "paper". The desired answer is a list of papers ranked by appropriateness as citations in the new paper. This task is similar to the TREC-CHEM Prior Art Search Task [17], and can also be seem as a simplified version of the context-aware citation recommendation task [14].

Expert finding is the problem of finding a domain expert for a particular topic. The query is again a list of terms and relevant entities, the current year, and the answer type is "person". The desired answer is a list of people with expertise on this topic.

Gene recommendation, considered by Arnold and Cohen [4], is the problem of predicting, given past publishing history, which genes an author will publish about over the next year. Here the query nodes are an author and a year, and the answer type is "gene". This task is an approximation to predicting future interests.

The first task is encountered in preparing a new paper, and the second in finding reviewers, or new collaborators. To evaluate performance on these tasks, we will compare the ranked list from a query associated with a paper to the actual metadata associated with the paper: specifically, we will compare the actual citations to the recommended citations. Perhaps more speculatively, we will also compare the authors of a paper to the experts recommended by the query based on the title and related-entity set for the paper. In each case the predictions will be made using a graph that does not contain the actual paper in question—see the next subsection for details.

5.2 Datasets

We created two publication data sets (Yeast and Fly) in the biological domain. Paper content and metadata information are crawled from two resources: *PubMed*[1] is a free on-line archive of over 18 million biological abstracts for papers published since 1948; *PubMed Central* (PMC)[2] contains full-text and references to over one million of these papers.

[1]www.ncbi.nlm.nih.gov/pubmed

[2]www.ncbi.nlm.nih.gov/pmc

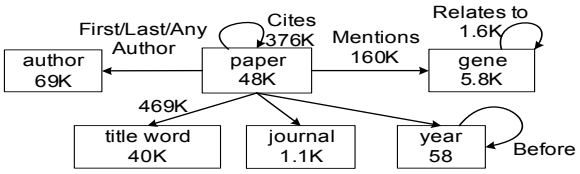


Figure 2: Schema of the yeast data.

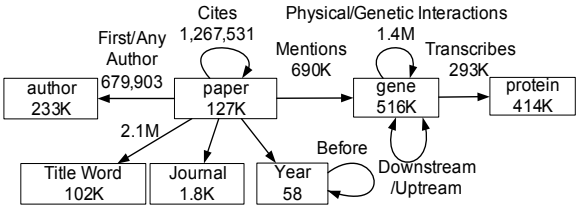


Figure 3: Schema of the fly data.

Figure 2 shows the schema of the yeast corpus. We extracted gene mentions from the *Saccharomyces Genome Database(SGD)*[3], which is a database of various types of information concerning the yeast organism Saccharomyces cerevisiae, including about 48K papers, each annotated with the genes it mentions. The title words are filtered by a stop word list of size 429. The *Authorship* relations are further distinguish into three sub-types: any author, first author, and last author. We extracted gene-gene relations from *Gene Ontology* (GO)[4], which is a large ontology describing the properties of and relationships between various biological entities across numerous organisms.

Figure 3 shows the schema of the fly corpus. It is extracted from *Flymine*[5], which is an integrated database for Drosophila and Anopheles genomics, and contains about 127K papers tagged with genes and proteins. The schema is similar to that of the yeast data, except for a new entity type *Protein*[6], and several relations among genes. The *Downstream* and *Upstream* relations connect a gene to its two neighbors on the DNA strand.

Each paper can be used to simulate a query and relevance judgements for any of the three above mentioned tasks. However, 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). During random walk for a query generated from a particular paper, we only consider edges that are earlier than that paper's publication date. As shown by Table 1, for each task on any of the two corpora, we randomly hold out 2000 queries for development, and another 2000 queries for testing. We evaluate models by Mean Average Precision (MAP) and Mean Reciprocal Rank (MRR).

5.3 Main Results

In order to investigate the trade-off between speedup of

[3]www.yeastgenome.org

[4]www.geneontology.org

[5]www.flymine.org

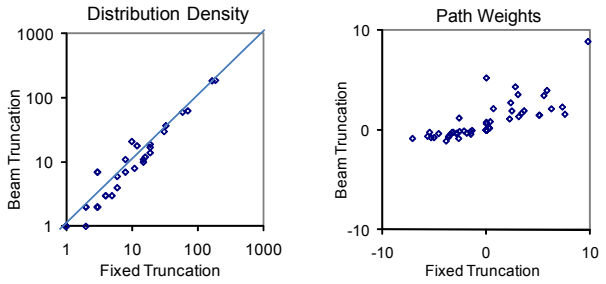
[6]In yeast, there is a nearly one-to-one relationship between genes and proteins, as most genes are transcribed to a unique protein; in flies, alternative splicing means that a gene can be transcribed to several different proteins.

Table 1: Corpus statistics

	Graph Size			No. Query		
	paper	**node**	**edge**	**train**	**dev**	**test**
Yeast	48K	164K	2.8M	2K	2K	2K
Fly	127K	770K	3.5M	2K	2K	2K

query execution and retrieval quality, we vary for each sparsity method its sparsity parameter— from the most inefficient to the 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 but unsupervised RWR (uniform weight 1.0), exact RWR, and exact PCRW. For supervised models we fix the regularization parameter λ to 0.001.

Figure 4 compares the speedup of query execution verses MAP for different sparsity strategies. From left to right we order the tasks in increasing order of random walk complexity. The average single query execution times range from 0.15 seconds to 2.7 seconds when using exact calculation of PCRW. We can see that all four strategies manage to speedup query execution to certain extend without significantly sacrifice the quality of retrieval. The strategies are relatively more effective on complex tasks like gene recommendation and reference recommendation.



(a) (b)

Figure 5: Compare sparsity and model weights of fixed and beam truncation for the reference recommendation task on Yeast data. (a) the number of nodes with non-zero probability for each relation path (averaged over all training queries). (b) the learned weight for each relation path

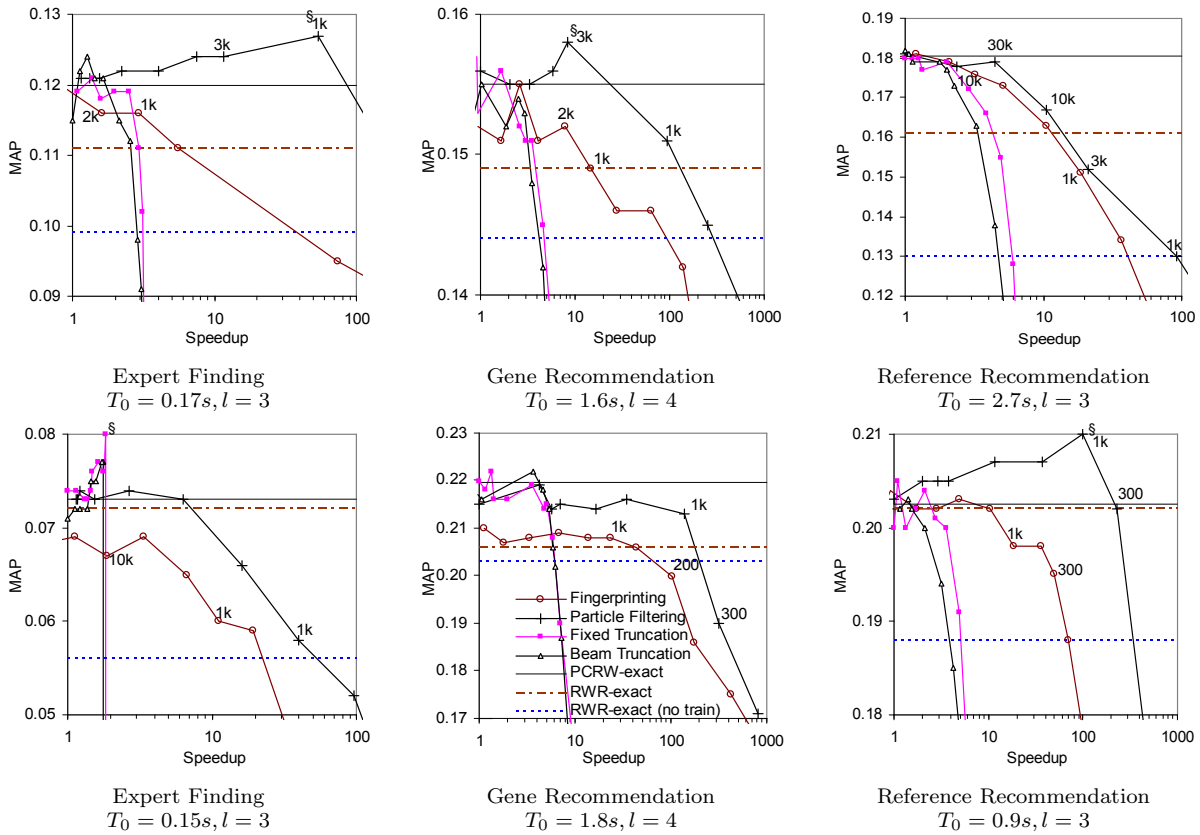
The two truncation strategies (fixed and beam truncation) have relatively limited ability to speedup query execution (ranging from 2- to 10-fold). This is because, at each step of random walk, they need to calculate the full distribution before truncation. When the truncation is harsh, on the other hand, it is likely to produce empty distributions, which are useless for retrieval. 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 truncate 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 5, where we compare sparsity and learned weights for fixed and beam truncation. Parameters are set so as both strategies give 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 sparsities of these two strategies correlates with each other very well, the learnt weights of fixed truncations are generally much smaller than that of beam truncation. This is because that the absolute values of the probabilities produced by fixed truncation are smaller than that of beam truncation. The path weights of fixed truncation need to be larger than those of beam truncation in order to achieve the same effect to ranking. Therefore, fixed truncation has the effect of putting heavier regularization on the weights of longer paths.

The two sampling based strategies (fingerprinting and particle filtering) have relatively larger speedups (ranging from 10- to 100-fold on various tasks). For plot in Figure 4, we mark the number of random walkers and the maximum number of particles (estimated by $1/\varepsilon_{min}$) at the points where retrieval quality starts to drop fast. We can see that, compared to fingerprinting, particle filtering is almost always 2- to 4-fold faster. This is what we have expected, since particle filtering can potentially represent a large number of walkers with a small number of particles during the first few steps of random walk. Overall, we can see that around 1k to 10k particles (or random walkers) are enough for producing good retrieval quality.

Furthermore, particle filtering almost always produces better retrieval quality than fingerprinting. This is a result of the fact that although both strategies rely on sampling to estimate the exact distribution, particle filtering will have lower variance than fingerprinting. For example, imagine a node with only two outlinks, and let the two strategies each have two particles (minimum particle size 0.5) and two walkers respectively. Fingerprinting has a high chance (0.5) to put all probability mass to only one of the outlinks. Particle filtering, on the other hand, will always assign equal probability 0.5 to both of the links.

More interestingly, in many cases applying the sparsity strategies not only speeds up the query execution but also produces better retrieval quality (in four out of six tasks). It is not immediately evident whether this is because that the sparsity strategies produce distributions that concentrated on important nodes or because that they produce models with better weighting of the relation paths. In Figure 6a we fix retrieval model to either the one trained with exact PCRW or the one with particle filtering. Then we apply particle filtering during test time and vary ε_{min}. We can see that when the model is fixed, particle filtering does not improve retrieval quality. However, the model trained with particle filtering still performs better than the one trained with exact PCRW. This result indicates that training with particle filtering PCRW can produce better models than exact PCRW.

Figure 6b shows the density of each relation path (i.e., the number of nodes with non-zero probability). We can see that for those paths with dense distribution (density $>$ 1000) under exact PCRW, particle filtering effectively reduces densities to a few hundreds. Figure 6c further shows that, in the model trained with exact PCRW, most of the high density paths have positive weights, and their weights are significantly dropped in the corresponding model trained



Expert Finding $T_0 = 0.17s, l = 3$ | Gene Recommendation $T_0 = 1.6s, l = 4$ | Reference Recommendation $T_0 = 2.7s, l = 3$

Expert Finding $T_0 = 0.15s, l = 3$ | Gene Recommendation $T_0 = 1.8s, l = 4$ | Reference Recommendation $T_0 = 0.9s, l = 3$

Figure 4: Speedup *vs.* MAP for different strategies on the yeast (upper row) and fly (lower row) data. T_0 is the average query execution time using exact PCRW. l is the maximum length of random walk paths. So points are marked with the number of random walkers or the maximum number of particles (estimated by $1/\varepsilon_{min}$). The points marked with § have improvements over exact PCRW that are statistically significant at p<0.00001 using paired s-test.

with particle filtering. Therefore, we can see that the sparsity strategies have the effect of putting higher regularization on the paths with dense distributions, which turns out to be useful in producing good retrieval models.

6. CONCLUSION

In this study, we evaluate the use of approximations to the computation of the path-constrained random-walk distributions. We compared fingerprinting, particle filtering, and truncation strategies. In experiments on several recommendation and retrieval problems using two large scientific publication corpora, we show speedups of factors of 2 to 100 with little loss in retrieval accuracy. More interestingly, we found that in many cases moderately applying the sparsity strategies not only speeds up the query execution but also 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

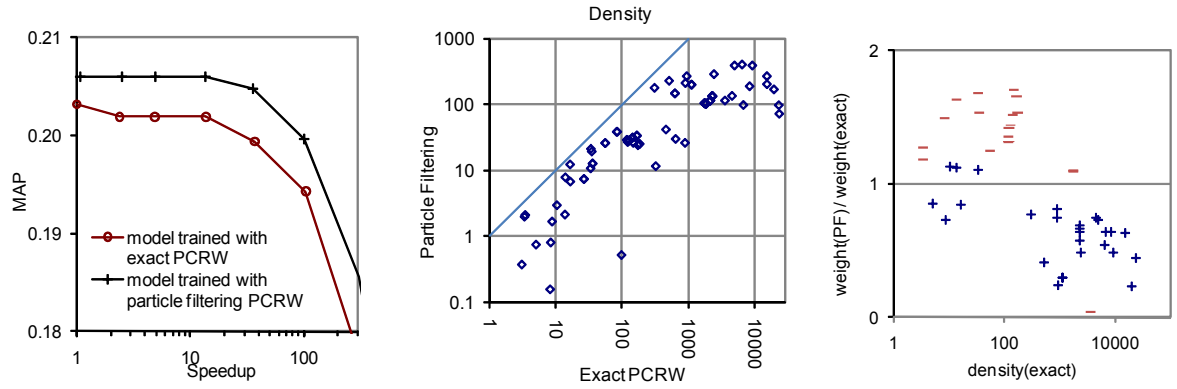


Figure 6: Compare the models trained with exact PCRW and particle filtering ($\varepsilon_{min} = 0.001$) for the reference recommendation task on Yeast data. (a) compares retrieval qualities with fixed model but various particle filtering settings at query time. The differences of the two models at the same speed up levels are statistically significant at p<0.00001 using paired s-test. (b) shows for each relation path the number of nodes with non-zero probability (averaged over all training queries). (c) shows for each relation path the ratio of its weight in the particle filtering model *vs.* its weight in the exact PCRW model. The '+' and '−' signs correspond to paths with positive and negative weights in the model trained with exact PCRW.

PageRank. In *Proceedings of the 3rd 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] Q. He, J. Pei, D. Kifer, P. Mitra, and C. L. Giles. Context-aware citation recommendation. 2010.

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

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

[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. The pagerank citation ranking: Bringing order to the web. *Technical report, Stanford University Database Group.*

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

[23] V. Pavlu. Large scale ir evaluation. In *PhD thesis, Northeastern University, College of Computer and Information Science*, 2008.

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

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