An Empirical Performance Evaluation of Relational Keyword Search Systems

University of Virginia
Department of Computer Science

Technical Report CS-2011-07

Joel Coffman, Alfred C. Weaver

Department of Computer Science, University of Virginia
Charlottesville, VA, USA
{jcoffman,weaver}@cs.virginia.edu

***Abstract*— In the past decade, extending the keyword search paradigm to relational data has been an active area of research within the database and information retrieval (IR) community. A large number of approaches have been proposed and implemented, but despite numerous publications, there remains a severe lack of standardization for system evaluations. This lack of standardization has resulted in contradictory results from different evaluations, and the numerous discrepancies muddle what advantages are proffered by different approaches. In this paper, we present a thorough empirical performance evaluation of relational keyword search systems. Our results indicate that many existing search techniques do not provide acceptable performance for realistic retrieval tasks. In particular, memory consumption precludes many search techniques from scaling beyond small datasets with tens of thousands of vertices. We also explore the relationship between execution time and factors varied in previous evaluations; our analysis indicates that these factors have relatively little impact on performance. In summary, our work confirms previous claims regarding the unacceptable performance of these systems and underscores the need for standardization—as exemplified by the IR community—when evaluating these retrieval systems.**

I. Introduction

The ubiquitous search text box has transformed the way people interact with information. Nearly half of all Internet users use a search engine daily [10], performing in excess of 4 billion searches [11]. The success of keyword search stems from what it does not require—namely, a specialized query language or knowledge of the underlying structure of the data. Internet users increasingly demand keyword search interfaces for accessing information, and it is natural to extend this paradigm to relational data. This extension has been an active area of research throughout the past decade. However, we are not aware of any research projects that have transitioned from proof-of-concept implementations to deployed systems. We posit that the existing, ad hoc evaluations performed by researchers are not indicative of these systems' real-world performance, a claim that has surfaced recently in the literature [1], [5], [33].

Despite the significant number of research papers being published in this area, existing empirical evaluations ignore or only partially address many important issues related to search performance. Baid *et al.* [1] assert that existing systems have unpredictable performance, which undermines their usefulness for real-world retrieval tasks. This claim has little support in the existing literature, but the failure for these systems to gain a foothold implies that robust, independent evaluation is necessary. In part, existing performance problems may be obscured by experimental design decisions such as the choice of datasets or the construction of query workloads. Consequently, we conduct an independent, empirical evaluation of existing relational keyword search techniques using a publicly available benchmark to ascertain their real-world performance for realistic query workloads.

A. Overview of Relational Keyword Search

Keyword search on semi-structured data (e.g., XML) and relational data differs considerably from traditional IR.[1] A discrepancy exists between the data's physical storage and a logical view of the information. Relational databases are normalized to eliminate redundancy, and foreign keys identify related information. Search queries frequently cross these relationships (i.e., a subset of search terms is present in one tuple and the remaining terms are found in related tuples), which forces relational keyword search systems to recover a logical view of the information. The implicit assumption of keyword search—that is, the search terms are related—complicates the search process because typically there are many possible relationships between two search terms. It is almost always possible to include another occurrence of a search term by adding tuples to an existing result. This realization leads to tension between the compactness and coverage of search results.

Figure 1 provides an example of keyword search in relational data. Consider the query "Switzerland Germany" where the user

[1] In this paper, we focus on keyword search techniques for relational data, and we do not discuss approaches designed for XML.

Country		
Code	Name	Capital
A	Austria	Vienna
CH	Switzerland	Bern
D	Germany	Berlin
F	France	Paris
FL	Liechtenstein	Vaduz
I	Italy	Rome

Borders		
C_1	C_2	Length
A	D	784
A	I	430
CH	A	164
CH	D	334
CH	F	573
CH	I	740
F	D	451
FL	A	37
FL	CH	41

Query: "Switzerland Germany"

Results:

1	Switzerland ← [borders] → Germany
2	Switzerland ← [borders] → Austria ← [borders] → Germany
2	Switzerland ← [borders] → France ← [borders] → Germany
4	Switzerland ← [borders] → Italy ← [borders] → Austria ← [borders] → Germany
4	Switzerland ← [borders] → Italy ← [borders] → France ← [borders] → Germany
4	Switzerland ← [borders] → Liechtenstein ← [borders] → Austria ← [borders] → Germany
7	Switzerland ← [borders] → Austria ← [borders] → Italy ← [borders] → France ← [borders] → Germany

Fig. 1. Example relational data from the MONDIAL database (left) and search results (right). The search results are ranked by size (number of tuples), which accounts for the ties in the list.

wants to know how the two countries are related. The borders relation indicates that the two countries are adjacent. However, Switzerland also borders Austria, which borders Germany; Switzerland borders France, which borders Germany; etc. As shown on the right in the figure, we can continue to construct results by adding intermediary countries, and we are only considering two relations and a handful of tuples from a much larger database!

Creating coherent search results from discrete tuples is the primary reason that searching relational data is significantly more complex than searching unstructured text. Unstructured text allows indexing information at the same granularity as the desired results (e.g., by documents or sections within documents). This task is impractical for relational data because an index over logical (or materialized) views is considerably larger than the original data [1], [31].

B. Contributions and Outline

As we discuss later in this paper, many relational keyword search systems approximate solutions to intractable problems. Researchers consequently rely on empirical evaluation to validate their heuristics. We continue this tradition by evaluating these systems using a benchmark designed for relational keyword search. Our holistic view of the retrieval process exposes the real-world tradeoffs made in the design of many of these systems. For example, some systems use alternative semantics to improve performance while others incorporate more sophisticated scoring functions to improve search effectiveness. These tradeoffs have not been the focus of prior evaluations.

The major contributions of this paper are as follows:

- We conduct an independent, empirical performance evaluation of 7 relational keyword search techniques, which doubles the number of comparisons as previous work.
- Our results do not substantiate previous claims regarding the scalability and performance of relational keyword search techniques. Existing search techniques perform poorly for datasets exceeding tens of thousands of vertices.
- We show that the parameters varied in existing evaluations are at best loosely related to performance, which is likely due to experiments not using representative datasets or query workloads.
- Our work is the first to combine performance and search effectiveness in the evaluation of such a large number of systems. Considering these two issues in conjunction provides better understanding of these two critical tradeoffs among competing system designs.

The remainder of this paper is organized as follows. In Section II, we motivate this work by describing existing evaluations and why an independent evaluation of these systems is warranted. Section III formally defines the problem of keyword search in relational data graphs and describes the systems included in our evaluation. Section IV describes our experimental setup, including our evaluation benchmark and metrics. In Section V, we describe our experimental results, including possible threats to validity. We review related work in Section VI and provide our conclusions in Section VII.

II. MOTIVATION FOR INDEPENDENT EVALUATION

Most evaluations in the literature disagree about the performance of various search techniques, but significant experimental design differences may account for these discrepancies. We discuss three such differences in this section.

A. Datasets

Table I summarizes the datasets and the number of queries used in previous evaluations.[2] Although this table suggests some uniformity in evaluation datasets, their content varies dramatically. Consider the evaluations of BANKS-II [17], BLINKS [13], and STAR [18]. Only BANKS-II's evaluation includes the entire Digital Bibliography & Library Project (DBLP)[3] and the Internet Movie Database (IMDb) dataset. Both BLINKS and STAR use smaller subsets to facilitate comparison with systems that assume the data graph fits entirely within main memory. The literature does not address the representativeness of database subsets, which is a serious threat because the choice of a subset has a profound effect on the experimental results. For example, a subset containing 1% of the original data is two orders of magnitude easier to search than the original database due to fewer tuples containing search terms.

[2]Omitted table entries indicate that the information was not provided in the description of the evaluation.

[3]http://dblp.uni-trier.de/

TABLE I

STATISTICS FROM PREVIOUS EVALUATIONS

System	Dataset	$\|V\|$	$\|E\|$	$\|Q\|$
BANKS [2]	bibliographic	100K	300K	7
DISCOVER [15]	TPC-H			200
DISCOVER-II [14]	DBLP			100
BANKS-II [17]	DBLP	2M	9M	200
	IMDb	2M	9M	
Liu *et al.* [21]	lyrics	196K	192K	50
DPBF [8]	DBLP	7.9M		500
	MovieLens	1M	1M	600
BLINKS [13]	DBLP	409K	591K	60
	IMDb	68K	248K	40
SPARK [22]	DBLP	882K	1.2M	18
	IMDb	9.8M	14.8M	22
	MONDIAL	10K		35
EASE [20]	DBLife	10K		5
	DBLP	12M		5
	MovieLens	1M		5
	previous 3			5
Golenberg *et al.* [12]	MONDIAL			36
BANKS-III [6]	DBLP	1.8M	8.5M	8
	IMDb	1.7M	1.9M	4
STAR [18]	DBLP	15K	150K	180
	IMDb	30K	80K	180
	YAGO	1.7M	14M	120

Legend

$|V|$ number of nodes (tuples)
$|E|$ number of edges in data graph
$|Q|$ number of queries in workload

B. Query Workloads

The query workload is another critical factor in the evaluation of these systems. The trend is for researchers either to create their own queries or to create queries from terms selected randomly from the corpus. The latter strategy is particularly poor because queries created from randomly-selected terms are unlikely to resemble real user queries [23]. The number of queries used to evaluate these systems is also insufficient. The traditional minimum for evaluating retrieval systems is 50 queries [32] and significantly more may be required to achieve statistical significance [34]. Only two evaluations that use relatistic query workloads meet this minimum number of information needs.

C. Experimental Discrepancies

Discrepancies among existing evaluations are prevalent. Table II lists the mean execution times of systems from three evaluations that use DBLP and IMDb databases. The table rows are search techniques; the columns are different evaluations of these techniques. Empty cells indicate that the system was not included in that evaluation. According to its authors, BANKS-II "significantly outperforms" [17] BANKS, which is supported by BANKS-II's evaluation, but the most recent evaluation contradicts this claim especially on DBLP. Likewise, BLINKS claims to outperform BANKS-II "by at least an order of magnitude in most cases" [13], but when evaluated by other researchers, this statement does not hold.

We use Table II to motivate two concerns that we have regarding existing evaluations. First, the difference in the relative performance of each system is startling. We do not expect the most recent evaluation to downgrade the *orders of magnitude* performance improvements to performance degradations, which is the certainly the case on the DBLP dataset. Second, the absolute execution times for the search techniques vary widely across different evaluations. The original evaluation of each system claims to provide "interactive" response times (on the order of a few seconds),[4] but other evaluations strongly refute this claim.

TABLE II

EXAMPLE OF CONTRADICTORY RESULTS IN THE LITERATURE

	execution time (s)						Evaluation
	DBLP			IMDb			
System	[17]	[13]	[18]	[17]	[13]	[18]	
BANKS [2]	14.8		5.9	5.0		10.6	
BANKS-II [17]	**0.7**	44.7	7.9	**0.6**	5.9	6.6	
BLINKS [13]		**1.2**	19.1		**0.2**	2.8	
STAR [18]			**1.2**			**1.6**	

III. RELATIONAL KEYWORD SEARCH SYSTEMS

Given our focus on empirical evaluation, we adopt a general model of keyword search over data graphs. This section presents the search techniques included in our evaluation; other relational keyword search techniques are mentioned in Section VI.

Problem definition: We model a relational database as a graph $G = (V, E)$. Each vertex $v \in V$ corresponds to a tuple in the relational database. An edge $(u, v) \in E$ represents each relationship (i.e., foreign key) in the relational database. Each vertex is decorated with the set of terms it contains. A query Q comprises a list of terms. A result for Q is a tree T that is reduced with respect to $Q' \subseteq Q$; that is, T contains all the terms of Q' but no proper subtree that also contains all of them.[5] Results are ranked in decreasing order of their estimated relevance to the information need expressed by Q.

A. Schema-based Systems

Schema-based approaches support keyword search over relational databases via direct execution of SQL commands. These techniques model the relational schema as a graph where edges denote relationships between tables. The database's full text indices identify all tuples that contain search terms, and a join expression is created for each possible relationship between these tuples.

DISCOVER [15] creates a set of tuples for each subset of search terms in the database relations. A candidate network is a tree of tuple sets where edges correspond to relationships in the database schema. DISCOVER enumerates candidate networks using a breadth-first algorithm but limits the maximum size to ensure efficient enumeration. A smaller size improves performance but risks missing results. DISCOVER creates a join expression for each candidate network, executes the join

[4] BANKS claims that most queries "take about a second to a few seconds" the execute against a bibliographic database [2].

[5] Alternative semantics are also possible—e.g., defining a result as a graph [19], [20], [28].

expression against the underlying database to identify results, and ranks these results by the number of joins.

Hristidis *et al.* [14] refined DISCOVER by adopting pivoted normalization scoring [30] to rank results:

$$\sum_{t \in Q} \frac{1 + \ln(1 + \ln tf)}{1 - s + s \cdot \frac{dl}{avgdl}} \cdot qtf \cdot \ln\left(\frac{N+1}{df}\right) \quad (1)$$

where t is a query term, $(q)tf$ is the frequency of the (query) term, s is a constant (usually 0.2), dl is the document length, $avgdl$ is the mean document length, N is the number of documents, and df is the number of documents that contain t. The score of each attribute (i.e., a document) in the tree of tuples is summed to obtain the total score. To improve scalability, DISCOVER-II creates only a single tuple set for each database relation and supports top-k query processing because users typically view only the highest ranked search results.

B. Graph-based Systems

The objective of proximity search is to minimize the weight of result trees. This task is a formulation of the group Steiner tree problem [9], which is known to be NP-complete [29]. Graph-based search techniques are more general than schema-based approaches, for relational databases, XML, and the Internet can all be modeled as graphs.

BANKS [2] enumerates results by searching the graph backwards from vertices that contain query keywords. The backward search heuristic concurrently executes copies of Dijkstra's shortest path algorithm [7], one from each vertex that contains a search term. When a vertex has been labeled with its distance to each search term, that vertex is the root of a directed tree that is a result to the query.

BANKS-II [17] augments the backward search heuristic [2] by searching the graph forwards from potential root nodes. This strategy has an advantage when the query contains a common term or when a copy of Dijkstra's shortest path algorithm reaches a vertex with a large number of incoming edges. Spreading activation prioritizes the search but may cause the bidirectional search heuristic to identify shorter paths after creating partial results. When a shorter path is found, the existing results must be updated recursively, which potentially increases the total execution time.

Although finding the optimal group Steiner tree is NP-complete, there are efficient algorithms to find the optimal tree for a fixed number of terminals (i.e., search terms). DPBF [8] is a dynamic programming algorithm for the optimal solution but remains exponential in the number of search terms. The algorithm enumerates additional results in approximate order.

He *et al.* [13] propose a bi-level index to improve the performance of bidirectional search [17]. BLINKS partitions the graph into blocks and constructs a block index and intra-block index. These two indices provide a lower bound on the shortest distance to keywords, which dramatically prunes the search space.

STAR [18] is a pseudopolynomial-time algorithm for the Steiner tree problem. It computes an initial solution quickly and then improves this result iteratively. Although STAR approximates the optimal solution, its approximation ratio is significantly better than previous heuristics.

TABLE III
CHARACTERISTICS OF THE EVALUATION DATASETS

Dataset	$\|V\|$	$\|E\|$	$\|T\|$
MONDIAL	17	56	12
IMDb	1673	6075	1748
Wikipedia	206	785	750

Legend, all values are in *thousands*

$\|V\|$	number of nodes (tuples)	$\|T\|$	number of unique terms
$\|E\|$	number of edges in data graph		

IV. EVALUATION FRAMEWORK

In this section, we present our evaluation framework. We start by describing the benchmark [5] that we use to evaluate the various keyword search techniques. We then describe the metrics we report for our experiments and our experimental setup.

A. Benchmark Overview

Our evaluation benchmark includes the three datasets shown in Table III: MONDIAL [24], IMDb, and Wikipedia. Two datasets (IMDb and Wikipedia) are extracted from popular websites. As shown in Table III, the size of the datasets varies widely: MONDIAL is more than two orders of magnitude smaller than the IMDb dataset, and Wikipedia lies in between. In addition, the schemas and content also differ considerably. MONDIAL has a complex schema with almost 30 relations while the IMDb subset has only 6. Wikipedia also has few relations, but it contains the full text of articles, which emphasizes more complex ranking schemes for results. Our datasets roughly span the range of dataset sizes that have been used in other evaluations (compare Tables I and III).

The benchmark's query workload was constructed by researchers and comprises 50 information needs for each dataset. The query workload does not use real user queries extracted from a search engine log for three reasons. First, Internet search engine logs do not contain queries for datasets not derived from websites. Second, many queries are inherently ambiguous and knowing the user's original information need is essential for accurate relevance assessments. Third, many queries in Internet search engine logs will reflect the limitations of existing search engines—that is, web search engines are not designed to connect disparate pieces of information. Users implicitly adapt to this limitation by submitting few (Nandi and Jagadish [25] report less than 2%) queries that reference multiple database entities.

Table IV provides the statistics of the query workload and relevant results for each dataset. Five IMDb queries are outliers because they include an exact quote from a movie. Omitting these queries reduces the maximum number of terms in any query to 7 and the mean number of terms per query to 2.91. The statistics for our queries are similar to those reported for

TABLE IV
QUERY AND RESULT STATISTICS

	Search log [26]	Synthesized			Results	
Dataset	$\overline{[\![q]\!]}$	$\lvert Q\rvert$	$[\![q]\!]$	$\overline{[\![q]\!]}$	$[\![R]\!]$	$\overline{[\![R]\!]}$
MONDIAL		50	1–5	2.04	1–35	5.90
IMDb	2.71	50	1–26	3.88	1–35	4.32
Wikipedia	2.87	50	1–6	2.66	1–13	3.26
Overall	2.37	150	1–26	2.86	1–35	4.49

Legend
$\lvert Q\rvert$ total number of queries
$[\![q]\!]$ range in number of query terms
$\overline{[\![q]\!]}$ mean number of terms per query
$[\![R]\!]$ range in number of relevant results per query
$\overline{[\![R]\!]}$ mean number of relevant results per query

web queries [16] and our independent analysis of query lengths from a commercial search engine log [26], which suggest that the queries are similar to real user queries. Example queries for each dataset are shown in Table V.

B. Metrics

We use two metrics to measure system performance. The first is execution time, which is the time elapsed from issuing the query until the system terminates. Because there are a large number of potential results for each query, systems typically return only the top-k results where k specifies the desired retrieval depth. Our second metric is response time, which we define as the time elapsed from issuing the query until i results have been returned by the system (where $i \leq k$). Because this definition is not well-defined when fewer than k results are retrieved by a system, we define it for j, where $i < j \leq k$ and i is the number of results retrieved (and k is the desired retrieval depth), as the system's execution time.

System performance should not be measured without also accounting for search effectiveness due to tradeoffs between runtime and the quality of search results. Precision is the ratio of relevant results retrieved to the total number of retrieved results. This metric is important because not every result is actually relevant to the query's underlying information need. Precision @ k (P@k) is the mean precision across multiple queries where the retrieval depth is limited to k results. If fewer than k results are retrieved by a system, we calculate the precision value at the last result. We also use mean average precision (MAP) to measure retrieval effectiveness at greater retrieval depths.

TABLE V
EXAMPLE QUERIES

Dataset	Query	$\lvert R\rvert$	$[\![r]\!]$
MONDIAL	city Granada	1	1
	Nigeria GDP	1	2
	Panama Oman	23	5
IMDb	Tom Hanks	1	1
	Brent Spiner Star Trek	5	3
	Audrey Hepburn 1951	6	3
Wikipedia	1755 Lisbon earthquake	1	1
	dam Lake Mead	4	1,3
	Exxon Valdez oil spill	6	1,3

Legend
$\lvert R\rvert$ number of relevant results
$[\![r]\!]$ size of relevant results (number of tuples)

C. Experimental Setup

Of the search tecniques described in Section III, we reimplemented BANKS, DISCOVER, and DISCOVER-II and obtained implementations of BANKS-II, DPBF, BLINKS, and STAR. We corrected a host of flaws in the specifications of these search techniques and the implementation defects that we discovered. With the exception of DPBF, which is written in C++, all the systems were implemented in Java.

The implementation of BANKS adheres to its original description except that it queries the database dynamically to identify nodes (tuples) that contain query keywords. Our implementation of DISCOVER borrows its successor's query processing techniques. Both DISCOVER and DISCOVER-II are executed with the sparse algorithm, which provides the best performance for queries with AND semantics [14]. BLINKS's block index was created using breadth-first partitioning and contains 50 nodes per block.[6] STAR uses the edge weighting scheme proposed by Ding *et al.* [8] for undirected graphs.

For our experiments, we executed the Java implementations on a Linux machine running Ubuntu 10.04 with dual 1.6 GHz AMD Opteron 242 processors and 3 GB of RAM. We compiled each system using `javac` version 1.6 and ran the implementations with the Java HotSpot 64-bit server VM. DPBF was written in Visual C++ with Windows bindings and was compiled with Microsoft Visual C++ 2008. Due to its Windows bindings, DPBF could not be run on the same machines as the Java implementations. Instead, DPBF was run on a 2.4 GHz Intel Core 2 quad-core processor with 4 GB of RAM running Windows XP. We used PostgreSQL as our database management system.

For all the systems, we limit the size of results to 5 nodes (tuples) and impose a maximum execution time of 1 hour. If the system has not terminated after this time limit, we stop its execution and denote it as a timeout exception. This threshold seems more than adequate for capturing executions that would complete within a reasonable amount of time. Unless otherwise noted, we allow ≈ 2 GB of virtual memory to keep the experimental setups as similar as possible. If a system exhausts the total amount of virtual memory, we mark it as failing due to excessive memory requirements.

V. EXPERIMENTS

Table VI lists the number of queries executed successfully by each system for our datasets and also the number and types of exceptions we encountered. Of interest is the number of queries that either did not complete execution within 1 hour

[6]Memory overhead increases when the index stores more nodes per block, and BLINKS's memory consumption is already considerable.

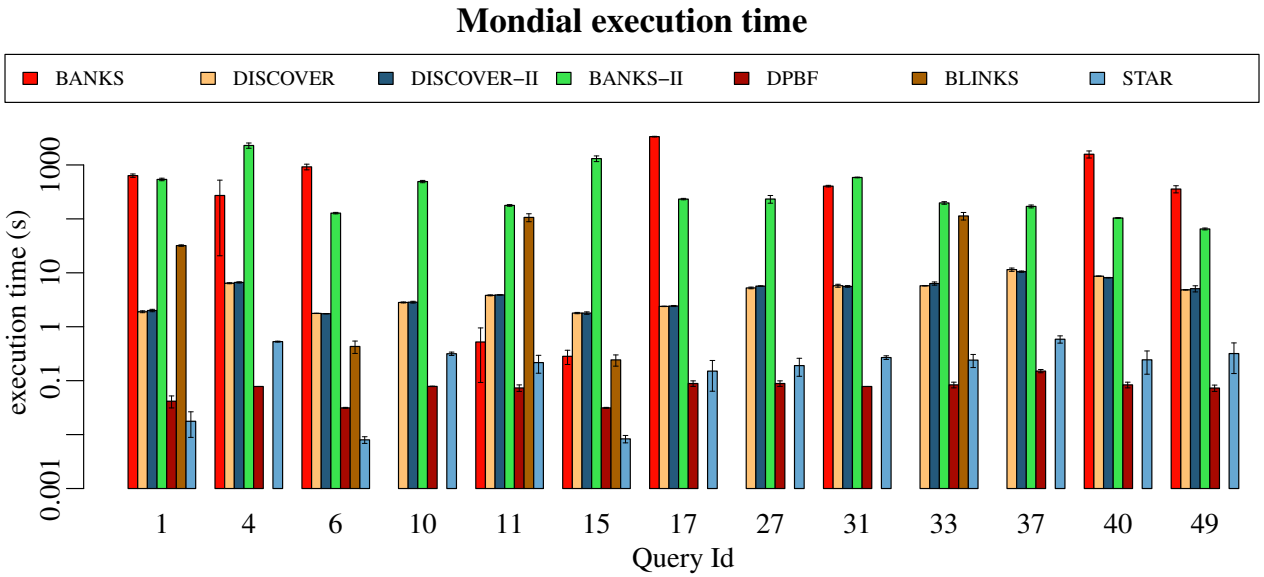

Fig. 2. Execution times for a subset of the MONDIAL queries. Note that the y-axis has a log scale and lower is better. The errors bars provide 95% confidence intervals for the mean. Systems are ordered by publication date and the retrieval depth was 100 results.

TABLE VI
SUMMARIES OF QUERIES COMPLETED AND EXCEPTIONS

System	✓	⏱	🖳	exec. time (s)	MAP
BANKS	30	18	2	1886.1	0.287
DISCOVER	**50**	—	—	5.5	0.640
DISCOVER-II	**50**	—	—	5.6	0.511
BANKS-II	**50**	—	—	282.1	0.736
DPBF	**50**	—	—	**0.1**	0.821
BLINKS	15	—	35	237.7	**0.839**
STAR	**50**	—	—	0.4	0.597

(a) MONDIAL

System	✓	⏱	🖳	exec. time (s)	MAP
BANKS	—	—	50	—	—
DISCOVER	**50**	—	—	220.6	0.097
DISCOVER-II	**50**	—	—	**195.0**	**0.143**
BANKS-II	—	—	50	—	—
DPBF	—	—	50	—	—
BLINKS	—	—	50	—	—
STAR	—	—	50	—	—

(b) IMDb

System	✓	⏱	🖳	exec. time (s)	MAP
BANKS	6	43	1	3174.3	0.000
DISCOVER	**50**	—	—	32.9	0.335
DISCOVER-II	**50**	—	—	31.8	**0.405**
BANKS-II	9	40	1	3202.7	0.098
DPBF	**50**	—	—	**6.5**	0.088
BLINKS	—	—	50	—	—
STAR	—	—	50	—	—

(c) Wikipedia

Legend
✓ Queries completed successfully (out of 50)
⏱ Timeout exceptions (> 1 hour execution time)
🖳 Memory exceptions (exhausted virtual memory)
exec. mean execution time (in seconds) across *all* queries

or exhausted the total amount of virtual memory. Most search techniques complete all the MONDIAL queries with mean execution times ranging from less than a second to several hundred seconds. Results for IMDb and Wikipedia are more troubling. Only DISCOVER and DISCOVER-II complete any IMDb queries, and their mean execution time is several minutes. DPBF joins these two systems by completing all the Wikipedia queries, but all three systems' mean execution times are less than ideal, ranging from 6–30 seconds.

To summarize these results, existing search techniques provide reasonable performance only on the smallest dataset (MONDIAL). Performance degrades significantly when we consider a dataset with hundreds of thousands of tuples (Wikipedia) and becomes unacceptable for millions of tuples (IMDb). The memory consumption for these algorithms is considerably higher than reported, preventing most search techniques from searching IMDb.

In terms of overall search effectiveness (MAP in Table VI), the various search techniques vary widely. Not surprisingly, effectiveness is highest for our smallest dataset. The best systems, DPBF and BLINKS, perform exceedingly well. We note that these scores are considerably higher than those that appear in IR venues (e.g., the Text REtreival Conference (TREC)), which likely reflects the small size of the MONDIAL database. If we accept DISCOVER and DISCOVER-II's trend as representative, we would expect search effectiveness to fall when we consider larger datasets. Unlike performance, which is generally consistent among systems, search effectiveness differs considerably. For examples, DISCOVER-II performs poorly (relative to the other ranking schemes) for MONDIAL, but DISCOVER-II proffers the greatest search effectiveness on IMDb and Wikipedia. Ranking schemes that perform well for MONDIAL queries are not necessarily good for Wikipedia

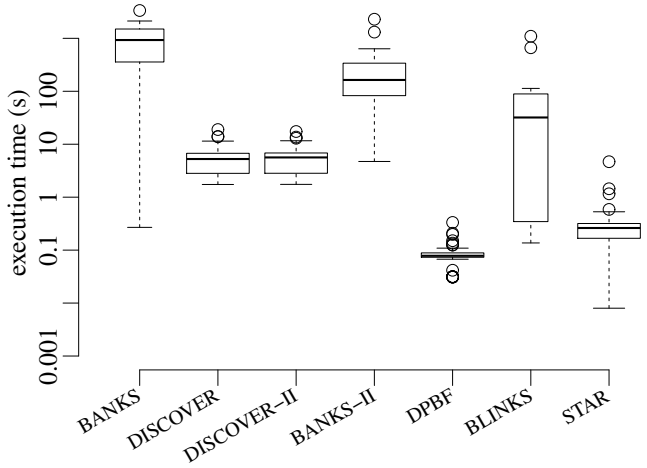

Fig. 3. Boxplots of the execution times of the systems for all the MONDIAL queries' execution times (lower is better). Note that the y-axis has a log scale. Systems are ordered by publication date and the retrieval depth was 100 results.

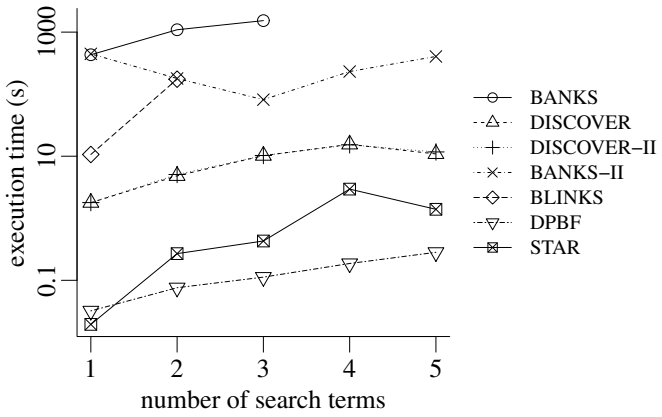

Fig. 4. Mean execution time vs. query length; lower execution times are better. Note that the y-axis has a log scale. The retrieval depth was 100 results.

queries. Hence it is important to balance performance concerns with a consideration of search effectiveness.

Given the few systems that complete the queries for IMDb and Wikipedia, we focus on results for the MONDIAL dataset in the remainder of this section.

A. Execution Time

Figure 2 displays the total execution time for each system on a selection of MONDIAL queries, and Figure 3 shows boxplots of the execution times for all queries on the MONDIAL dataset. Bars are omitted for queries that a system failed to complete (due to either timing out or exhausting memory). As indicated by the error bars in the graph, our execution times are repeatable and consistent. Figures 2 and 3 confirm the performance trends in Table VI but also illustrate the variation in execution time among different queries. In particular, the range in execution time for a search technique can be several orders of magnitude. Most search techniques also have outliers in their execution times; these outliers indicate that the performance of these search heuristics varies considerably due to characteristics of the dataset or queries.

1) Number of search terms: A number of evaluations [8], [14], [15], [17] report mean execution time for queries that contain different numbers of search terms to show that performance remains acceptable even when queries contain more keywords. Figure 4 graphs these values for the different systems. Note that some systems fail to complete some queries, which accounts for the omissions in the graph. As evidenced by the graph, queries that contain more search terms require more time to execute on average than queries than contain fewer search terms. The relative performance among the different systems is unchanged from Figure 2.

These results are similar to those published in previous evaluations. Using Figure 4 as evidence for the efficiency of a particular search technique can be misleading. In Figure 5, we show box plots of the execution times of BANKS and DISCOVER-II to illustrate the range in execution times encountered across the various queries. As evidenced by these graphs, several queries have execution times much higher than the rest. These queries give the system the appearance of unpredictable performance, especially when the query is similar to another one that completes quickly.

For example, the query "Uzbek Asia" for BANKS has an execution time three times greater than the query "Hutu Africa." DISCOVER-II has similar outliers; the query "Panama Oman" requires 3.5 seconds to complete even though the query "Libya Australia" completes in less than half that time. From a user's perspective, these queries would be expected to have similar execution times. These outliers (which are even more pronounced for the other datasets) suggest that simply looking at mean execution time for different numbers of query keywords does not reveal the complete performance profile of these systems. Moreover, existing work does not adequately

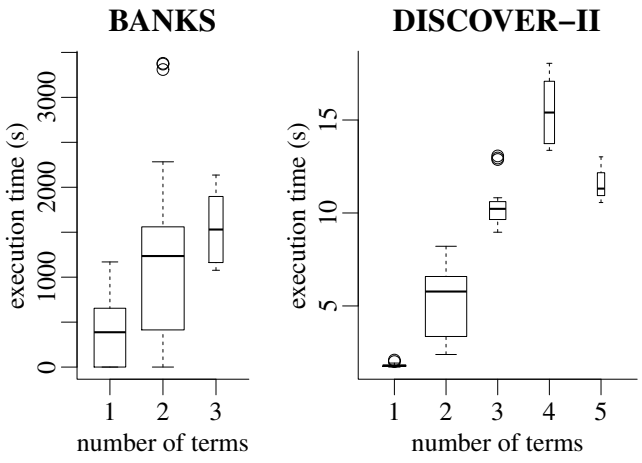

Fig. 5. Box plots of execution times for BANKS (left) and DISCOVER-II (right) at a retrieval depth of 100. The width of the box reflects the number of queries in each sample.

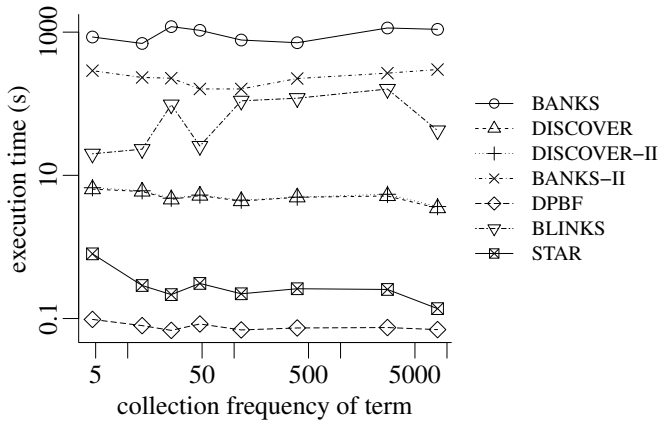

Fig. 6. Execution time vs. mean frequency of a query term in database. Lower execution times are better. Note that the x-axis and y-axis have a log scales. The retrieval depth was 100 results.

explain the existence of these outliers and how to improve the performance of these queries.

2) Collection frequency: In an effort to better understand another factor that is commonly cited as having a performance impact, we consider mean execution time and the frequency of search terms in the database (Figure 6). The results are surprising: execution time appears relatively uncorrelated with the number of tuples containing search terms. This result is counter-intuitive, as one expects the time to increase when more nodes (and all their relationships) must be considered. One possible explanation for this phenomenon is that the search space in the interior of the data graph (i.e., the number of nodes that must be explored when searching) is not correlated with the frequency of the keywords in the database. He *et al.* [13] imply the opposite; we believe additional experiments are warranted as part of future work.

3) Retrieval depth: Table VII considers the scalability of the various search techniques at different retrieval depths—10 and 100 results. Continuing this analysis to higher retrieval depths is not particularly useful given the small size of the MONDIAL database and given that most systems identify all the relevant results within the first 100 results that they return.[7] As evidenced by the table, both the absolute and the percentage slowdown vary widely. However, neither of these values is particularly worrisome: with the exception of BANKS-II, the slowdown is relatively small, and BANKS-II starts with the highest execution time. The negligible slowdown suggests that—with regard to execute time—all the systems will scale easily to larger retrieval depths (e.g., 1000 results). More importantly, only half the systems provide reasonable performance (a few seconds to execute each query) even at a small retrieval depth.

TABLE VII

PERFORMANCE COMPARISON AT DIFFERENT RETRIEVAL DEPTHS

	execution time (s)		slowdown	
System	$k = 10$	$k = 100$	Δ	%
BANKS	1883.8	1886.1	2.3	0.1
DISCOVER	5.1	5.5	0.4	7.8
DISCOVER-II	5.4	5.6	0.2	3.7
BANKS-II	176.5	282.1	105.6	59.8
DPBF	**0.1**	**0.1**	**0.0**	**0.0**
BLINKS	190.3	237.7	47.4	24.9
STAR	0.3	0.4	0.1	33

Legend
k retrieval depth

TABLE VIII

MEAN RESPONSE TIME TO RETRIEVE THE TOP-k QUERY RESULTS

System	resp. (s)	exec. (s)	%	P@1
BANKS	1883.2	1886.1	99.8	0.280
DISCOVER	5.5	5.5	100.0	0.647
DISCOVER-II	5.6	5.6	100.0	0.433
BANKS-II	67.4	282.1	**23.9**	0.700
DPBF	**0.1**	**0.1**	100.0	0.740
BLINKS	122.9	237.7	51.7	**0.853**
STAR	0.4	0.4	100.0	0.720

k = 1

System	resp. (s)	exec. (s)	%	P@10
BANKS	1883.7	1886.1	99.9	0.156
DISCOVER	5.5	5.5	100.0	0.363
DISCOVER-II	5.6	5.6	100.0	0.354
BANKS-II	225.7	282.1	**80.0**	0.422
DPBF	**0.1**	**0.1**	100.0	0.426
BLINKS	193.6	237.7	81.4	0.273
STAR	0.4	0.4	100.0	**0.591**

k = 10

Legend
resp. mean response time (in seconds)
exec. mean total execution time to retrieve 100 results
% percentage of total execution time

B. Response Time

In addition to overall search time, the response time of a keyword search system is of critical importance. Systems that support top-k query processing need not enumerate all possible results before outputting some to the user. Outputting a small number of results (e.g., 10) allows the user to examine the initial results and to refine the query if these results are not satisfactory.

In Table VIII, we show the mean response time to retrieve the first and tenth query result. The table also includes P@k, to show the quality of the results retrieved at that retrieval depth.

Interestingly, the response time for most systems is very close to the total execution time, particularly for $k = 10$. The ratio of response time to the total execution time provided in the table shows that some scoring functions are not good at quickly identifying the best search results. For example, DISCOVER-II identifies the highest ranked search result at the

[7]Investigating the performance of the systems at greater retrieval depths (e.g., on the IMDb dataset) would be ideal, but the failures to complete these queries undermine the value of such experiments.

TABLE IX

COMPARISON OF TOTAL EXECUTION TIME AND RESPONSE TIME

System	exec. (s)	resp. (s)	slowdown Δ (s)	slowdown %
BANKS	1883.8	1883.7	**-0.1**	**-0.0**
DISCOVER	5.1	5.5	0.4	7.8
DISCOVER-II	5.4	5.6	0.2	3.7
BANKS-II	176.5	225.7	49.2	21.8
DPBF	**0.1**	**0.1**	**0.0**	**0.0**
BLINKS	190.3	193.6	3.3	1.7
STAR	0.3	0.4	0.1	33.0

Legend
exec. total execution time when retrieving 10 results
resp. response time to retrieve top-10 of 100 results

same time as it identifies the tenth ranked result because its bound on the possible score of unseen results falls very rapidly after enumerating more than k results. In general, the proximity search systems manage to identify results more incrementally than the schema-based approaches.[8]

Another issue of interest is the overhead required to retrieve additional search results. In other words, how much additional time is spent maintaining enough state to retrieve 100 results instead of just 10? Table IX gives the execution time to retrieve 10 results and the response time to retrieve the first 10 results of 100. With the exception of BANKS-II, the total overhead is minimal—less than a few seconds. In the case of STAR, the percentage slowdown is high, but this value is not significant given that the execution time is so low.

C. Memory consumption

Limiting the graph-based approaches to ≈ 2 GB of virtual memory might unfairly bias our results toward the schema-based approaches. The schema-based systems offload much of their work to the underlying database, which swaps temporary data (e.g., the results of a join) to disk as needed. Hence, DISCOVER and DISCOVER-II might also require a significant amount of memory, and a more fair evaluation would allow the graph-based techniques to page data to disk. To investigate this possibility, we ran all the systems[9] with ≈ 3 GB of physical memory and ≈ 5 GB of virtual memory.[10] Note that once a system consumes the available physical memory, the operating system's virtual memory manager is responsible for paging data to and from disk.

Table X contains the results of this experiment. The overall trends are relatively unchanged from Table VI although BLINKS does complete all the MONDIAL queries with the help of the additional memory. The precipitous drop in execution time suggests that Java's garbage collector was responsible for the majority of BLINKS's execution time, and this overhead was responsible for BLINKS's poor performance. The other graph-based systems do not significantly improve from the additional virtual memory. In most cases, we observed severe thrashing, which merely transformed memory exceptions into timeout exceptions.

Initial Memory Consumption: To better understand the memory utilization of the systems—particularly the overhead of an in-memory data graph, we measured each system's memory footprint immediately prior to executing a query. The results

TABLE X

VIRTUAL MEMORY EXPERIMENTS

System	✓	⏱	🖳	exec. (s)	speedup (%)
BANKS	30	20	—	1817.3	3.7
DISCOVER	**50**	—	—	6.1	-10.9
DISCOVER-II	**50**	—	—	6.3	-12.5
BANKS-II	**50**	—	—	238.7	15.4
BLINKS	**50**	—	—	20.3	91.5
STAR	**50**	—	—	0.3	33

(a) MONDIAL

System	✓	⏱	🖳	exec. (s)	speedup (%)
BANKS	3	40	—	3448.8	—
DISCOVER	**50**	—	—	221.6	-0.5
DISCOVER-II	**50**	—	—	195.0	-0.6
BANKS-II	—	18	—	3607.0	—
BLINKS	—	—	50	—	—
STAR	—	—	50	—	—

(b) IMDb

System	✓	⏱	🖳	exec. (s)	speedup (%)
BANKS	4	46	—	3324.5	-4.7
DISCOVER	**50**	—	—	34.0	-3.3
DISCOVER-II	**50**	—	—	33.1	-4.1
BANKS-II	11	36	—	2909.4	9.2
BLINKS	—	—	50	—	—
STAR	—	—	50	—	—

(c) Wikipedia

Legend
✓ Queries completed successfully (out of 50)
⏱ Timeout exceptions (> 1 hour execution time)
🖳 Memory exceptions (exhausted virtual memory)
exec. mean execution time (in seconds)

TABLE XI

INITIAL MEMORY CONSUMPTION (MONDIAL)

System	Memory (KB) Graph	Total
BANKS [2]	9,200	9,325
DISCOVER [15]	203	330
DISCOVER-II [14]	203	330
BANKS-II [17]	16,751	21,325
DPBF [8]	24,320	
BLINKS [13]	17,502	878,181
STAR [18]	40,593	47,281

[8]DPBF actually identifies results more incrementally than DISCOVER, DISCOVER-II, and STAR but rounding (to account for timing granularity) obscures this result.

[9]DPBF had memory errors when we tried to increase its virtual memory allocation and static-sizing of structures; both are essential for execution on the IMDb dataset.

[10]This was the maximum amount we could consistently allocate on our machines without triggering Linux's out-of-memory killer. We also specified `-Xincgc` to enable Java's incremental garbage collector, which was essential for reasonable performance.

are shown in Table XI. The left column of values gives the size of the graph representation of the database; the right column of values gives the total size of all data structures used by the search techniques (e.g., additional index structures). As evidenced by the table, the schema-based systems consume very little memory, most of which is used to store the database schema. In contrast, the graph-based search techniques require considerably more memory to store their data graph.

When compared to the total amount of virtual memory available, the size of the MONDIAL data graphs are quite small, roughly two orders of magnitude smaller than the size of the heap. Hence, the data graph itself cannot account for the high memory utilization of the systems; instead, the amount of state maintained by the algorithms (not shown by the table) must account for the excessive memory consumption. For example, BANKS's worst-case memory consumption is $O(|V|^2)$ where $|V|$ is the number of vertices in the data graph. It is easy to show that in the worst case BANKS will require in excess of 1 GB of state during a search of the MONDIAL database even if we ignore the overhead of the requisite data structures (e.g., linked lists).

However, we do note that the amount of space required to store a data graph may prevent these systems from searching other, larger datasets. For example, BANKS requires ≈ 1 GB of memory for the data graph of the IMDb subset; this subset is roughly 40 times smaller than the entire database. When coupled with the state it maintains during a search, it is easy to see why BANKS exhausts the available heap space for many queries on this dataset.

D. Threats to Validity

Our results naturally depend upon our evaluation benchmark. By using publicly available datasets and query workloads, we hope to improve the repeatability of these experiments.

In an ideal world, we would reimplement all the techniques that have been proposed to date in the literature to ensure the fairest possible comparison. It is our experience—from implementing multiple systems from scratch—that this task is much more complex than one might initially expect. In general, more recent systems tend to have more complex query processing algorithms, which are more difficult to implement optimally, and few researchers seem willing to share their source code (or binaries) to enable more extensive evaluations. In the following paragraphs, we consider some of the implementation differences among the systems and how these differences might affect our results.

The implementation of DPBF that we obtained was in C++ rather than Java. We do not know how much of DPBF's performance advantage (if any) is due to the implementation language, but we have no evidence that the implementation language plays a significant factor in our results. For example, STAR provides roughly the same performance as DBPF, and DPBF's performance for Wikipedia queries is comparable to DISCOVER and DISCOVER-II when we ignore the length of time required to scan the database's full text indexes instead of storing the inverted index entirely within main memory (as a hash table). Simply rewriting DPBF in Java would not necessarily improve the validity of our experiments because other implementation decisions can also affect results. For example, a compressed graph representation would allow systems to scale better but would hurt the performance of systems that touch more nodes and edges during a traversal [18].

The choice of the graph data structure might significantly impact the proximity search systems. All the Java implementations use the JGraphT library,[11] which is designed to scale to millions of vertices and edges. We found that a lower bound for its memory consumption is $32 \cdot |V| + 56 \cdot |E|$ bytes where $|V|$ is the number of graph vertices and $|E|$ is the number of graph edges. In practice, its memory consumption can be significantly higher because it relies on Java's dynamically-sized collections for storage. Kacholia *et al.* [17] state that the original implementation of BANKS-II requires only $16 \cdot |V| + 8 \cdot |E|$ bytes for its data graph, making it considerably more efficient than the general-purpose graph library used for our evaluation. While an array-based implementation is more compact and can provide better performance, it does have downsides when updating the index. Performance issues that arise when updating the data graph have not been the focus of previous work and has not been empirically evaluated for these systems.

While there are other differences between the experimental setups for different search techniques (e.g., Windows vs. Linux and Intel vs. AMD CPUs), we believe that these differences are minor in the scope of the overall evaluation. For example, DPBF was executed on a quad-core CPU, but the implementation is not multi-threaded so the additional processor cores are not significant. When we executed the Java implementations on the same machine that we used for DPBF (which was possible for sample queries on our smaller datasets), we did not notice a significant difference in execution times.[12]

Our results for MAP (Table VI) differ slightly from previously published results [5]. Theoretically our results should be strictly lower for this metric because our retrieval depth is smaller, but some systems actually improve. The difference is due to the exceptions—after an exception (e.g., timeout), we return any results identified by the system, even if we are uncertain of the results' final ranking. Hence, the uncertain ranking is actually better than the final ranking that the system would enforce if allowed to continue to execute.

VI. RELATED WORK

Existing evaluations of relational keyword search systems are ad hoc with little standardization. Webber [33] summarizes existing evaluations with regards to search effectiveness. Although Coffman and Weaver [5] developed the benchmark that we use in this evaluation, their work does not include any performance evaluation. Baid *et al.* [1] assert that many existing keyword search techniques have unpredictable performance

[11] `http://www.jgrapht.org/`

[12] Windows XP could not consistently allocate > 1 GB of memory for Java's heap space, which necessitated running the Java implementations on Linux machines.

due to unacceptable response times or fail to produce results even after exhausting memory. Our results—particularly the large memory footprint of the systems—confirm this claim.

A number of relational keyword search systems have been published beyond those included in our evaluation. Chen *et al.* [4] and Chaudhuri and Das [3] both presented tutorials on keyword search in databases. Yu *et al.* [35] provides an excellent overview of relational keyword search techniques.

Liu *et al.* [21] and SPARK [22] both propose modified scoring functions for schema-based keyword search. SPARK also introduces a skyline sweep algorithm to minimize the total number of database probes during a search. Qin *et al.* [27] further this efficient query processing by exploring semi-joins. Baid *et al.* [1] suggest terminating the search after a predetermined period of time and allowing the user to guide further exploration of the search space.

In the area of graph-based search techniques, EASE [20] indexes all r-radius Steiner graphs that might form results for a keyword query. Golenberg *et al.* [12] provide an algorithm that enumerates results in approximate order by height with polynomial delay. Dalvi *et al.* [6] consider keyword search on graphs that cannot fit within main memory. CSTree [19] provides alternative semantics—the compact Steiner tree—to answer search queries more efficiently.

In general, the evaluations of these systems do not investigate important issues related to performance (e.g., handling data graphs that do not fit within main memory). Many evaluations are also contradictory, for the reported performance of each system varies greatly between different evaluations. Our experimental results question the validity of many previous evaluations, and we believe our benchmark is more robust and realistic with regards to the retrieval tasks than the workloads used in other evaluations. Furthermore, because our evaluation benchmark is available for other researchers to use, we expect our results to be repeatable.

VII. Conclusion and Future Work

Unlike many of the evaluations reported in the literature, ours is designed to investigate not the underlying algorithms but the overall, end-to-end performance of these retrieval systems. Hence, we favor a realistic query workload instead of a larger workload with queries that are unlikely to be representative (e.g., queries created by randomly selecting terms from the dataset).

Overall, the performance of existing relational keyword search systems is somewhat disappointing, particularly with regard to the number of queries completed successfully in our query workload (see Table VI). Given previously published results (Table II), we were especially surprised by the number of timeout and memory exceptions that we witnessed. Because our larger execution times might only reflect our choice to use larger datasets, we focus on two concerns that we have related to memory utilization.

First, no system admits to having a large memory requirement. In fact, memory consumption during a search has not been the focus of any previous evaluation. To the best of our knowledge, only two papers [6], [18] have been published in the literature that make allowances for a data graph that does not fit entirely within main memory. Given that most existing evaluations focus on performance, handling large data graphs (i.e., those that do not fit within main memory) should be well-studied. Relying on virtual memory and paging is no panacea to this problem because the operating system's virtual memory manager will induce much more I/O than algorithms designed for large graphs [6] as evidenced by the number of timeouts when we allowed these systems to page data to disk. Kasneci *et al.* [18] show that storing the graph on disk can also be extremely expensive for algorithms that touch a large number of nodes and edges.

Second, our results seriously question the scalability of these search techniques. MONDIAL is a small dataset (see Table III) that contains fewer than 20K tuples. While its schema is complex, we were not expecting failures due to memory consumption. Although we executed our experiments on machines that have a small amount of memory by today's standards, scalability remains a significant concern. If 2 GB of memory is not sufficient for MONDIAL, searching our IMDb subset will require $\gtrapprox 200$ GB of memory and searching the entire IMDb database would require $\gtrapprox 5$ TB. Without additional research into high-performance algorithms that maintain a small memory footprint, these systems will be unable to search even moderately-sized databases and will never to suitable for large databases like social networks or medical health records.

Further research is unquestionably necessary to investigate the myriad of experimental design decisions that have a significant impact on the evaluation of relational keyword search systems. For example, our results indicate that existing systems would be unable to search the entire IMDb database, which underscores the need for a progression of datasets that will allow researchers to make progress toward this objective. Creating a subset of the original dataset is common, but we are not aware of any work that identifies how to determine if a subset is representative of the original dataset. In addition, different research groups often have different schemas for the same data (e.g., IMDb), but the effect of different database schemas on experimental results has also not been studied.

Our results should serve as a challenge to this community because little previous work has acknowledged these challenges. Moving forward, we must address several issues. First, we must design algorithms, data structures, and implementations that recognize that storing a complete graph representation of a database within main memory is infeasible for large graphs. Instead, we should develop techniques that efficiently manage their memory utilization, swapping data to and from disk as necessary. Such techniques are unlikely to have performance characteristics that are similar to existing systems but must be used if relational keyword search systems are to scale to large datasets (e.g., hundreds of millions of tuples). Second, evaluations should reuse datasets and query workloads to provide greater consistency of results, for even our results vary widely depending on which dataset is considered. Having

the community coalesce behind reusable test collections would facilitate better comparison among systems and improve their overall evaluation [33]. Third, the practice of researchers reimplementing systems may account for some evaluation discrepancies. Making the original source code (or a binary distribution that accepts a database URL and query as input) available to other researchers would be ideal and greatly reduce the likelihood that observed differences are implementation artifacts.

VIII. Acknowledgments

Michelle McDaniel contributed to preliminary research and provided feedback on drafts of this paper. We thank Ding *et al.*, He *et al.*, and Kasneci *et al.* for providing us with system implementations that we used for our experiments.

References

[1] A. Baid, I. Rae, J. Li, A. Doan, and J. Naughton, "Toward Scalable Keyword Search over Relational Data," *Proceedings of the VLDB Endowment*, vol. 3, no. 1, pp. 140–149, 2010.

[2] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan, "Keyword Searching and Browsing in Databases using BANKS," in *Proceedings of the 18th International Conference on Data Engineering*, ser. ICDE '02, February 2002, pp. 431–440.

[3] S. Chaudhuri and G. Das, "Keyword Querying and Ranking in Databases," *Proceedings of the VLDB Endowment*, vol. 2, pp. 1658–1659, August 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1687553.1687622

[4] Y. Chen, W. Wang, Z. Liu, and X. Lin, "Keyword Search on Structured and Semi-Structured Data," in *Proceedings of the 35th SIGMOD International Conference on Management of Data*, ser. SIGMOD '09, June 2009, pp. 1005–1010.

[5] J. Coffman and A. C. Weaver, "A Framework for Evaluating Database Keyword Search Strategies," in *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, ser. CIKM '10, October 2010, pp. 729–738. [Online]. Available: http://doi.acm.org/10.1145/1871437.1871531

[6] B. B. Dalvi, M. Kshirsagar, and S. Sudarshan, "Keyword Search on External Memory Data Graphs," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 1189–1204, 2008.

[7] E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[8] 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: Proceedings of the 23rd International Conference on Data Engineering*, April 2007, pp. 836–845.

[9] S. E. Dreyfus and R. A. Wagner, "The Steiner Problem in Graphs," *Networks*, vol. 1, no. 3, pp. 195–207, 1971. [Online]. Available: http://dx.doi.org/10.1002/net.3230010302

[10] D. Fallows, "Search Engine Use," Pew Internet and American Life Project, Tech. Rep., August 2008, http://www.pewinternet.org/Reports/2008/Search-Engine-Use.aspx.

[11] "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.

[12] K. Golenberg, B. Kimelfeld, and Y. Sagiv, "Keyword Proximity Search in Complex Data Graphs," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '08, June 2008, pp. 927–940.

[13] H. He, H. Wang, J. Yang, and P. S. Yu, "BLINKS: Ranked Keyword Searches on Graphs," in *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '07, June 2007, pp. 305–316.

[14] V. Hristidis, L. Gravano, and Y. Papakonstantinou, "Efficient IR-style Keyword Search over Relational Databases," in *Proceedings of the 29th International Conference on Very Large Data Bases*, ser. VLDB '03, September 2003, pp. 850–861.

[15] V. Hristidis and Y. Papakonstantinou, "DISCOVER: Keyword Search in Relational Databases," in *Proceedings of the 29th International Conference on Very Large Data Bases*, ser. VLDB '02. VLDB Endowment, August 2002, pp. 670–681.

[16] B. J. Jansen and A. Spink, "How are we searching the World Wide Web? A comparison of nine search engine transaction logs," *Information Processing and Management*, vol. 42, no. 1, pp. 248–263, 2006.

[17] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar, "Bidirectional Expansion For Keyword Search on Graph Databases," in *Proceedings of the 31st International Conference on Very Large Data Bases*, ser. VLDB '05, August 2005, pp. 505–516.

[18] G. Kasneci, M. Ramanath, M. Sozio, F. M. Suchanek, and G. Weikum, "STAR: Steiner-Tree Approximation in Relationship Graphs," in *Proceedings of the 25th International Conference on Data Engineering*, ser. ICDE '09, March 2009, pp. 868–879.

[19] G. Li, J. Feng, X. Zhou, and J. Wang, "Providing built-in keyword search capabilities in RDBMS," *The VLDB Journal*, vol. 20, pp. 1–19, February 2011. [Online]. Available: http://dx.doi.org/10.1007/s00778-010-0188-4

[20] 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 *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '08, June 2008, pp. 903–914.

[21] F. Liu, C. Yu, W. Meng, and A. Chowdhury, "Effective Keyword Search in Relational Databases," in *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '06, June 2006, pp. 563–574.

[22] Y. Luo, X. Lin, W. Wang, and X. Zhou, "SPARK: Top-k Keyword Query in Relational Databases," in *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '07, June 2007, pp. 115–126.

[23] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. New York, NY: Cambridge University Press, 2008.

[24] W. May, "Information Extraction and Integration with FLORID: The MONDIAL Case Study," Universität Freiburg, Institut für Informatik, Tech. Rep. 131, 1999, available from http://dbis.informatik.uni-goettingen.de/Mondial.

[25] A. Nandi and H. V. Jagadish, "Qunits: queried units for database search," in *CIDR '09: Proceedings of the 4th Biennial Conference on Innovative Data Systems Research*, January 2009.

[26] G. Pass, A. Chowdhury, and C. Torgeson, "A Picture of Search," in *InfoScale '06: Proceedings of the 1st International Conference on Scalable Information Systems*, May 2006.

[27] L. Qin, J. X. Yu, and L. Chang, "Keyword Search in Databases: The Power of RDBMS," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '09, June 2009, pp. 681–694.

[28] L. Qin, J. Yu, L. Chang, and Y. Tao, "Querying Communities in Relational Databases," in *Proceedings of the 25th International Conference on Data Engineering*, ser. ICDE '09, March 2009, pp. 724–735.

[29] G. Reich and P. Widmayer, "Beyond Steiner's problem: A VLSI oriented generalization," in *Graph-Theoretic Concepts in Computer Science*, ser. Lecture Notes in Computer Science, M. Nagl, Ed. Springer, 1990, vol. 411, pp. 196–210. [Online]. Available: http://dx.doi.org/10.1007/3-540-52292-1_14

[30] A. Singhal, "Modern Information Retrieval: A Brief Overview," *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 24, no. 4, pp. 35–43, December 2001.

[31] Q. Su and J. Widom, "Indexing Relational Database Content Offline for Efficient Keyword-Based Search," in *Proceedings of the 9th International Database Engineering & Application Symposium*, ser. IDEAS '05, July 2005, pp. 297–306.

[32] E. M. Voorhees, "The Philosophy of Information Retrieval Evaluation," in *CLEF '01: Revised Papers from the Second Workshop of the Cross-Language Evaluation Forum on Evaluation of Cross-Language Information Retrieval Systems*. Springer-Verlag, 2002, pp. 355–370.

[33] W. Webber, "Evaluating the Effectiveness of Keyword Search," *IEEE Data Engineering Bulletin*, vol. 33, no. 1, pp. 54–59, 2010.

[34] W. Webber, A. Moffat, and J. Zobel, "Statistical Power in Retrieval Experimentation," in *CIKM '08: Proceeding of the 17th ACM International Conference on Information and Knowledge Management*, 2008, pp. 571–580.

[35] J. X. Yu, L. Qin, and L. Chang, *Keyword Search in Databases*, 1st ed. Morgan and Claypool Publishers, 2010.