Hub Rank Optimized Bin Rank for Efficient Dynamic Authority Based Search Results

P.Jhansi Rani#1, D.Durga Prasad#2

#1 Student, Dvr & Dr. Hs Mic College Of Technology, Kanchikacherla,Krishna(dt)

#2 Assoc. professor, Dvr & Dr. Hs Mic College Of Technology, Kanchikacherla,Krishna(dt)

#1 jhansip22@gmail.com, #2 ddprasad@rediffmail.com,

Abstract : Credibility of information should be a key-metric for search page results. So Dynamic Authority Ranking over Popularity Ranking is used. Existing dynamic authority-based keyword search algorithms, such as Object Rank and personalized Page Rank are used to provide high quality, high recall search in Web databases but they have huge computation overhead over full graph and are not feasible at query time. Besides preprocessing an index of keywords is expensive. So inspired by materialized views in traditional query processing, BinRank system that approximates Object Rank results was developed earlier. BinRank generates the materialized sub graphs(MSG) by partitioning all the terms in the corpus(information results) based on their co-occurrence, and then executing Object Rank 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. Finally 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. This approach achieves better results in dynamic authority based keyword searches. Simulation over a 10^8 edges Wikipedia data set confirms this. But the limitations of Object Rank concerning the query time still persists. So we use HubRank a Query optimization and index management technique especially for graphs as an alternate to Object Rank. This reduces the query time and has significant performance boost over smaller graphs such as MSG.

Keywords: Personalized Pagerank, ObjectRank, BinRank, HubRank.

I INTRODUCTION

Search is maturing to take advantage of taggers, annotators and extractors that associate entities and relations (ER) with text. A very useful search paradigm that has surfaced in many forms recently is proximity search or spreading activation. In general, fully offline static ranking is not feasible in this application domain, because the match predicates can be diverse, even if limited to words.

Spreading activation has been proposed for searching in graphs for over a decade. ObjectRank was among the first large-scale implementations of proximity search. In ObjectRank, a personalized page rank vector (PPV) is precomputed for each word in the corpus vocabulary and stored in decreasing order of node scores. Object- Rank supports a few monotone score-combining functions for multi-word queries. A multi-word query is executed by an efficient merge of per-word PPVs. Balmin et al. demonstrated, through a relevance feedback survey, that ObjectRank captured a sufficiently powerful class of scoring/ranking functions. Precomputing a million PPVs will take too long, even though ObjectRank uses some clever tricks to reduce PPV computation time for “almost-acyclic” graphs. The public ObjectRank demo appears to maintain a disk cache of word PPVs which are used as and when possible.

A cache of reasonable size will generally be much smaller than the vocabulary size times |V |. To save space, ObjectRank truncates the PPVs if a PPV element is smaller than some threshold. Our goal is to preprocess the ER graph must faster than computing all word PPVs, and yet answer queries much faster

than a query-time ObjectRank computation, while consuming additional index space comparable to a basic text index.

In this paper, BinRank generates the materialized sub graphs(MSG) by partitioning all the terms in the corpus(information results) based on their co-occurrence, and then executing Object Rank. ObjectRank extends PageRank to perform keyword search in databases. ObjectRank 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. BinRank query execution easily scales to large clusters by distributing the subgraphs between the nodes of the cluster we use HubRank a Query optimization and index management technique especially for graphs as an alternate to Object Rank. This reduces the query time and has significant performance boost over smaller graphs such as MSG.

II RELATED WORK

The issue of scalability of personalized pagerank (PPR) 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). 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 suggests materialization of 16 PPVs of selected topics and linearly combining them at query time. The personalized PageRank computation suggested 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.

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, 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. Also, since it materialized only PPVs of H, 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.

III ARCHITECTURE OVERVIEW

we first describe the ObjectRank scoring model. Then we give an overview of how a query is executed; this naturally leads to hub selection and query optimization issues.

A) Scoring in a TypedWordGraph:

A query is a set of distinct words. To process the query, the ER graph is augmented with one node for each query word, connected to entity nodes where the word appears, with special edges of type “word-to-entity”. A word node appears in W only if it matches at least one entity; thus, no word

node is a dead end. For the moment assume that no entity node is a dead end.

Word rareness. Note that an "inverse document frequency" (IDF) effect is built into the design. Suppose the query has one rare and one frequent word. Each gets half the Pagerank of d, but the rare word passes on the Pagerank to a few entity neighbors, each getting a large share. The frequent word is connected to many entity nodes, each getting a tiny share of its Pagerank.

Deadends and irreducibility. A subgraph NW ⊆ N is reachable from W, the rest can be ignored for a specific query and our graph is effectively W ∪ NW. To make this irreducible and aperiodic, we add fake edges from entity nodes in NW to a sink entity node s. This eliminates dead ends in NW. s has a self-loop.

Learning β automatically. Assigning weights β(t) for each edge type t might seem like an empirical art from the above discussion. Indeed, ObjectRank and related systems use manually-tuned weights.

B) Query processing overview

At startup, our system preloads only the entity nodes N (including the sink node s). This would be impractical for Web-scale data, but is reasonable for ER search applications. Only the graph skeleton is loaded, costing us only about eight bytes per node and edge. Entity graphs with hundreds of millions of nodes and edges can be loaded into regular workstations.

Given a keyword query to execute, we instantiate the query words as nodes in W and attach each word node to the entity nodes where the word appears. This gives us a setup time not too large compared to IR engines. To answer the query, we need the PPVs of the nodes corresponding to the query words.

A total teleport mass of 1 first reaches the word nodes, then diffuses out to N. Every node on the way attenuates the total outflow of Pagerank by a factor α < 1. Therefore, we expect the effect of distant nodes on a word PPV to be decay rapidly. We stop expanding the active subgraph at two kinds of boundary nodes: blocker2 nodes B ⊂ H whose PPVs have been precomputed and stored, and loser nodes that are too far from the word nodes to assert much influence on the word PPVs.

IV WORKLOADDRIVENHUBSELECTION

We wish to minimize the average time taken to compute PPVs for all active nodes over a representative query mix. PPV computation time is likely to be directly related to the number of active nodes, but the connection is not mathematically predictable. Even if we were to make simplifying assumptions (such as a fixed number of iterations), the problem of picking the best subset reduces to hard problems like dominating set or vertex cover. Even quadraticor cubic-time algorithms on CiteSeer-scale graphs may be impractical. Therefore we turn to efficient and reasonable near-linear-time heuristics.

An indirect approach is to try to arrest active graph expansions with blocker nodes as quickly and effectively as possible. I.e., we want to pick a small number of hub nodes that will block expansions from a large number of frequent query word nodes. If a node is a good hub, it will be reachable along short paths from frequent query word nodes. This leads to the greedy hub ordering approach shown in Figure 1; it might be regarded as a fast if crude approximation to the push step.

```
1: initialize a score map score(u) for nodes u ∈ W0 ∪ N
2: for each query word w ∈ W0 do
3:    attach node w to the preloaded entity graph
4:    let frontier = {w} and priority(w) = Pr(w)
5:    create an empty set of visited nodes
6:    while frontier ≠ ∅ do
7:       remove some u from frontier and mark visited
8:       accumulate priority(u) to score(u)
9:       for each visited neighbor v do
10:         accumulate priority(u) α C(v,u) to score(v)
11:      for each unvisited neighbor v do
12:         let priority(v) = priority(u) α C(v,u)
13:         add v to frontier
14: sort word and entity nodes by decreasing score(u)
```

Fig 1: Greedy estimation of a measure of meritto including each node into the hub set

V ACTIVE SUBGRAPH EXPANSION

When a keyword query is submitted, we perform an expansion process similar to that in Figure 1 to collect the active nodes A, except that we also identify blocker nodes B ⊂ H whose FPs have been indexed, and loser nodes ` which are so distant from the originating node w that even the largest element of PPV(l) is unlikely to influence PPV(w) much.

As in earlier work on local Pagerank computation we judge if PPVv is "unlikely to influence" PPVu much via a heuristic. Ideally, we should check if the conductance from u to v is small, but that amounts to solving part of the PPV estimation problem (which is what BCA does). Chien et alpropose a one-pass weight-dissipation type algorithm similar to ours, except that, in the interest of speed, we further omit conductance via multiple paths, noting only the largest conductance path from u to v.

VI PERFORMANCE

Effect of increasing cache size. Average query time for HubRank drops steeply as the FP cache size increases. For our testbed, the steep decrease happens right around the same size as the Lucene text index (Figure 2). But long before the "knee" is reached, HubRank times are less than 1/12 th that of ObjectRank.

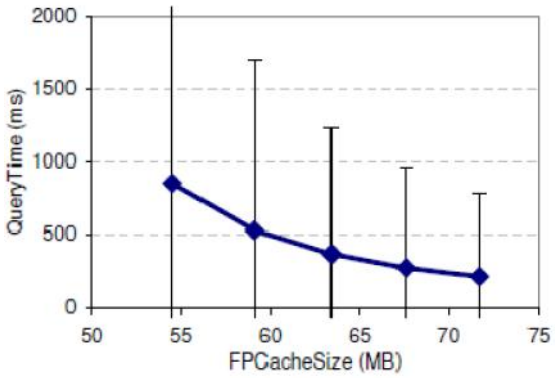

Figure 2: Effect of cache size on HubRank query execution time

Smoothing. Figure3 shows the beneficial effects of workloadsmoothing on accuracy and speed. Smoothing ensures that hubs are picked reasonably close to word nodes that do not even appear in the training query log. This improves both speed and accuracy.

#words→	1	2	3	4
NoSmoothTime	622	566	756	994
SmoothTime	280	310	549	836
NoSmoothPrec	.81	.85	.85	.85
SmoothPrec	.85	.91	.92	.91

Figure 3: HubRank time and precision with andwithout smoothing

Comparison with BCA. BCA can be regarded as a refined version. To maintain precise guarantees,BCA starts with a residual of r(w) at word node w, andprogresses using push steps. A push from node u transfers 1−α times its current residual to its score, and the remaining α fraction of its current

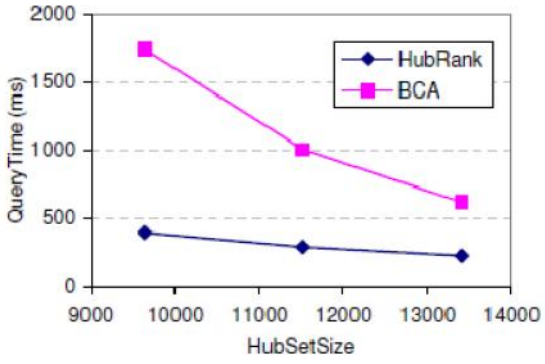

Figure 4: BCA has somewhat higher overhead thanHubRank.

residual to its out-neighbors' residuals. BCA has to maintain a heap for the node residuals. Using a Fibonacci heap, BCA is somewhat slower than HubRank in our preliminary experiments (Figure 4), but both are substantially faster than ObjectRank.

VII CONCLUSION

Previously, BinRank as a practical solution for scalable dynamic authority-based ranking. It is based on partitioning and approximation using a number of materialized subgraphs. In this paper, We presented HubRank, a workload-driven indexingand dynamic personalized search system for ER graphs. we use HubRank a Query optimization and index management technique especially for graphs as an alternate to Object Rank. This reduces the query time and has significant performance boost over smaller graphs such as MSG. Our indexing and search are

"anytime algorithms" in the sense that we can abandon them at any time but get increasing quality with the effort invested. At present HubRank scales to the size of DBLP and CiteSeer, with several millions nodes and almost a million words. We separated sample queries where all words were blocked (empty active set, complete word FPs loaded) vs. queries with nontrivial active sets. We are working on graph clustering and indexing techniques to reduce disk seeks while expanding the active subgraph and loading blocker PPVs, while the graph is largely on disk. We would also like to support broader classes of predicates on nodes, perhaps involving structured attributes and cached views over and above word matches.

VII 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] S. Abiteboul, M. Preda, and G. Cobena. Adaptive on-line page importance computation. In WWW Conference, pages 280–290, 2003.
[6] Apache Software Group. Jakarta Lucene text search engine. GPL Library, 2002.
[7] A. Balmin, V. Hristidis, and Y. Papakonstantinou. Authority-based keyword queries in databases using ObjectRank. In VLDB, Toronto, 2004.
[8] P. Berkhin. Bookmark-coloring approach to personalized pagerank computing. Internet Mathematics, 3(1), Jan. 2007. Preprint.
[9] Z. Nie, Y. Zhang, J.-R. Wen, and W.-Y. Ma. Object-level ranking: Bringing order to Web objects. In WWW Conference, pages 567–574, 2005.
[10] S. Chakrabarti, "Dynamic Personalized PageRank in Entity-Relation Graphs," Proc. Int'l World Wide Web Conf. (WWW), 2007.