Learning to Rank Results in Relational Keyword Search

Joel Coffman
University of Virginia
Charlottesville, VA
jcoffman@cs.virginia.edu

Alfred C. Weaver
University of Virginia
Charlottesville, VA
weaver@cs.virginia.edu

ABSTRACT

Keyword search within databases has become a hot topic within the research community as databases store increasing amounts of information. Users require an effective method to retrieve information from these databases without learning complex query languages (viz. SQL). Despite the recent research interest, performance and search effectiveness have not received equal attention, and scoring functions in particular have become increasingly complex while providing only modest benefits with regards to the quality of search results. An analysis of the factors appearing in existing scoring functions suggests that some factors previously deemed critical to search effectiveness are at best loosely correlated with relevance. We consider a number of these different scoring factors and use machine learning to create a new scoring function that provides significantly better results than existing approaches. We simplify our scoring function by systematically removing the factors with the lowest weight and show that this version still outperforms the previous state-of-the-art in this area.

リレーショナル・データに拡張された
キーワード検索が使用されない原因

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; H.2.4 [**Database Management**]: Systems—*Relational databases*

General Terms

Design, Experimentation, Performance

Keywords

relational keyword search, ranking

1. INTRODUCTION

The success of web search engines has made keyword search the preferred method for individuals to discover and to retrieve information [12] as evidenced by the 4 billion keyword searches performed daily [14]. Given the ubiquity of the search text box, it is natural to extend the keyword search paradigm to support other sources of information, including semi-structured data (e.g., XML) and relational data.[1] Many data-driven websites store information in relational databases; periodically crawling their content inevitably results in outdated information appearing in search results. While websites commonly provide a means to search their content, such facilities often are inferior to Internet search engines or constrain searches by supporting only the simplest queries (e.g., find articles whose titles are like . . .). Hence, a niche exists for systems that search databases like Internet search engines search the web.

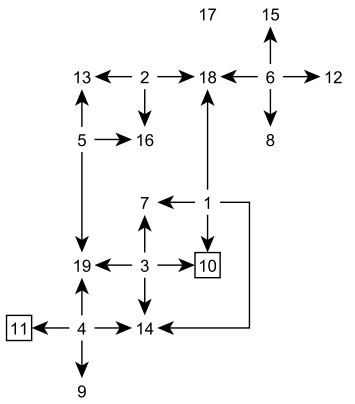
A decade of academic research has led to a number of systems that extend the keyword search paradigm to relational data, but we are not aware of any of these systems that are actively used outside the academic community. We posit that this transition has not occurred due to a host of challenges that plague research in this field. First, the evaluation of many systems remains ad hoc, and results do not seem to generalize to other datasets and query workloads. Second, many systems propose algorithms to improve performance while also adopting new functions for scoring results, and existing evaluations tend to focus on the former while ignoring the latter. Third, extant literature has not dealt with the numerous issues that surround different ranking functions. For example, although one ranking function might be shown empirically to provide higher quality search results, studies have not investigated which factors in the scoring function are responsible for the improvement.

In this paper, we focus on this last issue by examining existing ranking methods and by investigating the importance of different factors in these scoring functions. Our work is timely because many researchers to date have relied on intuition to create new scoring functions. For example, many systems assume that the relevance of a search result is negatively correlated with the total edge weight of a result. However, the literature contains little evidence for this relationship, and in this paper, we show that total edge weight is actually positively correlated with relevance. We couple our analysis of existing scoring functions with machine learning to create a new ranking scheme.

Machine learning is preferable to deriving a new scoring function by hand due to machine learning's ability to consider many potential factors and to provide confidence measures

[1]In this paper, we focus on keyword search within relational databases, and due to length, we omit the numerous techniques for searching semi-structured data.

CIKM'11, October 24–28, 2011, Glasgow, Scotland, UK.



Postings	
actor	→ 14
ark	→ 18
composer	→ 17
connery	→ 11
crusade	→ 19
director	→ 16
ford	→ 10
harrison	→ 10
indiana	→ 7, 19
jones	→ 7, 9, 19
last	→ 19
raiders	→ 18
sean	→ 11
...	

Figure 1: Example of an IMDb data graph (left) and a portion of its inverted index (right).

for its results. Moreover, scoring functions typically include a number of parameters that have previously been tuned by researchers using a small number of test queries. It is reasonable to believe that significant improvement is possible simply by applying better tuning methodologies (e.g., via machine learning). In our evaluation using a diverse set of datasets and queries, we show that our scoring function, which includes many factors identified in existing work, significantly improves upon the retrieval effectiveness of state-of-the-art systems.

1.1 Example of Relational Keyword Search

As an example of relational keyword search, consider the data graph and the portion of its associated inverted index shown in Figure 1. Furthermore, assume that the user wants to identify the relationship between "Harrison Ford" and "Sean Connery" (e.g., to determine if they have appeared together in any films). To start, the inverted index must be scanned to identify nodes (tuples) that contain the search terms. In our example, nodes 10 and 11 both match disjoint subsets of the search terms. Hence, it is not appropriate to return each individual node as a search result because each node by itself does not address the relationship between the search terms. Search heuristics are typically used to identify these relationships. In this case, nodes 10 and 11 are related by the path 10–3–**14**–4–11 and by the path 10–3–**19**–4–11. These two relationships can be described loosely as "both are actors" (10–3–**14**–4–11) and "both appeared in the film *Indiana Jones and the Last Crusade*" (10–3–**19**–4–11).

The aforementioned steps cover the *enumeration* of search results. Enumeration is distinct from *ranking*, where the list of results is ordered by each result's estimated relevance to the user's information need. In our example, the latter result (both appeared in the film *Indiana Jones and the Last Crusade*) should appear before the former (both are actors). The existing literature includes a host of different ranking strategies, but little analysis of why particular strategies perform better than others.

1.2 Contributions and Outline

In this paper, we use machine learning to investigate the importance of different factors when ranking search results. The major contributions of this work are as follows.

- We review existing scoring functions and show that they have become increasingly complex over the past decade. More recent scoring functions introduce additional features that increase the number of free parameters, but previous work does not address issues related to tuning these parameters.
- We use ordinal regression to learn a linear function for scoring search results. Our scoring function significantly outperforms existing systems at high recall levels.
- We simplify our scoring function by reducing the total number of factors that are included and show that many factors can be eliminated without significantly sacrificing search quality.
- We compare our simplified scoring function against more conventional alternatives. Even though the alternatives are computationally more expensive, they do not improve search quality.

The remainder of this paper is organized as follows. In Section 2, we present background material—existing scoring functions and how to use machine learning to weight the features of a scoring function. Section 3 describes the creation of our scoring function. We evaluate our scoring function against an established benchmark for relational keyword search and show that it outperforms existing systems. Section 4 analyzes our scoring function, compares it to alternatives, and discusses the limitations of our approach. In Section 5, we review related work. Finally, we conclude and describe future work in Section 6.

2. BACKGROUND

This section starts by examining existing functions for scoring search results. Our objective is to underscore the variety of schemes that have been proposed and to illustrate the complexity of existing scoring functions. We also present machine learning as a technique to derive appropriate weights for the features in a scoring function.

2.1 Existing Scoring Functions

Tables 1 and 2 summarize the features advocated by systems that propose a novel scoring scheme. Many systems are omitted from these tables because they address performance

Table 1: Summary of scoring functions used by proximity search systems. Systems that do not propose an alternative scoring function are omitted (i.e., each row contains at least one unique definition •).

System	Features: $\lvert N\rvert$	Features: $weight(e)$	Features: $weight(n)$	Factors: $\sum_e weight(e)$	Factors: $\sum_n weight(n)$	Parameters
BANKS [1]	○	•	•	•	•	1
DISCOVER [19]	○					0
BANKS-II [22]		•	•	•	•	1
DPBF [10]		○		•		0
BLINKS [17]		•	○	•	•	3
EASE [31]		○		•		0
Golenberg *et al.* [16]		•		○		2

Legend

•	unique definition	$n \in N$	nodes in result
○	uses existing definition	e	edge in result

Table 2: Summary of IR-style scoring functions used for relational keyword search. The factors *ntf*, *ndl*, and *idf* are components of pivoted normalization weighting; ⊕ indicates the combination of attribute scores.

System	Features			Factors						Parameters
	$\lvert N\rvert$	$\overline{\lvert N\rvert}$	$\lvert N'\rvert$	*nsize*	L^2 *norm*	*ntf*	*ndl*	*idf*	⊕	
DISCOVER-II [18]	○			○		○	○	○	•	1
Liu *et al.* [32]	○	○		•		○	•	•	•	2
SPARK [33]	○		○	•	•	○	•	•	○	4
EASE [31]						•	○	•	○	1

Legend

- • unique definition
- ○ uses existing definition
- ⊕ attribute combination
- N nodes in result
- $\overline{\lvert N\rvert}$ average result size
- N' non-leaf nodes

issues (i.e., enumeration) rather than ranking. Space prevents us from presenting the actual scoring functions and a detailed discussion of the differences among the systems.[2] Instead, distinctions are indicated in the tables where a unique definition (•) differs from previous work instead of using an existing technique (○) to score results. Each row of the table contains at least one unique element. For example, although Golenberg *et al.* [16] reuse DPBF's scoring function [10] (see Table 1), they propose an alternative method to assign edge weights.

The number of "features" (that is, atomic values from a search result, user's query, or database) has increased steadily as the various scoring functions distinguish between a search result's root, leaves, interior nodes, etc. Accompanying this increase is a rise in the number of "factors" (that is, values derived from one or more features). As the number of factors increases so does the total number of free parameters, which are used to weight the importance of each factor in the final scoring function. This is problematic because tuning these parameters is expensive, requiring exploration of a low-dimensional grid of possible parameter values or heuristics that partially explore many dimensions [45]. Unfortunately, the existing literature does not suggest that these experiments are being conducted by researchers.

2.2 Machine Learning

Machine learning offers an alternative to creating scoring functions by hand and also may be used to determine the relative weights of different factors. In this case, the goal of machine learning is to determine the weight of each feature that maximizes the relevance of results. Such work is described as *ordinal regression*, which predicts the ranking of search results. Ordinal regression lies between classification—that is, predicting a categorical variable (e.g., relevant or non-relevant)—and regression—that is, predicting a real number as a function of inputs.

Given a document collection $C = \{d_1, d_2, \ldots, d_n\}$ and query Q, the objective of an IR system is to provide a ranking σ that optimally orders the documents according to their relevance to Q. A support vector machine (SVM) is a type of large-margin classifier, which attempts to maximize the decision boundary between classes. We use a *ranking SVM* in this work. The goal of a ranking (or ordinal regression) SVM is to learn a function f such that for any d_i, d_j

$$f(\phi(q, d_i)) > f(\phi(q, d_j)) \Rightarrow \sigma(d_i) > \sigma(d_j)$$

where $\phi(q, d)$ is the feature vector of the query and a document d and $\sigma(d)$ is the position of d in the optimal ranking. Let P be the set of pairs (i, j) where $\sigma(d_i) > \sigma(d_j)$. Formally, a ranking SVM optimizes

$$\min_{\vec{w}, \xi_{ij} \geq 0} \frac{1}{2}\vec{w}^T\vec{w} + \frac{C}{P}\sum_{(i,j)\in\mathcal{P}} \xi_{ij}$$

subject to

$$\forall (i, j) \in P, \vec{w}^T f(\phi(q, d_i)) \geq \vec{w}^T f(\phi(q, d_j)) + 1 - \xi_{ij}$$

where $\vec{w}$ is the decision hyperplane that separates different classes (ranks) and ξ is a slack variable that allows for training errors. The function f identifies the ideal weights for each feature by minimizing the number of pairs that are swapped (with regard to the optimal ranking) when training the SVM.

3. MODEL

This section describes the creation and evaluation of our model for scoring relational keyword search results.

3.1 Features

The features included in any scoring function should be efficient to compute and highly correlated with the ideal ranking of the search results [16]. From our review of existing scoring functions, we identified 10 different features that are used in 8 different factors. Some features included in previous work cannot be computed efficiently. For example, calculating the average number of nodes in the set of results ($\overline{\lvert N\rvert}$) requires enumerating all results for each query. Including this feature in a scoring function immediately precludes top-k query processing schemes.

We derive 12 different factors from the features present in existing scoring functions. Our additional factors revolve around simple ways to combine multi-valued features (e.g., averaging the edge weights). Table 3 lists all our factors. Note that some of our factors subsume the features shown in Tables 1 and 2. For example, pivoted normalization weighting [42] includes *ntf*, *ndl*, and *idf* factors. Rather than considering these factors in isolation, we only use their combination in pivoted normalization weighting, which is a state-of-the-art scoring function [41].

Our factors are generally a superset of those included in previous work, but we do not explore all the variations of these factors that appear in the literature because doing so would greatly expand the search space. For example, there are nearly 100 different formulations of pivoted normalization weighting alone if all the proposed factors are considered independently. In addition, several factors have been created to improve the tractability of enumerating search results (e.g., distinct root semantics [17, 22]), and these factors are typically compared to more traditional methods for scoring results (in this case, minimizing the total edge weight), which we do include.

Given the variety of existing techniques to calculate the value of each factor, we chose one instead of exploring all the alternatives. We use BLINKS's formula for deriving the

[2]For additional coverage of this material, we refer the reader to an extended version of this paper [8].

Table 3: Correlations between factors and the relevance of results ($p < 0.01$). Larger absolute values indicate stronger correlations.

Factor	r	ρ	τ
$\|N\|$	-0.007	-0.041	-0.045
$\|E\|$	-0.007	-0.041	-0.045
$\min weight(e)$	-0.096	-0.039	-0.036
$\max weight(e)$	0.149	0.243	0.215
$\sum weight(e)$	0.048	0.142	0.123
$\overline{weight(e)}$	0.113	0.172	0.154
$\min weight(n)$	0.043	0.008	0.005
$\max weight(n)$	-0.097	-0.009	-0.010
$\sum weight(n)$	-0.095	0.035	0.026
$\overline{weight(n)}$	-0.095	0.031	0.024
pivoted normalization	0.264	0.389	0.310
Euclidean (L^2) distance	-0.132	-0.219	-0.207

weights of edges in the data graph:

$$weight(e) = \ln(1 + out(u)) \quad (1)$$

where e is an edge from u to v and $out(u)$ is the outdegree of u. Node weights are computed using the PageRank algorithm [2], which is the most common method in the literature. Pivoted normalization's document-related factors are computed over an entire search result and collection statistics are taken from the entire database. Unlike traditional IR, the collection (and consequently its statistics) is not well-defined when searching semi-structured and relational data.[3] Our decision to use statistics from a complete search result and the entire database follows Robertson *et al.*'s previous work [38] and techniques from XML retrieval [5].

Correlating Scoring Factors with Relevance

Before we present the creation of our learned scoring function, we present the correlation between each factor and the relevance of search results. We consider three different measures of the correlation between each factor and relevance: Pearson's r, which measures the degree of linear relationship [39]; Spearman's p, which measures the degree of dependence with any monotonic function [43]; and Kendall's τ, which is the number of bubble-sort swaps required to transform one list into another [26]. Kendall's τ is perhaps the best metric for comparing ordinal correlations [20], but we also report Pearson's r and Spearman's p because they are more common. In general, the trends are similar regardless of the measurement used.

Table 3 summarizes our findings. Note that none of the correlations are particularly strong although all the values are significant at the 0.01 level. Pivoted normalization weighting is moderately correlated with relevance. The Euclidean distance (which is 0 if the result contains all terms present in the query Q and 1 if the result does not contain any query terms) has one of the most negative correlations. These two results are not altogether unsurprising because pivoted normalization is a state-of-the-art scoring function and the Euclidean distance is a very coarse measure of relevance. While most relevant results will contain all the query terms, the converse is not true: a result is not relevant just because it contains the query terms—it must address the query's underlying information need.

The correlations for our different edge weight factors is extremely surprising. Note that the table simply uses the raw values of each factor. In contrast, proximity search systems minimize the weight of results. Hence, we expect a larger total edge weight ($\sum weight(e)$) to be *negatively* correlated with relevance or—stated conversely—a smaller sum should be positively correlated with relevance. Our results indicate the exact opposite: results with higher weights are more likely to be relevant than results with smaller weights! This result differs substantially from the conventional wisdom in this field, and we will return to it in our discussion of our scoring function in Section 4 and in the appendix.

[3]It is straightforward to show that calculating exact collection statistics in the context of relational keyword search requires violating desirable properties of the original IR scoring function [18, 32], estimation [33], or enumeration of all possible search results *a priori* [31].

3.2 Creation

We use 10-fold cross validation [27] to mitigate the threat of overfitting our scoring function. In 10-fold cross validation, the data is randomly partitioned into 10 different folds (subsets). Training uses 9 of the folds (that is, 90% of the original data), and the model is tested against the final fold. This process is repeated so each fold is used exactly once to test the model.

We use SVM^{rank} [21] to learn the weight for each factor in our scoring function. SVM^{rank} is designed to efficiently solve ordinal regression problems and operates on a set of training instances. The training instances are derived from potential query results, and each instance consists of a feature vector of all the factors that we include in our scoring function and the ideal rank of the result.

Because different enumeration algorithms have been shown to omit results [16], we pool the top-1000 results returned by 9 different systems [1, 7, 10, 17, 18, 19, 22, 32, 33] to create the set of training instances. Pooling ensures that we consider as many different query and result semantics as possible and increases the generality of our scoring function.

Binary relevance judgments are distributed as part of the test collection. While these judgments would be sufficient to train a classifier, they are not suitable for ordinal regression, and the limited number potentially could diminish training effectiveness. We augment the binary relevance judgments with marginal relevance judgments of the results returned by each system. Given a large number of factors and available training instances (see Table 4), a SVM may fail to terminate within a timely fashion. We give the SVM 1% of the available training instances and ignore the remainder. This subset always includes all the ideal (completely relevant) results and—if possible—at least as many marginally relevant and irrelevant results as the number of ideal results to ensure the training set covers the gamut of available relevance judgments.

3.3 Evaluation

We use a publicly available evaluation benchmark in our experiments. We briefly describe the datasets and queries but refer the reader to its original description [6] for additional details.

The evaluation workload includes three datasets: MONDIAL, the Internet Movie Database (IMDb), and Wikipedia.

Table 4: Characteristics of the evaluation datasets and number of training instances available.

	Evaluation		Training
Dataset	Tuples	Size (MB)	Instances
MONDIAL	17K	10	33500
IMDb	1.7M	427	132175
Wikipedia	200K	378	119969

Two datasets (IMDb and Wikipedia) are derived from popular websites. The MONDIAL database [34] contains geographical, political, and demographic information.The IMDb database, which is a subset of the original, was constructed from the IMDb's plain text files using IMDbPY 4.1. The Wikipedia database is drawn from the English Wikipedia and includes all the articles chosen for the 2008–2009 Wikipedia schools DVD. Table 4 summarizes the characteristics of the datasets.

The query workload distributed with the benchmark comprises 50 synthetic information needs for each dataset. While these information needs are not sampled from search engine query logs, they are designed to be representative of the types of queries users submit to search engines. In particular, the statistics of the supplied queries are similar to those encountered by web search engines.

Examining the existing binary relevance judgments suggests that they should be viewed as the *ideal recall-base* for each query. The ideal recall-base includes all non-overlapping results [29]. In recent years, evaluation forums have moved toward marginal relevance judgments, and systems should return results in decreasing order of their estimated marginal relevance. Hence, we extend the supplied relevance judgments by creating the *full recall-base*, which includes all results that have any relevance to the query. We construct the full recall-base by pooling the results returned by each system, and any result that completely overlaps an ideal result is judged marginally relevant.[4] We adopt the practice used at the Initiative for the Evaluation of XML retrieval (INEX) workshop for assigning relevance scores: the marginal relevance of a result is the ratio of relevant information to the total information [29, 30]. Issues related to overlap have been investigated in the context of XML retrieval [25] so we do not explore them further in this paper.

3.3.1 Metrics

We use a variety of metrics to evaluate our work. Precision is the ratio of the number of relevant documents retrieved to the total number of documents retrieved. When computing precision, we define the set of relevant results to include all results that have any relevance—i.e., they may only be marginally relevant to the query.

Precision @ k (P@k) is the mean precision value at a fixed retrieval depth (e.g., the first 10 results returned by a system). Mean reciprocal rank (MRR) is the reciprocal of the rank of the first relevant result retrieved by a system. Although neither of these metrics is robust, we report them to enable more direct comparison with previous evaluations. Average precision (AP) is the mean of the precision values calculated after each relevant result is returned. Any relevant result not retrieved receives a score of 0. Mean average precision (MAP) averages this value across all queries to compute a single-valued measurement of retrieval effectiveness. Normalized discounted cumulative gain (nDCG) was designed for evaluations with non-binary (i.e., marginal) relevance assessments. The family of cumulative gain metrics rewards systems that return results in the order of their relevance to the query.

Because our focus is ranking search results, we do not address the runtime performance or efficiency of any system. Result enumeration is largely orthogonal to ranking, and our objective in this paper is to investigate scoring results.[5]

3.3.2 Systems

We compare our learned scoring function, SVM rank, to 9 other relational keyword search systems. We implemented 6 systems (BANKS [1], DISCOVER [19], DISCOVER-II [18], Liu *et al.* [32], SPARK [33], and cover density ranking (CD) [7]) ourselves and obtained implementations of the other 3 systems (BANKS-II [22], DPBF [10], and BLINKS [17]). While we would like to compare our scoring function against all other systems described in the literature, little sharing of their source code hampers this objective.

All the systems adhere to their original description[6] although we did correct a number of minor deficiencies that we found in their specification or implementation. For each system, we set all tuning parameters to the values suggested by the authors. If a system failed due to exhausting virtual memory (> 2.7 GB), our results include anything output prior to the error. A system's omission from a table or figure indicates that no query returned even a single result on that particular dataset.

3.3.3 Results

Table 5 presents our results for P@k. Our results are similar to those previously reported [6]; the minor deviations are due to our full recall-base, which increases the total number of relevant results. On MONDIAL and Wikipedia, SVM rank lags slightly behind the best systems for P@1, but it scores best for P@10. However, SVM rank falters on the IMDb dataset, scoring only half as well as the best systems, which is likely due to their slight variations on our scoring factors. For example, BANKS uses a different edge weighting scheme than SVM rank. We note that SVM rank outperforms the systems that share exactly the same factors.

The results for MRR (Table 6) are very similar to the results of P@k. In fact, the relative performance of each system compared to the others is largely unchanged from P@1. SVM rank continues to provide excellent performance on the MONDIAL and Wikipedia datasets but is only average for IMDb.

In Figure 2, we present MAP (left) for each system and dataset. SVM rank nearly doubles the performance of any previous system on the MONDIAL dataset and also outscores the other systems for Wikipedia queries. On the IMDb dataset, SVM rank is edged out by cover density ranking.

[4]Unlike XML, which is tree-structured, relational keyword search systems create result trees from a data graph. We define two results as overlapping when one is a subset of the other and not when they merely share tuples in common.

[5]In some instances, combining enumeration and ranking (i.e., enumerating the top-k results in the order their final ranking) improves performance; we leave this task for future work due to the intricacies involved in their integration.

[6]The implementation of Liu *et al.*'s work [32] does not include phrase-based ranking.

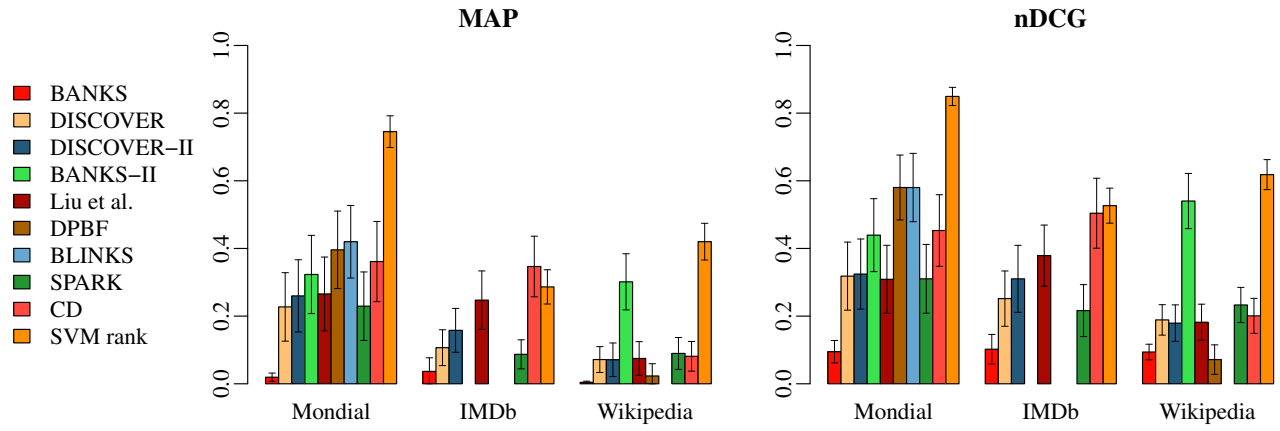


Figure 2: MAP ($\in [0,1]$, higher is better) and NDCG ($\in [0,1]$, higher is better) for each system and dataset. Systems are ordered by date of publication, top to bottom in the legend and left to right in the graphs. Omitted bars indicate that the system failed to retrieve *any* results for any query. The error bars provide 95% confidence intervals for the mean.

Cover density ranking uses a unique scoring scheme that differs dramatically from other systems and includes features not considered in our model. The results for MAP are considerably lower than those of P@k and MRR due to many systems failing to enumerate all the relevant results (e.g., due to different search semantics). Unlike P@k and MRR, these results differ dramatically from those previously reported [6], which again is attributable to using our full recall-base in lieu of the ideal recall-base supplied with the evaluation framework.

Figure 2 also shows nDCG (right) for each system and dataset. In terms of the relative rank of each system, these results are similar to MAP. However, nDCG is designed to reward systems that rank more relevant results ahead of less relevant results. We see several examples of systems that significantly improve under nDCG; indeed every system improves at least slightly compared to MAP. SVM rank shows a moderate boost in its measured effectiveness; this improvement is sufficient for SVM rank to edge cover density ranking as the best system on the IMDb dataset, which indicates that SVM rank is more likely than cover density ranking to rank more relevant results ahead of less relevant results.

Table 5: P@k ($\in [0,1]$, higher is better) for the systems on each dataset; the best score for each dataset is bolded. Omitted values indicate that the system failed to retrieve *any* results for any query.

$k = 1$

System	Mondial	IMDb	Wikipedia
BANKS [1]	0.420	**0.540**	0.500
DISCOVER [19]	0.620	0.140	0.120
DISCOVER-II [18]	0.580	0.120	0.180
BANKS-II [22]	0.720		0.500
Liu *et al.* [32]	0.660	0.400	**0.640**
DPBF [10]	0.740		0.020
BLINKS [17]	0.720		
SPARK [33]	0.560	0.100	0.280
Cover Density [7]	**0.780**	0.380	0.540
SVM rank	0.753	0.220	0.540

$k = 10$

System	Mondial	IMDb	Wikipedia
BANKS [1]	0.132	0.206	0.094
DISCOVER [19]	0.385	0.120	0.140
DISCOVER-II [18]	0.417	0.166	0.173
BANKS-II [22]	0.442		0.190
Liu *et al.* [32]	0.429	**0.440**	0.227
DPBF [10]	0.438		0.048
BLINKS [17]	0.594		
SPARK [33]	0.375	0.114	0.216
Cover Density [7]	0.588	0.388	0.452
SVM rank	**0.744**	0.221	**0.540**

4. DISCUSSION

In this section, we analyze the linear function that we used to score search results. We show that many features may be eliminated without having a significant impact on search quality. We also discuss the limitations of our approach.

Table 6: MRR ($\in [0,1]$, higher is better) for the systems on each dataset; the best score for each dataset is bolded. Omitted values indicate that the system failed to retrieve *any* results for any query.

System	Mondial	IMDb	Wikipedia
BANKS [1]	0.488	**0.608**	0.558
DISCOVER [19]	0.676	0.196	0.241
DISCOVER-II [18]	0.647	0.202	0.394
BANKS-II [22]	0.781		0.591
Liu *et al.* [32]	0.683	0.446	**0.748**
DPBF [10]	0.825		0.111
BLINKS [17]	0.771		
SPARK [33]	0.627	0.171	0.492
Cover Density [7]	**0.849**	0.456	0.619
SVM rank	0.834	0.332	0.613

Table 7: Comparison (nDCG) of SVM rank trained using different cross folds. The "baseline" is graphed in Figure 2.

	nDCG		
Dataset	baseline	μ	σ
MONDIAL	0.849	0.862	0.006
IMDb	0.527	0.525	0.002
Wikipedia	0.618	0.621	0.007

4.1 Analysis of Model

Using machine learning to create our scoring function potentially introduces several sources of error in our results. We consider these sources to show that their effect is negligible before moving on to feature selection.

4.1.1 Sensitivity

As mentioned previously, we use cross validation to train and to test our scoring function. Because cross validation requires randomly partitioning the query workload, it is possible that the partitions bias the results. To investigate this possibility, we compare retrieval effectiveness across 10 different random partitions (i.e., we repeat the cross validation 10 times). The results are shown in Table 7. As evidenced by the table, different cross folds do not significantly impact our results.

A second source of possible error in our results stems from using a fraction of the available training instances. Table 8 provides the search effectiveness from different percentages of training instances where we randomize the set of training instances included at each level. Because we ensure that all the ideal results are retained for each query, doubling the percentage does not exactly double the number of training instances. The table also shows that increasing the number of training instances increases the time required to train the model, from approximately 5 minutes (per fold) for 0.5% to almost 20 minutes for 3.0%. However, the impact that the training percentage has on search effectiveness is negligible. MAP sees moderate improvement when moving from 0.5% to 3.0%, but the impact on nDCG is much smaller.

4.1.2 Feature Selection

Feature selection tries to reduce the number of factors in the scoring function without significantly impacting search effectiveness. We start with our baseline scoring function that includes 12 different factors and then identify the factor with the lowest weight (that is, it contributes least to the final ranking) and train a new model without that factor. This process is then repeated.

Table 8: Average number of training instances and average training time across all cross-folds for different percentages of training instances. MAP and nDCG improve only slightly with more training instances.

	Training		Effectiveness	
%	Instances	Time (s)	MAP	nDCG
0.5	2279	298	0.457	0.657
1.0	3430	321	0.489	0.663
2.0	5909	550	0.509	0.664
3.0	8453	1024	0.516	0.667

While greedy backward selection is non-optimal—finding the optimal requires searching all possible combinations of factors [28]—it does provide a simple measure of the importance of each factor. We note that there is overlap among our factors: a principal component analysis shows that 98% of the variability can be attributed to 6 different components. Hence, we should be able to remove a number of factors without significantly impacting search quality.

Table 9 shows nDCG as we remove factors from our scoring function. Note that the number of factors removed is cumulative—i.e., the first variation is SVM rank without $\max weight(n)$ and the second variation is SVM rank without $\max weight(n)$ and without $\sum weight(n)$. The final variation, SVM rank (minimal), includes the factors *not* appearing in the table: $\min weight(n)$ and pivoted normalization. As evidenced by the table, retrieval effectiveness remains largely unchanged as several factors are removed. However, performance drops precipitously when the fourth factor (the Euclidean distance between a result and the query) is removed. This impact is most noticeable for the MONDIAL dataset. We posit that this factor is very noisy, having almost no impact for many queries but is very important for a few.

We can remove all but two factors from our scoring function without seriously impacting search effectiveness. Retrieval effectiveness actually improves on the Wikipedia dataset as we eliminate additional features. This trend is not completely surprising given that pivoted normalization weighting was developed by the IR community to score lengthy text documents and the Wikipedia articles are most similar to this use case.

4.2 Comparison with Alternative Functions

If we take the order in which we remove features (see Table 9) to be indicative of their importance, we would conclude that node weights (with the exception of the minimum node weight, which is included in SVM rank (minimal)) are relatively unimportant when ranking search results. Similarly, the total sum of edge weights is the first edge-based factor to be dropped. This result clashes sharply with conventional wisdom in this field. In particular, all proximity search systems attempt to rank search results in order of their total edge weight. Strangely enough, we find that other factors (e.g., the minimum edge weight) are more indicative of relevance as evidenced by its stronger correlation with relevance (see Table 3) and by its longer retention in feature selection. We hope that this opens the door for future debate regarding the design of new scoring functions.

In Figure 3, we consider our two versions of SVM rank and two alternatives that score search results more conventionally. We compare SVM rank (minimal), which includes only 2 factors ($\min weight(n)$ and pivoted normalization), to one using pivoted normalization and $\sum weight(e)$ and to one using $\sum weight(n)$ and $\sum weight(e)$. Both of these scoring functions are more similar to those proposed in previous work. The weights for the various factors are determined using SVM^{rank}; hence, the alternatives may be viewed as a rough upper bound of the search effectiveness of existing techniques. As evidenced in the figure, both SVM rank and SVM rank (minimal) always outperforms the more conventional scoring functions. We conclude—contrary to the established norm in this area—that ranking results by their total edge weight

Table 9: Comparison of retrieval effectiveness when removing factors (listed in Table 3) from the scoring function. The factors that have been removed are *cumulative*—i.e., the second variation of SVM rank (the 3rd row of the table) does not include $\max weight(n)$ or $\sum weight(n)$. SVM rank (minimal) includes the 2 factors that were *not* removed: $\min weight(n)$ and pivoted normalization. It is the same as the next-to-last row of the table (but is duplicated for clarity). The best score for each dataset is bolded.

Removal		Mondial			IMDb			Wikipedia		
			Improvement			Improvement			Improvement	
Order	Factor	NDCG	Δ	%	NDCG	Δ	%	NDCG	Δ	%
	SVM rank	0.849	—	—	**0.527**	—	—	0.618	—	—
1	$\max weight(n)$	0.854	0.005	0.6	0.525	-0.002	-0.4	0.627	0.009	1.5
2	$\sum \overline{weight(n)}$	**0.866**	**0.017**	**2.0**	0.515	-0.012	-2.3	0.630	0.012	1.9
3	$\overline{weight(n)}$	0.855	0.006	0.7	0.518	-0.009	-1.7	0.616	-0.002	-0.3
4	Euclidean (L^2) distance	0.673	-0.177	-20.8	0.454	-0.073	-13.6	0.560	-0.058	-7.7
5	$\sum weight(e)$	0.681	-0.169	-19.9	0.464	-0.063	-12.0	0.576	-0.042	-6.8
6	$\lvert N \rvert$	0.679	-0.170	-20.0	0.446	-0.081	-15.4	0.580	-0.038	-6.1
7	$\max \overline{weight(e)}$	0.676	-0.173	-20.4	0.456	-0.071	-13.5	0.573	-0.045	-7.3
8	$\overline{weight(e)}$	0.691	-0.158	-18.6	0.469	-0.058	-9.4	**0.634**	**0.016**	**2.8**
9	$\lvert E \rvert$	0.711	-0.139	-16.4	0.494	-0.033	-6.3	0.619	0.001	0.2
10	$\min weight(e)$	0.772	-0.077	-9.1	0.495	-0.032	-6.1	0.623	0.005	0.8
	SVM rank (minimal)	0.772	-0.077	-9.1	0.495	-0.032	-6.1	0.623	0.005	0.8

is not an ideal ranking scheme. This conclusion overturns general intuition and questions the importance of algorithms that attempt to enumerate results in the order of their total edge weight. We refer the interested reader to the appendix for further discussion of this result.

4.3 Threats to Validity

There are several factors that may have impacted our evaluation. First, our results are dependent on our evaluation datasets. As mentioned previously, our evaluation includes 3 diverse datasets with a large query workload for each so the likelihood that our results generalize to other databases is high. Second, we do not address the enumeration of results in this work. A variety of enumeration algorithms has already been proposed; it would be straightforward to marry our scoring function with any one of these algorithms.

Our work uses a linear SVM to fit a linear function for scoring results. This decision follows naturally from most existing relational keyword search scoring functions, which are linear. In contrast, most scoring functions used by the IR community are non-linear and include—for example—logarithmic damping of term frequency and inverse document frequency. Although our edge weighting scheme is non-linear, we combine all our factors linearly, and it is certainly possible for a different combination function to provide even better results.

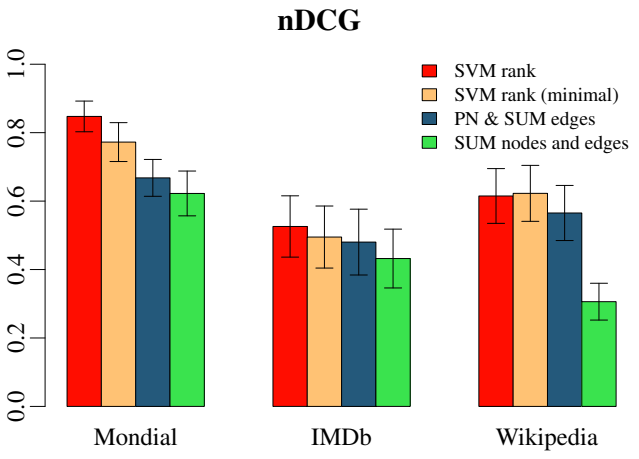


Figure 3: Comparison of retrieval effectiveness among SVM rank and traditional scoring functions proposed in the literature.

5. RELATED WORK

A dichotomy exists within relational keyword search systems. Proximity search endeavors to minimize the distance between search terms within a data graph. While minimizing the total edge weight is NP-hard [36] (it is equivalent to the group Steiner tree problem [11]), different semantics allow more efficient enumeration of results. Schema-based approaches identify search results by executing queries against the relational database, and many of these systems have adopted IR-style scoring functions to rank the search results. Yu *et al.* [46] provide an excellent survey of relational keyword search.

BANKS [1] proposed the backward expanding search heuristic to enumerate search results. BANKS-II [22] uses the bidirectional search heuristic to alleviate performance problems. DPBF [10] finds the optimal group Steiner tree via a dynamic programming algorithm. BLINKS [17] creates a two-level index to improve enumeration performance. EASE [31] precomputes all r-radius Steiner graphs within a data graph and ranks search results with both structural compactness and IR-style scoring factors. Golenberg *et al.* [16] enumerate search results in approximate order by height rather than weight. Dalvi *et al.* [9] investigate keyword search on data graphs that do not fit within main memory. STAR [23] provides a pseudo-polynomial time algorithm to identify search results.

DISCOVER [19] was an early system that executed SQL queries to identify potential search results. DISCOVER-II [18] adopted pivoted normalization for ranking search

results and also provided efficient query processing algorithms. Liu *et al.* [32] proposed a variety of modifications and additional factors for pivoted normalization. SPARK [33] proposed new query processing techniques to handle its non-monotonic scoring function. Qin *et al.* [35] address query processing techniques under different query semantics. Structured cover density ranking [7] ranks search results in decreasing order of their coordination with the query terms.

The major focus for much of this previous work is the enumeration of search results. The significant amount of work on enumeration algorithms complements our work, for our scoring function can be used to re-rank results following their enumeration (as proposed by Golenberg *et al.* [16]). We note that further improvements in enumeration algorithms are likely when the scoring function need not minimize the total edge weight of results. Coffman and Weaver [6] analyze the various ranking schemes that appear in the literature. SVM rank outperforms 9 of 11 existing ranking schemes. The primary reasons that we did not compare our work to the remaining scoring functions are little sharing of source code and significant reimplementation effort.

To the best of our knowledge, using machine learning to explore how to weight different factors when scoring search results has not been previously investigated in the context of relational keyword search. Machine learning in IR is a well-studied topic. Sebastiani [40] provides an overview of automated text categorization. Joachims [20] applies ordinal regression to clickstream data and shows that the learned function outperforms existing search engines. Burges *et al.* [3] and Richardson *et al.* [37] also apply learning to rank to web search. Cao *et al.* [4] address how to adapt ranking SVMs to document retrieval. Geng *et al.* [13] consider the problem of feature selection and proposes an approach based on the importance and similarity of the various features. Joachims [20] and Richardson *et al.*'s [37] work is closest to our own, but our context and the features that we consider both differ. Incorporating Cao *et al.*'s work [4] on ranking SVMs would have undoubtedly improved our search effectiveness for P@k and MRR, the instances where SVM rank was only comparable to existing systems.

6. CONCLUSION AND FUTURE WORK

Although many scoring functions for relational keyword search have been proposed in the literature, relatively little work has considered their effectiveness when ranking results. Instead, researchers have relied on intuition and anecdotal evidence without extensive experiments to determine whether or not various factors are indicative of relevance. In this work, we consider factors that have been previously proposed in the literature, and we show that many existing scoring functions—indeed complete systems—have been constructed around factors that are not highly correlated with relevance. The best example is identifying group Steiner trees, which is the goal of many systems. We use machine learning to create a new scoring function, which outperforms existing approaches. Finally, we simplify our scoring function by eliminating all the features except the minimum node weight and pivoted normalization and show that it continues to outperform existing approaches.

In the future, we would like to investigate the variation in search effectiveness from different formulations of the same feature. For example, different methods for assigning edge weights in the data graph could alter our results; we note that this question has not been addressed by any previous work in this field. We would also like to investigate non-linear scoring functions.

As a challenge to the research community, differentiating the separate steps of enumeration and ranking would lend clarity to these largely orthogonal issues. In particular, research focused on enumeration algorithms should reuse existing scoring functions (to the greatest extent possible) so as not to merge evaluations of performance with search effectiveness. Likewise, new ranking schemes should not only be compared to existing systems, but an analysis of the various factors (i.e., their impact on search effectiveness) should also be provided.

7. ACKNOWLEDGMENTS

Ray Buse provided insight on how to conduct this study and helpful comments regarding drafts of this paper. We thank Ding *et al.* and He *et al.* for sharing implementations of systems that we included in our evaluation.

References

[1] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword Searching and Browsing in Databases using BANKS. In *ICDE '02*, pages 431–440, February 2002.

[2] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, 1998.

[3] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to Rank using Gradient Descent. In *ICML '05*, pages 89–96, 2005.

[4] Y. Cao, J. Xu, T.-Y. Liu, H. Li, Y. Huang, and H.-W. Hon. Adapting Ranking SVM to Document Retrieval. In *SIGIR '06*, pages 186–193, 2006.

[5] C. L. A. Clarke. Controlling Overlap in Content-Oriented XML Retrieval. In *SIGIR '05*, pages 314–321, 2005.

[6] J. Coffman and A. C. Weaver. A Framework for Evaluating Database Keyword Search Strategies. In *CIKM '10*, pages 729–738, October 2010.

[7] J. Coffman and A. C. Weaver. Structured Data Retrieval using Cover Density Ranking. In *KEYS '10*, pages 1–6, June 2010.

[8] J. Coffman and A. C. Weaver. Learning to Rank Results in Relational Keyword Search. Technical Report CS-2011-06, University of Virginia, 2011.

[9] B. B. Dalvi, M. Kshirsagar, and S. Sudarshan. Keyword Search on External Memory Data Graphs. *PVLDB*, 1(1):1189–1204, 2008.

[10] B. Ding, J. X. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin. Finding Top-k Min-Cost Connected Trees in Databases. In *ICDE '07*, pages 836–845, April 2007.

[11] S. E. Dreyfus and R. A. Wagner. The Steiner Problem in Graphs. *Networks*, 1(3):195–207, 1971.

[12] S. Fox. Search engines. Technical report, Pew Internet and American Life Project, July 2002. `http://www.pewinternet.org/Reports/2002/Search-Engines.aspx`.

[13] X. Geng, T.-Y. Liu, T. Qin, and H. Li. Feature Selection for Ranking. In *SIGIR '07*, pages 407–414, 2007.

[14] Global Search Market Grows 46 Percent in 2009. `http://www.comscore.com/Press_Events/Press_Releases/2010/1/Global_Search_Market_Grows_46_Percent_in_2009`, January 2010.

[15] R. Goldman, N. Shivakumar, S. Venkatasubramanian, and H. Garcia-Molina. Proximity Search in Databases. In *VLDB '98*, pages 26–37, August 1998.

[16] K. Golenberg, B. Kimelfeld, and Y. Sagiv. Keyword Proximity Search in Complex Data Graphs. In *SIGMOD '08*, pages 927–940, June 2008.

[17] H. He, H. Wang, J. Yang, and P. S. Yu. BLINKS: Ranked Keyword Searches on Graphs. In *SIGMOD '07*, pages 305–316, June 2007.
[18] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient IR-style Keyword Search over Relational Databases. In *VLDB '03*, pages 850–861, September 2003.
[19] V. Hristidis and Y. Papakonstantinou. DISCOVER: Keyword Search in Relational Databases. In *VLDB '02*, pages 670–681. VLDB Endowment, August 2002.
[20] T. Joachims. Optimizing Search Engines using Clickthrough Data. In *KDD '02*, pages 133–142, 2002.
[21] T. Joachims. Training linear SVMs in linear time. In *KDD '06*, pages 217–226, August 2006.
[22] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional Expansion For Keyword Search on Graph Databases. In *VLDB '05*, pages 505–516, August 2005.
[23] G. Kasneci, M. Ramanath, M. Sozio, F. M. Suchanek, and G. Weikum. STAR: Steiner-Tree Approximation in Relationship Graphs. In *ICDE '09*, pages 868–879, March 2009.
[24] G. Kasneci, F. M. Suchanek, G. Ifrim, M. Ramanath, and G. Weikum. NAGA: Searching and Ranking Knowledge. In *ICDE '08*, pages 953–962, April 2008.
[25] J. Kekäläinen, M. Junkkari, P. Arvola, and T. Aalto. TRIX 2004 – Struggling with the Overlap. In N. Fuhr, M. Lalmas, S. Malik, and Z. Szlávik, editors, *Advances in XML Information Retrieval*, volume 3493 of *LNCS*, pages 145–162. Springer, 2005.
[26] M. Kendall. A New Measure of Rank Correlation. *Biometrika*, 30(1/2):81–93, June 1938.
[27] R. Kohavi. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *IJCAI*, volume 2, pages 1137–1143, 1995.
[28] R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1–2):273–324, 1997.
[29] M. Lalmas, G. Kazai, J. Kamps, J. Pehcevski, B. Piwowarski, and S. Robertson. INEX 2006 Evaluation Measures. In N. Fuhr, M. Lalmas, and A. Trotman, editors, *Comparative Evaluation of XML Information Retrieval Systems*, volume 4518 of *LNCS*, pages 20–34. Springer, 2007.
[30] M. Lalmas and A. Tombros. INEX 2002 - 2006: Understanding XML Retrieval Evaluation. In C. Thanos, F. Borri, and L. Candela, editors, *Digital Libraries: Research and Development*, volume 4877 of *LNCS*, pages 187–196. Springer, 2007.
[31] G. Li, B. C. Ooi, J. Feng, J. Wang, and L. Zhou. EASE: An Effective 3-in-1 Keyword Search Method for Unstructured, Semi-structured and Structured Data. In *SIGMOD '08*, pages 903–914, June 2008.
[32] F. Liu, C. Yu, W. Meng, and A. Chowdhury. Effective Keyword Search in Relational Databases. In *SIGMOD '06*, pages 563–574, June 2006.
[33] Y. Luo, X. Lin, W. Wang, and X. Zhou. SPARK: Top-k Keyword Query in Relational Databases. In *SIGMOD '07*, pages 115–126, June 2007.
[34] W. May. Information Extraction and Integration with Florid: The Mondial Case Study. Technical Report 131, Universität Freiburg, Institut für Informatik, 1999.
[35] L. Qin, J. X. Yu, and L. Chang. Keyword Search in Databases: The Power of RDBMS. In *SIGMOD '09*, pages 681–694, June 2009.
[36] G. Reich and P. Widmayer. Beyond Steiner's problem: A VLSI oriented generalization. In M. Nagl, editor, *Graph-Theoretic Concepts in Computer Science*, volume 411 of *LNCS*, pages 196–210. Springer Berlin / Heidelberg, 1990.
[37] M. Richardson, A. Prakash, and E. Brill. Beyond PageRank: Machine Learning for Static Ranking. In *WWW '06*, pages 707–715, 2006.
[38] S. Robertson, H. Zaragoza, and M. Taylor. Simple BM25 Extension to Multiple Weighted Fields. In *CIKM '04*, pages 42–49, November 2004.
[39] J. L. Rodgers and W. A. Nicewander. Thirteen Ways to Look at the Correlation Coefficient. *The American Statistician*, 42(1):59–66, February 1988.
[40] F. Sebastiani. Machine Learning in Automated Text Categorization. *ACM Computing Surveys*, 34:1–47, March 2002.
[41] A. Singhal. Modern Information Retrieval: A Brief Overview. *IEEE Data Engineering Bulletin*, 24(4):35–42, December 2001.
[42] A. Singhal, J. Choi, D. Hindle, D. Lewis, and F. Pereira. AT&T at TREC-7. In *TREC-7*, pages 239–252, November 1999.
[43] C. Spearman. The Proof and Measurement of Association between Two Things. *The American Journal of Psychology*, 15(1):72–101, January 1904.
[44] F. M. Suchanek, G. Kasneci, and G. Weikum. YAGO: A Core of Semantic Knowledge. In *WWW '07*, pages 697–706, 2007.
[45] M. Taylor, H. Zaragoza, N. Craswell, S. Robertson, and C. Burges. Optimisation Methods for Ranking Functions with Multiple Parameters. In *CIKM '06*, pages 585–593, November 2006.
[46] J. X. Yu, L. Qin, and L. Chang. *Keyword Search in Databases*. Morgan and Claypool Publishers, 1st edition, 2010.

APPENDIX

Justification for Minimizing Total Edge Weight

To the best of our knowledge, Goldman *et al.* [15] first proposed proximity search in databases. Intuitively, users want results that are more closely related to be ranked ahead of results that are less closely related. For example, if two nodes are connected by a single edge, this relationship is stronger than one that requires multiple edges and intermediary nodes. Goldman *et al.* [15] do not consider search effectiveness in their evaluation, but BANKS [1] provides anecdotal evidence that total edge weight is correlated with relevance. In NAGA [24] and STAR [23], Kasneci *et al.* claim that a Steiner-tree-based scoring function is correlated with relevance.

We cannot immediately reconcile these claims with our results, but we have identified three possible factors that might account for this discrepancy. First, Kasneci *et al.* search the YAGO knowledge base [44], which includes millions of facts connected by predefined relationships. Its structure is considerably different from the graphs created from a relational database, and NAGA's evaluation [24] did not include graphs created from relational data. Second, NAGA [24] does indicate that its weighting of graph edges (based on the strength of its beliefs regarding each relationship) outperforms BANKS's scoring function [1], which cannot capture the informativeness of different relationships. Hence, alternative edge weighting schemes might produce different results. BANKS significantly outperforms SVM rank on the IMDb dataset for the highest ranked results (see the P@1 and MRR results), and one possible reason is its different method for assigning edge weights. A formal examination of these different edge weighting schemes would undoubtedly be useful for future work in this area. Third, the general intuition—results should be composed of closely related tuples—seems valid, particularly for identifying the *most relevant* result, but it may not be as good at identifying marginally relevant results, which would cause it to perform worse at higher recall levels (e.g., on MAP and nDCG). Again, a formal study of this phenomenon seems prudent for future work in this area.