Optimized Dynamic results using Hub Rank

KISHORE REDDY.THUMMALA[1]

ASSOCIATE PROFESSOR, RAMANANDHA ENGINEERING COLLEGE, NALGONDA, ANDHRA PRADESH

Abstract: In recent years retrieving information from the dynamic web using data mining is the main concept of research work. For that reason our contributors develop many of techniques for retrieving information. Using Page ranks and Object rank. But they are not support for dynamic data retrieving in web pages effectively so we have to go for better technique for above problem. In this paper developing Hub Rank and Bin Rank based on Object Rank for retrieving efficient results.

Keywords: Personalized Page rank, Object Rank, Bin Rank, Hub Rank.

I INTRODUCTION

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.

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. Spreading activation has been proposed for searching in graphs for over a decade. Object Rank was among the first large-scale implementations of proximity search. In Object Rank, a personalized page rank vector (PPV) is recomputed 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 Object Rank captured a sufficiently powerful class of scoring/ranking functions. Precomputing a million PPVs will take too long, even though Object Rank uses some clever tricks to reduce PPV computation

time for “almost-acyclic” graphs. The public Object Rank demo appears to maintain a disk cache of word PPVs which are used as and when possible. Our goal is to preprocess the ER graph must faster than computing all word PPVs, and yet answer queries much than a query-time Object Rank computation, while consuming additional index space comparable to a basic text index.

Object Rank 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. Bin Rank 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.

In this paper, Bin Rank 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. Object Rank extends Page Rank to perform keyword search in databases.

II RELATED WORK

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.

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.

ISSN-2320-7884 (ONLINE)
ISSN-2321-0257 (PRINT)

Hub-based approaches materialize only a selected subset of PPVs. Topic-sensitive Page Rank suggests materialization of 16 PPVs of selected topics and linearly combining them at query time. The personalized Page Rank 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.

To avoid the expensive iterative calculation at runtime, one can naively pre computing and materialize all the possible personalized Page Rank vectors (PPVs). However, it is still not a fully personalized Page Rank, because it can personalize only on a preference set subsumed within a hub set H.

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 sub graph by expanding the base set with its neighbors. Hub Rank is a search system based on Object Rank that improved the scalability of Object Rank by combining the above two approaches. It stops following a path when it Encounters a hub node who's PPV was materialized, or the distance from the base set exceeds a fixed maximum length. Hub Rank 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 Object Rank by a noticeable margin. However, by limiting the precision of hub vectors, Hub Rank 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 Hub Rank.

III ARCHITECTURE OVERVIEW

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

A) Scoring in a Typed Word Graph:

A word node appears in W only if it matches at least one entity; thus, no word 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". 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 Page rank of d, but the rare word passes on the Page rank 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 Page rank.

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

B) Query processing overview

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.

C) 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.

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 $\alpha < 1$. Therefore, we expect the effect of distant nodes on a word PPV to be decay rapidly. We

ISSN-2320-7884 (ONLINE)
ISSN-2321-0257 (PRINT)

stop expanding the active subgraph at two kinds of boundary nodes: blocker2 nodes B ⊂ H who's PPVs have been precompiled and stored, and loser nodes that are too far from the word nodes to assert much influence on the word PPVs.

IV WORKLOAD DRIVEN HUBSELECTION

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 quadratic or 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.

V ACTIVE SUBGRAPH EXPANSION

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. 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.

```
initialize a score map score(u) for nodes u ∈ W0 ∪ N
for each query word w ∈ W0 do
  attach node w to the preloaded entity graph
  let frontier = {w} and priority(w) = Pr(w)
  create an empty set of visited nodes
  while frontier ≠ ∅ do
    remove some u from frontier and mark visited
    accumulate priority(u) to score(u)
    for each visited neighbor v do
      accumulate priority(u) α C(v, u) to score(v)
    for each unvisited neighbor v do
      let priority(v) = priority(u) α C(v, u)
      add v to frontier
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

VI PERFORMANCE

Effect of increasing cache size.

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

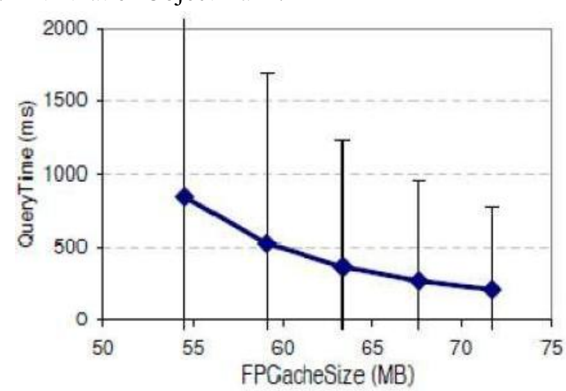

Figure 2: Effect of cache size on HubRank query execution time

Smoothing. Figure3 shows the beneficial effects of workload smoothing 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.

ISSN-2320-7884 (ONLINE)
ISSN-2321-0257 (PRINT)

#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, and progresses 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 residual to its out-neighbors' residuals. BCA has to maintain a heap for the node residuals.

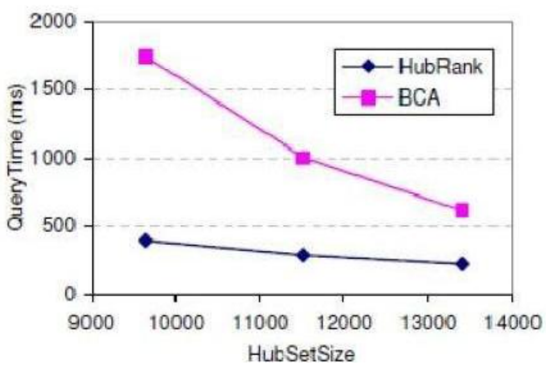

Figure 4: BCA has somewhat higher overhead than HubRank

VII CONCLUSION

In this paper, we presented HubRank, a workload-driven indexing and 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. 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.

VIII 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.

About Author:

I am Tummala Kishore Reddy, Completed M.Tech in Swami Ramananda Tirtha Institute of Science & Technology, Working as associate Professor in Swami Ramananda Tirtha Institute of Science & Technology since five years. My interest is research in different programming languages and in datamining.