ISSN:0975-9646

Muthe V Satya Nagaraju et al, / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 2 (6) , 2011, 2541-2547.

Dynamic Approch to Retrive the Web pages

MUTHE V SATYA NAGARAJU[*], U. NANAJI[#]

[*] Department of CSE, Saint Theresa Institute of Engg. & Technology, Garividi, Vizayanagaram, (A.P.), India

[#] HOD, Prof., Department of CSE, Saint Theresa Institute of Engg. & Technology, Garividi, Vizayanagaram, (A.P.), India

***Abstract*—** Dynamic authority-based keyword search algorithms,such as ObjectRank and personalized PageRank, leverage semantic link information to provide high quality, high recall search in databases, and the Web. Conceptually, these algorithms require a querytime PageRank-style iterative computation over the full graph. This computation is too expensive for large graphs, and not feasible at query time. Alternatively, building an index of precomputed results for some or all keywords involves very expensive preprocessing.We introduce BinRank, a system that approximates ObjectRank results by utilizing a hybrid approach inspired by materialized views in traditional query processing. We materialize a number of relatively small subsets of the data graph in such a way that any keyword query can be answered by running ObjectRank on only one of the subgraphs. BinRank generates the subgraphs by partitioning all the terms in the corpus based on their co-occurrence, executing ObjectRank for each partition using the terms to generate a set of random walk starting points, and keeping only those objects that receive non-negligible scores. The intuition is that a subgraph that contains all objects and links relevant to a set of related terms should have all the information needed to rank objects with respect to one of these terms. We demonstrate that BinRank can achieve subsecond query execution time on the English Wikipedia data set, while producing high-quality search results that closely approximate the results of ObjectRank on the original graph. The Wikipedia link graph contains about 108 edges, which is at least two orders of magnitude larger than what prior state of the art dynamic authority-based search systems have been able to demonstrate. Our experimental evaluation investigates the trade-off between query execution time, quality of the results, and storage requirements of BinRank.

INTRODUCTION

The PageRank algorithm [1] utilizes the Web graph link structure to assign global importance to Web pages. It works by modeling the behavior of a "random Web surfer"who starts at a random Web page and follows outgoing links with uniform probability. The PageRank score is independent of a keyword query. Recently, dynamic versions of the PageRank algorithm have become popular. They are characterized by a query-specific choice of the random walk starting points. In particular, two algorithms have got a lot of attention: Personalized PageRank (PPR) for Web graph data sets [2],[3],[4],[5] and ObjectRank for graph-modeled databases [6],[7],[8],[9],[10]. PPR is a modification of PageRank that performs search personalized on a preference set that contains Web pages
that a user likes. For a given preference set, PPR performs a very expensive fixpoint iterative computation over the entire Web graph, while it generates personalized search results. Therefore, the issue of scalability of PPR has attracted a lot of attention. ObjectRank extends (personalized) PageRank to perform keyword search in databases. ObjectRank [6] uses a query term posting list as a set of random walk starting points and conducts the walk on the instance graph of the database. The resulting system is well suited for "high recall" search, which exploits different semantic connection paths between objects in highly heterogeneous data sets. ObjectRank has successfully been applied to databases that have social networking components, such as bibliographic data and collaborative product design. However, ObjectRank suffers from the same scalability issues as personalized PageRank, as it requires multiple iterations over all nodes and links of the entire database graph. The original ObjectRank system has two modes: online and offline. The online mode runs the ranking algorithm once the query is received, which takes too longon large

graphs. For example, on a graph of articles of English Wikipedia1 with 3.2 million nodes and 109 million links, even a fully optimized in-memory implementation of ObjectRank takes 20-50 seconds to run. In the offline mode, Object Rank recomputed top-k results for a query workload in advance. This precomputation is very expensive and requires a lot of storage space for precomputed results. Moreover, this approach is not feasible for all terms outside the query workload that a user may search for, i.e., for all terms in the data set dictionary. For example, on the same Wikipedia data set, the full dictionary precomputation would take about a CPU-year. In this paper,weintroduce a BinRank system that employs a hybrid approach where query time can be traded off for preprocessing time and storage. BinRank closely approximates ObjectRank scores by running the same ObjectRank algorithm on a small subgraph, instead of the full data graph. The subgraphs are precomputed offline. The precomputation can be parallelized with linear scalability. For example, on the full Wikipedia data set, BinRank can answer any query in less
than 1 second, by precomputing about a thousand subgraphs, which takes only about 12 hours on a single CPU. BinRank query execution easily scales to large clusters by distributing the subgraphs between the nodes of the cluster. This way, more subgraphs can be kept in RAM, thus decreasing the average query execution time. Since the distribution of the query terms in a dictionary is usually very uneven, the throughput of the system is greatly improved by keeping duplicates of popular subgraphs on multiple nodes of the cluster. The query term is routed to the least busy node that has the corresponding subgraph. There are two dimensions to the subgraph precomputation problem:

1) how many subgraphs to precompute and
2) how to construct each subgraph that is used for approximation.

The intuition behind our approach is thata subgraph that contains all objects and links relevant to aset of related terms should have all the information needed to rank objects w.r.t. one of these terms. For 1), we group all terms into a small number (around 1,000 in case of Wikipedia) of "bins" of terms based on their co-occurrence in the entire data set. For 2), we execute ObjectRank for each bin using the terms in the bins as random walk starting points and keep only those nodes that receive non-negligible scores. Our experimental evaluation highlights the tuning of the system needed to balance the query performance with size and number of the precomputed subgraphs. Intuitively, query performance is highly correlated to the size of the subgraph, which, in turn, is highly correlated with the number of documents in the bin. Thus, normally, it is sufficient to create bins with a certain size limit to achieve a specific target running time. However, there is some variability in the process and some bins may still result in unusually large subgraphs and slow queries. To address this, we employ an adaptive iterative process that further splits the problematic subgraphs to guarantee that a vast majority of queries will be executed within the allotted time budget.Other approximation techniques have been considered before to improve scalability of dynamic authority-based search algorithms. Monte Carlo algorithms are introduced in and for approximation during precomputation. HubRank uses the same approximation as , but performs precomputation only for "hub" nodes. Other techniques might also suggest sampling-based techniques online. However, although these techniques claim online query processing, they have only been demonstrated on graphs with less than 106 links. In contrast, we demonstrate superior scalability of our approach on a Wikipedia graph that is two orders of magnitude larger. We also show that our approximation using ObjectRank itself is more precise than the sampling-based techniques.

I. RELATED WORK

The issue of scalability of PPR [3] has attracted a lot of attention. PPR performs a very expensive fixpoint iterative computation over the entire graph, while it generates personalized search results. To avoid the expensive iterative calculation at runtime, one can naively precompute and materialize all the possible personalized PageRank vectors (PPVs) [2]. Although this method guarantees fast user response time, such precomputation is impractical as it requires a huge amount of time and storage especially when done on large graphs. In this section, we examine hub-based and Monte Carlo style methods that address the scalability problem of PPR, and give an overview of HubRank that integrates the two approaches to improve the scalability of ObjectRank. Even though these approaches enabled PPR to be executed on large graphs, they either limit the degree of personalization or deteriorate the quality of the top-k result lists significantly. Hub-based approaches materialize only a selected subset of PPVs. Topic-sensitive PageRank [2] suggests materialization of 16 PPVs of selected topics and linearly combining them at query time. The personalized PageRank computation suggested in enables a finer-grained personalization by efficiently materializing significantly more PPVs (e.g., 100 K) and combining them using the hub decomposition theorem and dynamic programming techniques. However, it is still not a fully personalized

PageRank, because it can personalize only on a preference set subsumed within a hub set H. Monte Carlo methods[4],[5] replace the expensive power iteration algorithm with a randomized approximation algorithm . In order to personalize PageRank on any arbitrary preference set with maintaining just a small amount of precomputed results, Fogaras et al. introduce the Fingerprint generation can be easily parallelized and the quality of search results improves as the number of fingerprints increases. However, as mentioned in [4], the precision of search results generated by the fingerprint algorithm is somewhat less than that of power-iteration-based algorithms, and sometimes, the quality of its results may be inadequate especially for nodes that have many close neighbors. In a Monte Carlo algorithm that takes into account not only the last visited nodes, but also all visited nodes during the sampled walks, is proposed. Also, it showed that Monte Carlo algorithms with iterative start outperform those with random start.

HubRank is a search system based on ObjectRank that improved the scalability of ObjectRank by combining the above two approaches. It first selects a fixed number of hub nodes by using a greedy hub selection algorithm that utilizes a query workload in order to minimize the query execution time. Given a set of hub nodes H, it materializes the fingerprints of hub nodes in H. At query time, it generates an active subgraph by expanding the base set with its neighbors. It stops following a path when it encounters a hub node whose PPV was materialized, or the distance from the base set exceeds a fixed maximum length. HubRank recursively approximates PPVs of all active nodes, terminating with computation of PPV for the query node itself. During this computation, the PPV approximations are dynamically pruned in order to keep them sparse. As stated in , the dynamic pruning takes a key role in outperforming ObjectRank by a noticeable margin. However, by limiting the precision of hub vectors, HubRank may get somewhat inaccurate search results, as stated in . Also, since it materialized only PPVs ofH, just as , the efficiency of query processing and the quality of query results are very sensitive to the size of H and the hub selection scheme. Finally, Chakrabarti did not show any large-scale experimental results to verify the scalability of HubRank. , we perform quality and scalability experiments on the full English Wikipedia data set exported in October 2007, to show that BinRank is an efficient ObjectRank approximation method that generates a highquality top-k list for any keyword query in the corpus. For comparative evaluation of the performance of BinRank, we implemented the Monte Carlo algorithm in that was shown to outperform other variations in . We also implemented HubRank to check its scalability on our Wikipedia data set. Unlike which proves the convergence to the exact solution on arbitrary graphs, and and which offer exact methods at the expense of limiting the choice of personalization, our solution is entirely heuristic. However,extensive experimental evaluation confirms that on realworld graphs, BinRank can strike a good balance between query performance and closeness of approximation.

3 OBJECTRANK BACKGROUND

In this section, [6],[9],[10] we describe the essentials of ObjectRank We first explain the data model and query processing, and then, discuss the result quality and scalability issues that motivate this paper.

3.1 Data Model

ObjectRank performs top-k relevance search over a database modeled as a labeled directed graph. The data graph *G(V,E)* models objects in a database as nodes, and the semantic relationships between them as edges. A node v €V contains a set of keywords and its object type. For example, a paper in a bibliographic database can be represented as a node containing its title and labeled with its type, "paper." A directed edge e € E from u to v is labeled with its relationship type λ(e).

3.2 Query Processing

For a given query, ObjectRank [3] returns top-k objects relevant to the query. We first describe the intuition behind ObjectRank, introduce the ObjectRank equation, and then, elaborate on important calibration factors. ObjectRank query processing can be illustrated using the random surfer model. A random surfer starts from a random node v_i among nodes that contain the given keyword. These These random surfer starting points are called a base set. For a given keyword t, the keyword base set of t, BS(t), consists of nodes in which t occurs. Note that any node in G can be part of the base set, which makes ObjectRank support the full degree of personalization.

3.3 Quality and Scalability

ObjectRank [4],[5] returns top-k search results for a given query using both the content and the link structure in G. Since it utilizes the link structure that captures the semantic relationships between objects, an object that does not contain a given keyword but is highly relevant to the keyword can be included in the top-k list. This is in contrast to the static PageRank approach that only returns objects containing the keyword sorted according to their PageRank score. This key difference is one of the main reasons for ObjectRank's superior result quality

4 RELEVANT SUBGRAPHS

Our goal is to improve the scalability of ObjectRank while maintaining the high quality of top-k result lists. We focus on the fact that ObjectRank does not need to calculate the exact full ObjectRank vector r to answer a top-k keyword query . We identify three important properties of ObjectRank vectors that are irectly relevant to the result quality and the performance of ObjectRank. First, for many of the keywords in the corpus, the number of objects with non-negligible ObjectRank values is much less than k<<|V| . This means that just a small portion of G is relevant to a specific keyword. Here, we say that an ObjectRank value of v, U(v) is non-negligible if r(v) is above the convergence threshold. The intuition for applying the threshold is thatdifferences between the scores that are within the threshold of each other are noise after ObjectRank execution. Thus, scores below threshold are effectively indistinguishable from zero, and objects that have such scores are not at all relevant to the query term. Second, we observed that top-k results of any keyword term t generated on subgraphs of G composed of nodes with non-negligible ObjectRank values, w.r.t. the same t, are very close to those generated on G. Third, when an object has a non-negligible ObjectRank value for a given base set BS1, it is guaranteed that the object gains a non-negligible ObjectRank score for another base set BS2 if $BS1 \subseteq BS2$. Thus, a subgraph of G composed of nodes with non-negligible ObjectRank values.

5 BIN CONSTRUCTION

As outlined above, we construct a set of MSGs for terms of a dictionary or a workload by partitioning the terms into a set of term bins based on their co-occurrence. We generate an MSG for every bin based on the intuition that a subgraph that contains all objects and links relevant to a set of related terms should have all the information needed to rank objects with respect to one of these terms. There are two main goals in constructing term bins. First, controlling the size of each bin to ensure that the resulting subgraph is small enough for ObjectRank to execute in a reasonable amount of time. Second, minimizing the number of bins to save the preprocessing time. After all, we know that precomputing ObjectRank for all terms in our corpus is not feasible. To achieve the first goal, we introduce a maxBinSize parameter that limits the size of the union of the posting lists of the terms in the bin, called bin size. As discussed above, ObjectRank uses the convergence threshold that is inversely proportional to the size of the base set, i.e., the bin size in case of subgraph construction. Thus, there is a strong correlation between the bin size and the size of the materialized subgraph. As show in Section 8, the value of maxBinSize should be determined by quality and performance requirements of the system. The problem of minimizing the number of bins is NPhard. In fact, if all posting lists are disjoint, this problem reduces to a classical NP-hard bin packing problem . We apply a greedy algorithm that picks an unassigned term with the largest posting list to start a bin and loops to add the term with the largest overlap with documents already in the bin. We use a number of heuristics to minimize the required number of set intersections, which dominate the complexity

PackTermsIntoBins
Input: A set of workload terms W, with their posting lists
Output: A set of bins B

```
( 1) while W is not empty do
( 2)   create a new empty bin b
( 3)   create an empty cache of candidate terms C
( 4)   pick term t ∈ W with the largest posting list size |t|
( 5)   while t is not null do
( 6)     add t to b, and remove it from W
( 7)     compute a set of terms T that co-occur with t
( 8)     for each t' ∈ T do
( 9)       insert (or update) mapping < t', null > into C
(10)     end for each
(11)     bestI := 0
(12)     for each mapping < c, i >∈ C do
(13)       if i = null then // b ∩ c has not been computed yet
(14)         i := |b ∩ c|
(15)         update mapping < c, i > in C
(16)       end if
(17)       union := |b| + |c| − i
(18)       if union > maxBinSize then
(19)         remove < c, i > from C
(20)       else if i > bestI then
(21)         bestI := i
(22)         t := c
(23)       end if
(24)     end for each
(25)     if bestI = 0 then // no candidates left
(26)       pick t ∈ W with maximum |t| ≤ maxBinSize − |b|
(27)       if no such t exists, t := null
(28)     end if
(29)   end while
(30)   add completed b to B
(31) end while
```

Fig. 1. Bin computation algorithm

of the algorithm. The tight upper bound on the number of set intersections that our algorithm needs to perform is the number of pairs of terms that co-occur in at least one document. To speed-up the execution of set intersections for larger posting lists, we use KMV synopses [13] to estimate the size of set intersections.

The algorithm in Fig. 1 works on term posting lists from a text index. As the algorithm fills up a bin, it maintains

a list of document IDs that are already in the bin, and a list of candidate terms that are known to overlap with the bin (i.e., their posting lists contain at least one document that was already placed into the bin). The main idea of this greedy algorithm is to pick a candidate term with a posting list that overlaps the most with documents already in the bin, without posting list union size exceeding the maximum bin size . While it is more efficient to prepare bins for a particular workload that may come from a system query log, it is dangerous to assume that a query term that has not been seen before will not be seen in the future. We demonstrate that it is feasible to use the entire data set dictionary as the workload, in order to be able to answer any query. Due to caching of candidate intersection results in lines 12-14 of the algorithm, the upper bound on the number of set intersections performed by this algorithm is the number of pairs of co-occurring terms in the data set. Indeed, in the worst case, for every term t that has just been placed into the bin, we need to intersect the bin with every term t0 that co-occurs with t, in order to check if t0 is subsumed by the bin completely, and can be placed into the bin "for free."

6 SYSTEM ARCHITECTURE

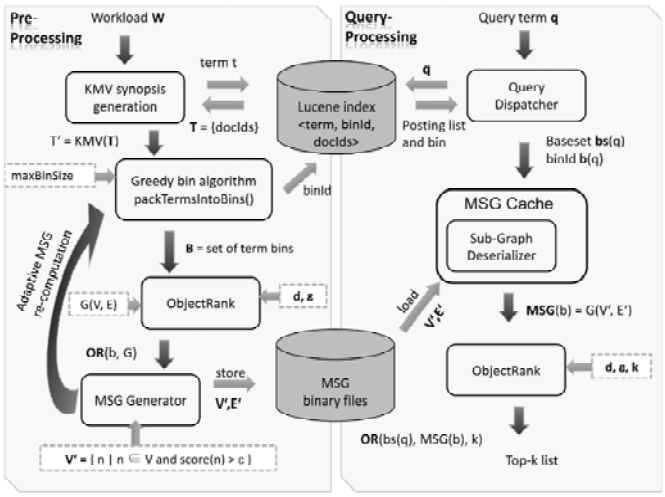

Fig. 2 shows the architecture of the BinRank system.

During the preprocessing stage (left side of figure), we generate MSGs as defined in Section 4. During query processing stage (right side of figure), we execute the ObjectRank algorithm on the subgraphs instead of the full graph and produce high-quality approximations of top-k lists at a small fraction of the cost. In order to save preprocessing cost and storage, each MSG is designed to answer multiple term queries. We observed in the Wikipedia data set that a single MSG can be used for 330-2,000 terms, on average.

6.1 Preprocessing

The preprocessing stage of BinRank starts with a set of workload terms W for which MSGs will be materialized. If an actual query workload is not available, W includes the entire set of terms found in the corpus. We exclude from W all terms with posting lists longer than a system parameter maxPostingList. The posting lists of these terms are deemed too large to be packed into bins. We execute ObjectRank for each such term individually and store the resulting top-k lists. Naturally, maxPostingList should be tuned so that there are relatively few of these frequent terms. In the case of Wikipedia, we used maxPostingList ¼ 2;000 and only 381 terms out of about 700,000 had to be precomputed individually. This process took 4.6 hours on a single CPU. For each term w 2 W, BinRank reads a posting list T from the Lucene3 index and creates a KMV synopsis T0 that is used to estimate set intersections.The bin construction algorithm, PackTermsIntoBins, partitions W into a set of bins composed of frequently co-occurring terms. The algorithm takes a single parameter maxBinSize, which limits the size of a bin posting list, i.e., the union of posting lists of all terms in the bin. During the bin construction, BinRank stores the bin identifier of each term into the Lucene index as an additional field. This allows us to map each term to the corresponding bin and MSG at query time .

6.2 Query Processing

For a given keyword query q, the query dispatcher retrieves from the Lucene index the posting list bs(q) (used as the base set for the ObjectRank execution) and the bin identifierB(q). Given a bin identifier, the MSG mapper determines whether the corresponding MSG is already in memory. If it is not, the MSG deserializer reads the MSG representation from disk. The BinRank query processing module uses all available memory as an LRU cache of MSGs. For smaller data graphs, it is possible to dramatically reduce MSG storage requirements by storing only a set of MSGnodes V`, and generating the corresponding set of edges E0 only at query time. However, in our Wikipedia, data setthat would introduce an additional delay of 1.5-2 seconds, which is not acceptable in a keyword search system. The ObjectRank module gets the in-memory instance of MSG, the base set, and a set of ObjectRank calibrating parameters: 1) the damping factor d; 2) the convergence threshold ε; and 3) the number of top-k list entries k. Once the ObjectRank scores are computed and sorted, the resulting document ids are used to retrieve and present the top-k objects to the user.

Multikeyword queries are processed as follows: For a given conjunctive query composed of n terms $\{t_1 \ldots t_n\}$, the ObjectRank module gets MSGs, MSG(b(t_1) . . . MSG(b(t_n) and evaluates each term over the corresponding MSG. Then, it multiplies the ObjectRank scores obtained over MSGs to generate the top-k list for the query. For a disjunctive query, the ObjectRank module sums the ObjectRank scores w.r.t. each term calculated using MSGs to produce BinRank scores.

7 EXPERIMENTAL EVALUATION

7.1 Object Rank on the full bingo Graph

ObjectRank on G_{bingO} takes too long to be executed online and consumes around 80 MB of memory just for the link information of G_{YAHOO}. As shown in Fig. 3, it takes around 20-50 seconds (30 seconds on average) to compute thedynamically generated top-k list for a given single keyword

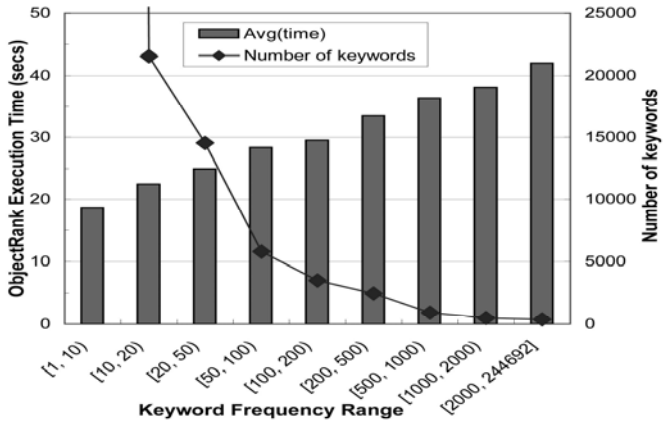

Fig. 3. The number of keywords and average ObjectRank execution time on the bingo graph per frequency range

query even with our optimized, in-memory ObjectRank execution engine. For frequent keywords that have postinglists with more than 200 documents, the ObjectRank is likelyto take longer. Since frequent keywords are found in many articles, they are more likely to be meaningfully connected to many other articles through many paths, resulting in a wider search space for ObjectRank to evaluate and rank. Fig. 3 also shows the keyword frequency distribution obtained from the Lucene text index built on the article titles. The total number of keywords in the index is 698,214, and the keyword frequencies follow the typical power law distribution.

7.3 BinRank

During the BinRank preprocessing stage, we generate bins for all the keywords in the corpus. Once the bins are constructed, we generate an MSG per bin by executing ObjectRank on G_{bingo} using the union of the posting lists of the terms in a bin as a single base set. We first describe the performance of the bin construction and MSG generation, and then, measure the query result quality and the impact of maxBinSize

7.3.1 Preprocessing

Bin construction. To measure the performance of the binconstruction stage, we examine the bin construction timeand the number of bins constructed with different maxBinSize values

.

maxBinSize	bin construction time(secs)	number of bins	number of keywords per bin
2000	180	2107	331
4000	322	1043	669
6000	509	693	1007
8000	737	519	1345
10000	920	414	1686
12000	1106	345	2023

Fig. 4. Performance of bin construction

maxBinSize	num MSGs constructed	avg MSG construction time (secs)	total MSG construction time (hrs)	avg MSG size (MB)	total MSG size (MB)
2000	2107	28.9	16.9	21	44253
4000	1043	40.6	11.8	42	44035
6000	693	42.6	8.2	64	44556
8000	519	46.0	6.6	85	44143
10000	414	48.0	5.5	104	43209
12000	345	50.0	4.8	127	43919

Fig. 5. The effect of maxBinSize on the MSG construction cost

We construct bins for all terms in our Lucene index, except for the 381 most frequent terms which have posting lists longer than a system parameter maxPostingList ¼ 2;000. Recall from Section 7 that such terms are deemed to be too frequent, so we precompute their ObjectRank authority vectors individually. This process takes 1.6 hrs

CONCLUSION

In this paper, we proposed BinRank as a practical solution for scalable dynamic authority-based ranking. It is based on partitioning and approximation using a number of materialized subgraphs. We showed that our tunable system offers a nice trade-off between query time and preprocessing cost.

We introduce a greedy algorithm that groups co-occurring terms into a number of bins for which we compute materialized subgraphs. Note that the number of bins is much less than the number of terms. The materialized subgraphs are computed offline by using ObjectRank itself. The intuition behind the approach is

that a subgraph that contains all objects and links relevant to a set of related terms should have all the information needed to rank objects with respect to one of these terms. Our extensive experimental evaluation confirms this intuition. For future work, we want to study the impact of other keyword relevance measures, besides term co-occurrence, such as thesaurus or ontologies, on the performance of BinRank. By increasing the relevance of keywords in a bin, we expect the quality of materialized subgraphs, thus the top-k quality and the query time can be improved. We also want to study better solutions for queries whose random surfer starting points are provided by Boolean.

REFERENCES :

[1] S. Brin and L. Page, "The Anatomy of a Large-Scale Hypertextual Web Search Engine," Computer Networks, vol. 30, nos. 1-7, pp. 107-117, 1998.
[2] T.H. Haveliwala, "Topic-Sensitive PageRank," Proc. Int'l World Wide Web Conf. (WWW), 2002.
[3] G. Jeh and J. Widom, "Scaling Personalized Web Search," Proc.Int'l World Wide Web Conf. (WWW), 2003.
[4] D. Fogaras, B. Ra´cz, K. Csaloga´ny, and T. Sarlo´ s, "Towards Scaling Fully Personalized PageRank: Algorithms, Lower Bounds, and Experiments," Internet Math., vol. 2, no. 3, pp. 333-358, 2005.
[5] K. Avrachenkov, N. Litvak, D. Nemirovsky, and N. Osipova, "Monte Carlo Methods in PageRank Computation: When One Iteration Is Sufficient," SIAM J. Numerical Analysis, vol. 45, no. 2,pp. 890-904, 2007.
[6] A. Balmin, V. Hristidis, and Y. Papakonstantinou, "ObjectRank:Authority-Based Keyword Search in Databases," Proc. Int'l Conf.Very Large Data Bases (VLDB), 2004.
[7] Z. Nie, Y. Zhang, J.-R. Wen, and W.-Y. Ma, "Object-Level Ranking: Bringing Order to Web Objects," Proc. Int'l World Wide Web Conf.(WWW), pp. 567-574, 2005.
[8] S. Chakrabarti, "Dynamic Personalized PageRank in Entity-Relation Graphs," Proc. Int'l World Wide Web Conf. (WWW), 2007.
[9] H. Hwang, A. Balmin, H. Pirahesh, and B. Reinwald, "Information Discovery in Loosely Integrated Data," Proc. ACM SIGMOD, 2007.
[10] V. Hristidis, H. Hwang, and Y. Papakonstantinou, "Authority- Based Keyword Search in Databases," ACM Trans. Database Systems, vol. 33, no. 1, pp. 1-40, 2008.
[11] M. Kendall, Rank Correlation Methods. Hafner Publishing Co., 1955.
[12] M.R. Garey and D.S. Johnson, "A 71/60 Theorem for Bin
Packing," J. Complexity, vol. 1, pp. 65-106, 1985.
[13] K.S. Beyer, P.J. Haas, B. Reinwald, Y. Sismanis, and R. Gemulla,"On Synopses for Distinct-Value Estimation under Multiset Operations," Proc. ACM SIGMOD, pp. 199-210, 2007..

Mr. M V S NAGARAJU received the MCA Degree from ANU, GUNTUR in 2009 and He is currently pursuing M.Tech in the Department Of Computer Science and Engineering in Saint Theresa Institute Of Engg & Tech Garividi, Vizianagaram, Of JNTUK Affiliation. His research interests include Data Mining and Computer Networks.

Uppe.Nanaji received the B. Tech degree from JNTU, Hyderabad, India and the M. Tech degree in Computer Science Technology from GITAM College Of Engg Of Andhra University Affiliation in Vishakhapatnam in 2003, and he is currently pursuing the Ph. D in Computer Networks from Andhra University Visakhapatnam. He is working as a Head of the Department for CSE in Saint Theresa College Of Engg & Technology Garividi, Vizianagartam (Dist) India. His research interests include Computer Networks & Data Ware Housing