Ranking the Answer Trees of Graph Search by both Structure and Content

Ming Zhong
State Key Laboratory of Software Engineering,
Computer School, Wuhan University
Luojiashan
Wuhan, China
clock@whu.edu.cn

Mengchi Liu
School of Computer Science, Carleton University
1125 Colonel By Drive
Ottawa, Canada
mengchi@scs.carleton.ca

ABSTRACT

Keyword search on graphs aims to find minimum connected trees containing the keywords. Normally, the answer trees are ranked by their topological structures. However, this basic ranking scheme does not distinguish answer trees well when many answer trees have the same structures or contain redundant information. This paper proposes a novel ranking scheme, which combines both structure-based and content-based ranking factors. It can effectively prioritize the answer trees with more valuable content and punish the ones with redundant information. Meanwhile, it will not reduce the efficiency of top-k search algorithms by performing an edge re-weighting process offline.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms

Theory

Keywords

Graph, Search, Ranking

1. INTRODUCTION

A growing number of graph-structured data repositories have been available in recent years, such as DBPedia[1], Yago[2], FreeBase[3] and etc. Consequently, keyword search on such data graphs becomes a popular research topic. The search results are normally defined as *minimum connected trees* in the graph. In this way, meaningful information can be captured when there are no single entities (i.e., vertices in the graph) matched by all keywords, which is a common scene because of database normalization. The answer

[1]http://dbpedia.org/
[2]http://www.mpi-inf.mpg.de/yago-naga/yago/
[3]http://www.freebase.com/

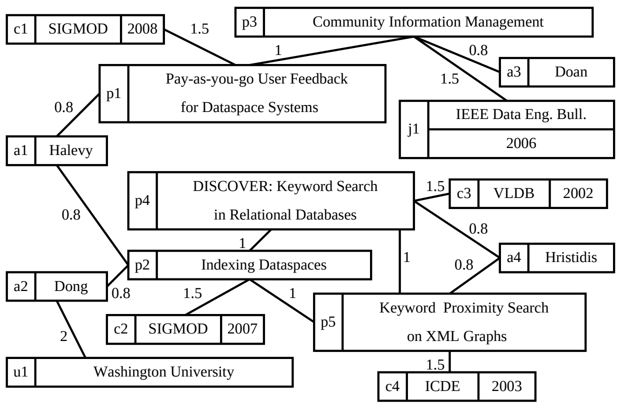

c1 SIGMOD 2008
p3 Community Information Management
p1 Pay-as-you-go User Feedback for Dataspace Systems
a3 Doan
j1 IEEE Data Eng. Bull. 2006
a1 Halevy
p4 DISCOVER: Keyword Search in Relational Databases
c3 VLDB 2002
a4 Hristidis
p2 Indexing Dataspaces
a2 Dong
c2 SIGMOD 2007
p5 Keyword Proximity Search on XML Graphs
u1 Washington University
c4 ICDE 2003
1.5
1
0.8
1.5
0.8
0.8
1.5
0.8
1
1
0.8
0.8
1.5
1
2
1.5

Figure 1: An example graph.

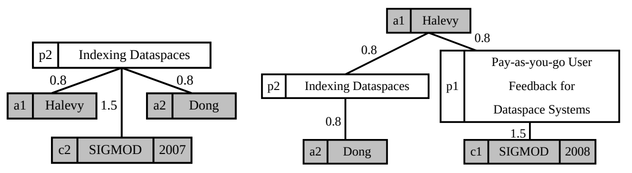
trees are scored and ranked by their *compactness*, since the compacter answer trees are generally considered to be more relevant to the users' information needs.

However, ranking only by tree compactness is too coarse-grained sometimes, especially for large-scale graphs where a large number of answer trees with the same topological structures can be found. For example, consider search the papers coauthored by "Tom" and "Jerry". The typical answer trees are composed by a paper vertex as root and two author vertices as leaves. There could be many such answer trees of different combinations of papers and authors. Surely we would like to see that the trees containing more important papers and authors are ranked higher.

Therefore, we also need to consider the content-based ranking factors. Previous works (e.g., [1, 7]) have dealt with the combination of tree compactness and keyword relevance, which is measured by the classical tf·idf function. However, this "dynamical" scoring function usually leads to inefficiency in the sense of top-k search (e.g., [2, 3, 5, 9–11]). Moreover, if the vertex labels only contain a few of terms like in the graph illustrated in Figure 1, keyword relevance rarely contributes to ranking.

Another issue has to be considered is the redundant information among answer trees. Many answer trees could be composed of almost same vertices. However, existing solutions (e.g., [5, 6]) either unreasonably get rid of too many meaningful answers or are very inefficient.

As a result, we focus on ranking answer trees by both topological structure and "static" content in this paper. We use the vertex weight to represent the importance of a vertex based on its con-



(a) The best MCT answer. (b) Another MCT answer.

Figure 2: Answer trees to query {Halevy Dong SIGMOD}.

tent. The principle challenge is how to integrate two types of ranking factors uniformly so that the search algorithm does not need to enumerate all answers for identifying the top-k answers. To address this problem, we propose a novel ranking scheme, which has two features: 1) we combine the content-based vertex weights into edge weights before search so that the monotonicity of structure-based scoring function is maintained and thus the top-k search is still efficient; 2) we incrementally punish the produced answer trees sharing the same root for avoiding redundant information, which is more efficient than the previous isomorphism testing approach [5].

The rest of this paper is organized as follows. Section 2 introduces the data model and query model. Section 3 presents our ranking scheme. Section 4 reports a simple evaluation. Lastly, we conclude in Section 5.

2. DATA AND QUERY MODELS

We formally represent a data graph as $\mathcal{G}$ = ($\mathcal{V}$, $\mathcal{E}$, Σ, ζ, ω_v, ω_e), where

- $\mathcal{V}$ is a set of vertices;
- $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of edges, which could be either directed or undirected;
- Σ is a set of terms;
- ζ is a vertex label function, and for each vertex $v \in \mathcal{V}$, $\zeta(v) \subseteq \Sigma$;
- ω_v is a vertex weight function which assigns a positive weight $\omega_v(v)$ to each vertex $v \in \mathcal{V}$;
- ω_e is an edge weight function which assigns a positive weight $\omega_e(e)$ to each edge $e \in \mathcal{E}$.

A query $\mathcal{Q}$ is a set of keywords. For a vertex v, if $\exists t \in \mathcal{Q}$ such that $t \in l(v)$, v is called a *matched vertex* of keyword t. The answers to $\mathcal{Q}$ are minimum connected trees, which are connected trees of $\mathcal{G}$ containing at least one matched vertex of each keyword. Moreover, any proper subtrees of them do not contain (the matched vertices of) all keywords. For an answer tree a, let $root(a)$ be its root and $rkpath(a, t)$ be the path between $root(a)$ and the nearest matched vertex of keyword t in a.

For example, consider the graph in Figure 1. Two answer trees to a query {Halevy Dong SIGMOD} are illustrated in Figure 2(a) and (b), respectively.

3. RANKING SCHEME

We treat *tree compactness* as the priority of ranking answer trees, which is a common structure-based ranking factor. Compacter trees in which matched vertices are closer to each other are generally considered to be better. For example, the answer tree in Figure 2(a) is surely better than the answer tree in Figure 2(b). The previous one exactly match the information need of user and contains no useless information.

There are different ways of measuring tree compactness. Some previous works (e.g., [2, 7, 8]) measure tree compactness by the number or total weight of edges. In that case, finding the top-k answers can be reduced to the Group Steiner Tree Problem (GSTP), which is NP-Complete. For supporting efficient top-k search, we follow the root-path semantics of [6] to compute the compactness of an answer tree a to query $\mathcal{Q}$ as the sum of lengths of all root-keyword paths in a, as shown in the following equation.

$$score(a) = \sum_{t \in \mathcal{Q}} |rkpath(a, t)| \cdot \gamma \qquad (1)$$

where $|rkpath(a, t)|$ is the length of path, and $\gamma \geqslant 1$ is a redundancy penalty factor. The rationale behind redundancy penalty is that users would not like to see many answers sharing the same information. For that, [6] permits only a single answer tree rooted at a distinct vertex to be produced, which leads to loss of most answers. [5] implements redundancy penalty by computing tree isomorphism between answers, which is too time-consuming. We do not get rid of other answers having the same root, but incrementally raise their scores by multiplying by γ = $1 + 0.05 \cdot \beta$, where β is the number of produced answers with the same root as this answer. Therefore, the more answers with the same root have been produced, the more penalty this answer will receive.

For example, the scores of answer trees illustrated in Figure 2(a) and (b) are 0.8 + 0.8 + 1.5 = 3.1 and 0 + 1.6 + 2.3 = 3.9 respectively, if no other answers rooted at vertices p2 and a1 are produced. Notice that, the answer with lower score is better under our scoring way. So, the scores clearly indicate that the answer in Figure 2(a) is better than the one in Figure 2(b).

Let us consider how to efficiently process a keyword query in top-k manner under the scoring function. A keyword query $\mathcal{Q}$ can be divided into $|\mathcal{Q}|$ subqueries, each of which aims to enumerate paths started from (the matched vertices of) a keyword $t \in \mathcal{Q}$. When a set of paths enumerated by each subquery have a same destination vertex, they can be combined to produce an answer tree rooted at the destination vertex. In particular, a produced answer tree will be checked if it should be punished for redundancy when it is inserted to the top-k answer list. Since the score of an answer tree (see Equation 1) is monotone with respect to the length of paths, we can follow the line of Threshold Algorithm [4] to handle all subqueries by enumerating paths in priority of their length. In each iteration of main loop, a scheduler determines which subquery to be evaluated. The discussion of detailed search algorithm is out of the scope of this paper.

As we mentioned above, ranking only by tree compactness is too coarse-grained to distinguish answers sometimes, since many answer trees have the same topology structures. Therefore, we also need the content-based ranking factors. However, the "dynamical" content-based factor like keyword relevance will break the monotonicity of scoring function, and thus lead to inefficiency. Consequently, we use the "static" vertex weight to represent the content importance of vertices and further rank answers in a fine-grained manner. The vertex weight is determined with respect to the specific applications. An example is given in Section 4.

There are two problems to be considered. First, how to combine two types of ranking factors uniformly so that we do not need to enumerate all answers for identifying the top-k? Second, an answer with higher score of tree compactness is worse, but an answer with higher score of vertex weight should be better. It is a contradiction.

Table 1: Test graph.

	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	$\lvert\Sigma\rvert$	$\omega'_e(e)$
DBLP	840K	1300K	95K	[0.31, 0.99]

To address these two problems, we present a simple and effective edge re-weighting mehtod as follows. The main idea is: combine the impact of vertex weight into edge weight, and still use Equation 1 to score answers. The new edge weight $\omega'_e(e)$ for $e = (v, u) \in \mathcal{E}$ is computed as follows.

$$\omega'_e(e) = (1 - \sqrt{\frac{\omega_v(v) + \omega_v(u)}{2 \cdot max\{\omega_v(y)|y \in \mathcal{V}\}}}) \cdot \omega_e(e) \qquad (2)$$

Equation 2 intends to reduce edge weights with respect to the vertices they link. The weight of an edge connecting vertices with higher weights will shrink more, and thus answers containing this edge (and of course the vertices it connects) will have a smaller score, thereby being ranked higher. The balance of two measures can be adjusted according to applications. By using Equation 1 and 2, structure-based and content-based ranking factors are seamlessly integrated. More importantly, existing top-k search algorithms only based on tree compactness can still be used.

4. EVALUATION

We used a graph generated from DBLP[4] to do a preliminary e-valuation. Table 1 lists the statistics of our test graph.

To properly rank the search results on DBLP graph, we first weighted the vertices with respect to their contents. The paper vertices were assigned four different initial weights, i.e., 1, 10 100, and 1000, according to the reputation of conferences and journals in which these papers are published. The initial weights of author vertices were all set as 1, since there is no objective way of judging the importance of authors. Then, the weights of vertices were computed iteratively by using the following Equation 3 in a PageRank manner, until all vertex weights did not change anymore.

$$\omega_v(v) = \alpha \cdot \frac{init(v)}{\sum_{u \in \mathcal{V}} init(u)} + (1 - \alpha) \cdot \sum_{u \in \mathcal{V}} \frac{\omega_v(u)}{degree(u)} \qquad (3)$$

where $init(v)$ is the initial weight of vertex v, $degree(v)$ is the number of edges connected with v, and α is a factor to adjust the proportion of two contributions to the final weight, i.e., initial self weight and incoming weights from connected vertices. In this way, the authors of important papers will become weighter than others. So, the final weights of all vertices became balanced.

There were two types of edges, "write" and "cite". We simply set the initial weights of all write edges as 1.0 and cite edges as 0.8, since cite is a weaker relationship than write. Then, the final weight of an edge was computed by using Equation 2 that reduces its initial weight with respect to the weights of two vertices connected by it. As a result, the final weights of all edges were in a range from 0.31 to 0.99. The weight of an edge connecting two important vertices with higher weights will shrink more, and thus answers containing this edge (and of course the vertices connected by it) have smaller scores and thereby are ranked higher. Hence, the top-k answers under such a ranking scheme would be more meaningful to users.

We built a collection of keyword queries to examine ranking effect. By analyzing the overall search results, we observed that the search results were indeed improved by re-weighting the edges with respect to vertex weights reflecting content. For the test queries, most top answers contained the influential authors or papers published on the top-tier conferences or journals, so that we can easily get the most important information from the top down.

Moreover, the redundancy penalty worked well too. For example, a keyword query {X rule mining}, where X was a common family name, was posed on DBLP to search authors named X and doing research on rule mining. Although the author in the top-1 answer published a lot of papers about rule mining, in other words, many good answers about this author could be produced, these answers did not occupy the whole top-k list due to redundancy penalty, and we still found other authors there.

[4]http://www.informatik.uni-trier.de/ ley/db/

5. CONCLUSION

In this paper, we study the problem of ranking the answer trees of graph search. We propose a novel ranking scheme that integrates structure-based and content-based factors together. It can improve the search results by updating the edge weights according to the weights of vertices they connect, and meanwhile does not cause any delay to existing top-k search algorithms. Moreover, it also has a simple and effective way, namely, punish the answers with the same root to avoid redundant information.

6. ACKNOWLEDGMENTS

Supported by the China Postdoctoral Science Foundation funded Project (No. 20100480869).

7. REFERENCES

[1] Z. Bao, T. W. Ling, B. Chen, and J. Lu. Effective xml keyword search with relevance oriented ranking. In *Proc. ICDE*, pages 517–528, 2009.

[2] G. Bhalotia, A. Hulgeriy, C. Nakhez, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using banks. In *Proc. ICDE*, pages 431–440, 2002.

[3] B. Ding, J. X. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin. Finding top-k min-cost connected trees in databases. In *Proc. ICDE*, pages 836–845, 2007.

[4] R. Fagin. Combining fuzzy information from multiple systems. In *Proc. PODS*, pages 1–10, 1998.

[5] K. Golenberg, B. Kimelfeld, and Y. Sagiv. Keyword proximity search in complex data graphs. In *Proc. SIGMOD*, pages 927–940, 2008.

[6] H. He, H. Wang, J. Yang, and P. S. Yu. Blinks: Ranked keyword searches on graphs. In *Proc. SIGMOD*, pages 305–316, 2007.

[7] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient ir-style keyword search over relational databases. In *Proc. VLDB*, pages 850–861, 2003.

[8] V. Hristidis, Y. Papakonstantinou, and A. Balmin. Keyword proximity search on xml graphs. In *Proc. ICDE*, pages 367–378, 2003.

[9] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional expansion for keyword search on graph databases. In *Proc. SIGMOD*, pages 505–516, 2005.

[10] B. Kimelfeld and Y. Sagiv. Finding and approximating top-k answers in keyword proximity search. In *Proc. PODS*, pages 173–182, 2006.

[11] W.-S. Li, K. S. Candan, Q. Vu, and D. Agrawal. Retrieving and organizing web pages by "information unit". In *Proc. WWW*, pages 230–244, 2001.