INTERNATIONAL JOURNAL OF RESEARCH
In Advanced Engineering Technologies

Volume 5, Issue 1 SEP 2015

AN EMPIRICAL PERFORMANCE EVALUATION OF RELATIONAL KEYWORD SEARCH

[1] Shendge Shankar, PG Scholar in C S,
[2]K.Praveen, Assistant Professor,
[3] CH.Poornima, Associate Professor and H O D (C S E),
[1] shendgeshanker@gmail.com. [2] kesana.praveen@gmail.com, [3] poornimachaparala@yahoo.com.
[1, 2, 3] Aurobindo Institute of Engineering and Technology, Hyderabad, R.R Dist, Telangana –India

Abstract: Now a day's for extending the keyword search to relational data set has been an area of research within the database and Information Retrieval. There is no standardization in the process if information retrieval, which will not clearly show the actual result also it displays keyword search without ranking and Execution time is more in existing system. We propose a system for; performance evaluation of relational keyword search systems. In the propose system combine schema-based and graph-based approaches and propose a Relational Keyword Search System to overcome the mentioned disadvantages of existing systems and manage the information and user access the information very efficiently. The objective of this technique is to manage Information, Database and Information Retrieval systems involved independently and developed their own unique systems to allow users to access information. We also explore the relationship between execution time and factors. The proposed search technique will overcome the poor performance for datasets exceeding tens of thousands of vertices.

Keywords: Keyword Search, Empirical Performance, Relational Data, Relational Keyword.

I. INTRODUCTION

With the growing use of internet more and more people search the data on internet. Advents of Internet, it became possible to store a large amount of information. Several techniques are used to Information Retrieval (IR). Keyword search is one of the techniques use for the same. Keyword search is possible on both structure and semi-structure databases, also it possible on graph structure which combines relational, HTML and XML data. In relational databases the keyword search is used to find the tuples in by giving queries. Keyword search use number of techniques and algorithm for storing and retrieving data, less accuracy, does not giving a correct answer, require large time for searching and large amount of storage space for data storage. We propose a system to overcome the disadvantages which discussed for efficient keyword search. Data mining or information retrieval is the process to retrieve data from dataset and transform it to user in understandable form, so user easily gets that information.

One important advantages of keyword search is user does not require a proper knowledge of database queries. User easily inserts a keyword for searching and gets a result related to that keyword. Keyword search on relational databases find the answer of the tuples which are connected to database keys like primary key and foreign keys. So we also present which comparative techniques used for keyword search like DISCOVER, BANKS, BLINKS, EASE, and SPARK. One important thing is that any existing techniques experimental result indicate that existing search techniques are not capable of real world information retrieval and data mining task. 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:

I. We conduct an independent, empirical performance evaluation of 7 relational keyword search techniques, which doubles the number of comparisons as previous work.

9. 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.

K. 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.

Λ. Our work is the first to combine performance and

for information retrieval on real world databases and also 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, Related Work Section III System Design Section IV describes our Project Description and Finally Section V provides our conclusions.

II. RELATED WORK

The 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. Thus, also exploring the relationship between execution time and factors varied in previous evaluations; the analysis indicates that these factors have relatively little impact on performance. In summary, the 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.

A. Dynamic Programming Algorithm

Dynamic programming algorithms are used for optimization (for example, finding the shortest path between two points, or the fastest way to multiply many matrices). A dynamic programming algorithm will examine all possible ways to solve the problem and will pick the best solution. Therefore, it can roughly think of dynamic programming as an intelligent, brute-force method that enables us to go through all possible solutions to pick the best one. If the scope of the problem is such that going through all possible solutions is possible and fast enough, dynamic programming guarantees finding the optimal solution. The alternatives are many, such as using a greedy algorithm, which picks the best possible choice "at any possible branch in the road". While a greedy algorithm does not guarantee the optimal solution, it is faster. Fortunately, some greedy algorithms (such as minimum spanning trees) are proven to lead to the optimal solution.

B. Pseudo Polynomial-Time Algorithm

A pseudo-polynomial-time algorithm is used to display the exponential behavior only when confronted with instances containing exponentially large numbers of clusters, which might be rare for the application, are interested in. If so, this type of algorithm might serve the purposes almost as well as a polynomial time algorithm. This algorithm helps to improve the time taken for searching the data from large set of cluster based on the respective keyword and produce the results quickly within a fraction of seconds with the help of Steiner Tree Problem. The Steiner tree problem is superficially similar to the minimum spanning tree problem: search effectiveness in the evaluation of such a large. resource from that clusters and produce the results within a minimum of time.

C. Bidirectional Search Algorithm

Bidirectional search algorithm is a searching algorithm that finds a shortest path from an initial highest point to a goal highest point in a directed way. It runs two simultaneous searches: one forward from the initial state and one backward from the goal, stopping when the two meet in the middle. The reason for this approach is that in many cases it is faster: for instance, in a simplified model of search problem complexity in which both searches expand a tree with branching factor b, and the distance from start to goal is d, each of the two searches has complexity O(bd/2) (in Big O notation), and the sum of these two search times is much less than the O(bd) complexity that would result from a single search from the beginning to the goal.

D. Sparse Algorithm

The Sparse algorithm discovers the files by its keyword those are presented into the content of the file and executes it in a fraction of second for the user.

E. Skyline Sweep Algorithm

The Skyline Sweep Algorithm is used to minimize the total number of database probes during a search. Searching keywords in databases is complex task than search in files. Information Retrieval (IR) process search keywords from text files and it is very important that queering keyword to the relational databases. Generally to retrieve data from relational database Structure Query Language (SQL) can be used to find relevant records from the database. There is natural demand for relation database to support effective and efficient IR Style Keyword queries. This algorithm clearly supporting effective and efficient top-k keyword search in relational databases also describe the frame word which takes keywords and K as inputs and generates top-k relevant records. The results of implemented system with Skyline Sweeping Algorithm show that it is one effective and efficient style of keyword search.

F. Breadth-First Algorithm

The BFS begins search at a root node and inspects all the neighboring nodes. Then for each of those neighbor nodes in turn, it inspects their neighbor nodes which were unvisited, and so on. In the approach it results the content wise usage of data or results the searching count based on the searched content.

III. SYSTEM DESIGN

A. System Architecture

In proposed system, 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 which gives set of clusters and searching the required

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. 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 main memory is limited.

Search techniques must manage their memory utilization efficiently, swapping data to and from disk as necessary. Such implementations are unlikely to have performance characteristics that are similar to existing approaches but

must be used if relational keyword search systems are to scale to large data sets (e.g., hundreds of millions of tuples). Second, evaluations should reuse data sets and query workloads to provide greater consistency of results, for even our results vary widely depending on which data set is considered. Fortunately, our evaluation benchmark is beginning to gain traction in this area as evidenced by others' adoption of it for their evaluations. Third, the practice of researchers implementing search techniques 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 greatly reduce the likelihood that observed differences are implementation artifacts.

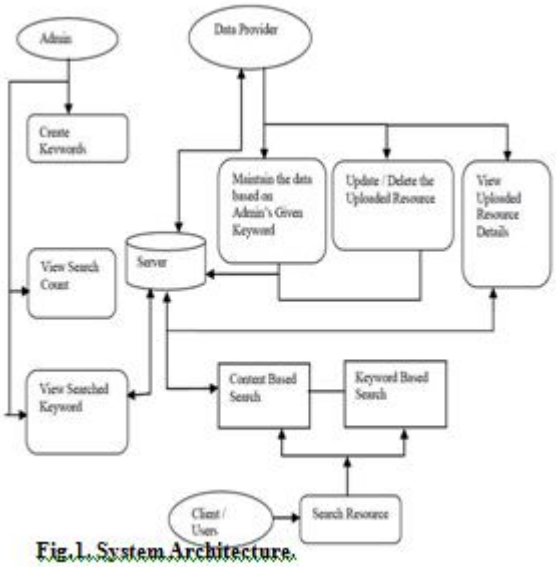

Fig.1. System Architecture.

B. Modules

- **Owner Resource Regulation Portal:** Owner resource regulation portal is the mode of generating resources. The resources are collected and stored into the database after the verification from the admin.
- **Resource Manipulation:** Resources are entered into the database through the resource regulation port and they are manipulated based on the particular keyword generated for that resource.
- **Client/Owner Authentication:** Authentication is the process of assuring that the particular resource is generated by the customer. It helps in reducing false data into database.
- **Header Keyword Requisition:** When you search, browse, create non-catalog request, or perform any action that initiates the creation of a requisition, your user preference is validated. In case of error, the preferences window is opened and you can click on the View Errors link to see the errors. You need to fix your preferences before you can proceed with creating

IV. EXPERIMENTS

Table I 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 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 completes 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.

Table I: Query and Result Statistics

Search log [26]		Synthesized			Results	
Dataset	$\overline{[q]}$	$\|Q\|$	$[q]$	$\overline{[q]}$	$[R]$	$\overline{[R]}$
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
$|Q|$ 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

Table II: Example Queries

Dataset	Query	$\|R\|$	$[r]$
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
$|R|$ number of relevant results
$[r]$ size of relevant results (number of tuples)

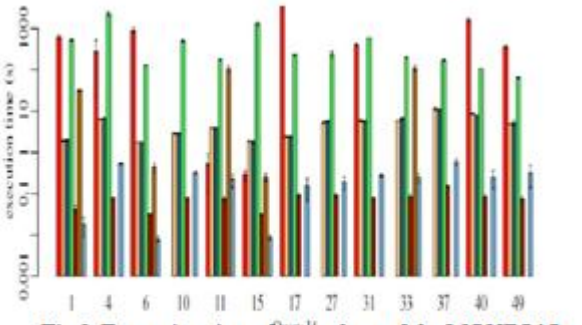

Fig.2. Execution times for a subset of the MONDIAL queries. Note that the y-axis has a log scale and lower is better. The errors bars provide 95% confidence intervals for the mean. Systems are ordered by publication date and the retrieval depth was 100 results.

Table III: 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

In terms of overall search effectiveness (MAP in Table III), 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 RE trivial 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 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

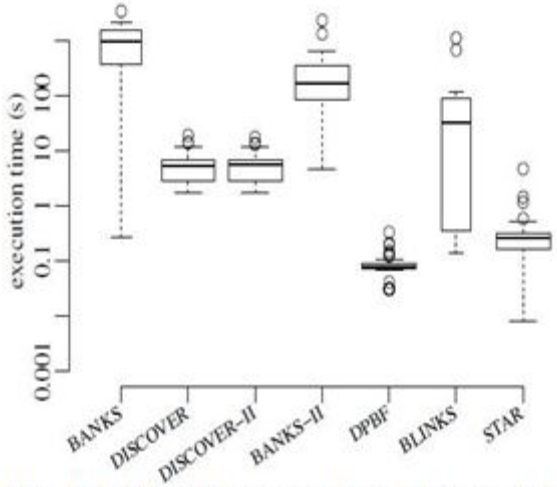

Fig.3. Box plots of the execution times of the systems for all the MONDIAL queries' execution times (lower is better). Note that the y-axis has a log scale. Systems are ordered by publication date and the retrieval depth was 100 results.

A. Execution Time

Fig. 2 displays the total execution time for each system on a selection of MONDIAL queries, and Fig. 3 shows box plots 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. Figs 2 and 3 confirm the performance trends in Table III 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, 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. Fig. 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 Fig.2. These results are similar to those published in previous evaluations. Using Fig.4 as evidence for the efficiency of a particular search technique can be misleading. In Fig.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

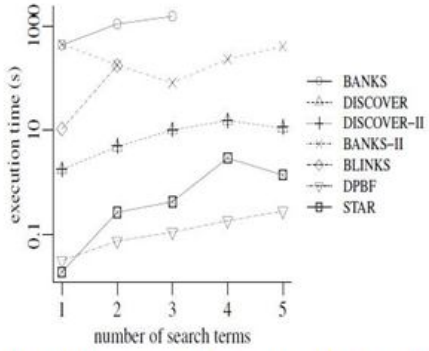

Fig.4. Mean execution time vs. query length; lower execution times are better. Note that the y-axis has a log scale. The retrieval depth was 100 results.

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

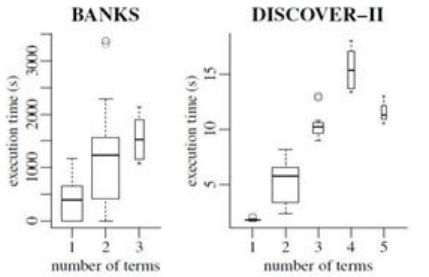

Fig.5. Box plots of execution times for BANKS (left) and DISCOVER-II (right) at a retrieval depth of 100 The width of the box reflects the number of queries in each sample.

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 (Fig. 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 implies the opposite; we believe additional experiments are warranted as part of future work

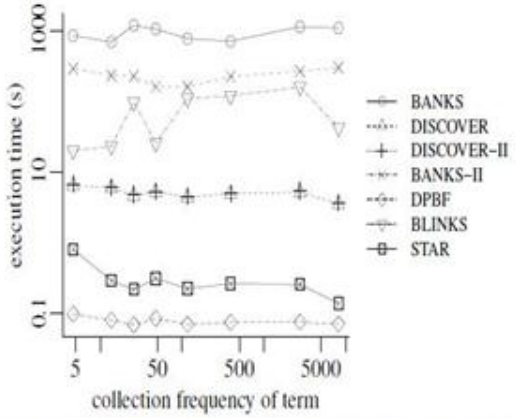

Fig.6. Execution times vs. mean frequency of a query term in database Lower execution times are better. Note that the x-axis and y-axis have a log scales. The retrieval depth was 100 results.

3. Retrieval depth

Table IV 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. 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.

Table IV. 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 V: 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

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.

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.

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 V, 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 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. 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 VII 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

Table VI: 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

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 with ≈

3 GB of physical memory and 5 GB of virtual memory. 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 VI contains the results of this experiment.

The overall trends are relatively unchanged from Table III 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

Table VII: 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)

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 are shown in Table VIII. 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.

Table VIII: 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

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 re-implement 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.

The choice of the graph data structure might significantly impact the proximity search systems. All the Java implementations use the JGraphT library, which is designed to scale to millions of vertices and edges. We found that a lower bound for its memory consumption is 32.|V| + 56 .|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 the original implementation of BANKS-II requires only |V| + 8 .|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 have 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. Our results for MAP (Table III) differ slightly from previously published results. 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.

V. CONCLUSION

Overall we will study all the existing techniques which is available in market. Each system has some advantages and some issues. We compare all the techniques and checked the performance. So finally conclude that any existing system cannot fulfill all the requirement of keyword query search. They require more space and time; also some techniques are limited for particular dataset. The Proposed technique is satisfying number of requirement of keyword query search using different algorithms. The performance of keyword search is also the better to compare other and it shows the actual result rather than tentative. It also shows the ranking of keyword and not requires the knowledge of database queries. Compare to existing algorithm it is a fast process. As a future work we can search the techniques which are useful for all the datasets, means only single technique can be used for MONDIAL, IMDb etc. Further research is necessary to investigate the experimental design decisions that have a significant impact on the evaluation of relational keyword search system.

VI. REFERENCES

[1] Joel Coffman, Alfred C. Weaver, ―An Empirical Performance Evaluation of Relational Keyword Search

Systems‖, IEEE Transactions on Knowledge and Data

Engineering, (Volume: 26, Issue: 1) Year: 2014.
[2] D. Fallows, ―Search Engine Use,‖ technical report, Pew

Internet and Am. Life Project, http://www.pewinter net.org/Reports/2008/Search-Engine-Use.aspx. Aug. 2008.

[3] Com Score, ―Global Search Market Grows 46 Percent in

2009,‖ http://www.comscore.com/Press_Events/Press_ Releases/2010/1/Global_Search_Market_Grows_46_%_in_2 009, Jan. 2010.

[4] J. Coffman and A.C. Weaver, ―A Framework for Evaluating Database Keyword Search Strategies,‖ Proc. 19th ACM Int'l Conf. Information and Knowledge Management

(CIKM '10), pp. 729-738, Oct. 2010.

[5] Y. Chen, W. Wang, Z. Liu, and X. Lin, ―Keyword Search on Structured and Semi-Structured Data,‖ Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '09), pp. 1005-1010, June 2009.

[6] W. Webber, ―Evaluating the Effectiveness of Keyword Search,‖ IEEE Data Eng. Bull., vol. 33, no. 1, pp. 54-59, Mar. 2010.

[7] A. Baid, I. Rae, J. Li, A. Doan, and J. Naughton, ―Toward Scalable Keyword Search over Relational Data,‖ Proc. VLDB Endowment, vol. 3, no. 1, pp. 140-149, 2010.

[8] Q. Su and J. Widom, ―Indexing Relational Database

Content Offline for Efficient Keyword-Based Search,‖ Proc. Ninth Int'l Database Eng. and Application Symp. (IDEAS '05), pp. 297-306, July 2005.

[9] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R.

Desai, and H. Karambelkar, ―Bidirectional Expansion For Keyword Search on Graph Databases,‖ Proc. 31st Int'l Conf. Very Large Data Bases (VLDB '05), pp. 505-516, Aug. 2005.

[10] H. He, H. Wang, J. Yang, and P.S. Yu, ―BLINKS: Ranked Keyword Searches on Graphs,‖ Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '07), pp. 305-316, June 2007.

[11] G. Kasneci, M. Ramanath, M. Sozio, F.M. Suchanek, and G. Weikum, ―STAR: Steiner-Tree Approximation in Relationship Graphs,‖ Proc. Int'l Conf. Data Eng. (ICDE '09), pp. 868-879, Mar. 2009.

[12] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan, ―Keyword Searching and Browsing in Databases Using BANKS,‖ Proc. 18th Int'l Conf. Data Eng. (ICDE '02), pp. 431-440, Feb. 2002