Organization and Query Optimization of Large-Scale Product Knowledge

You Li[1], Taoyi Huang[2], Hao Song[2], and Yuming Lin[2(✉)]

[1] Guangxi Key Laboratory of Automatic Detecting Technology and Instruments, Guilin University of Electronic Technology, 541004 Guilin, China
[2] Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, 541004 Guilin, China
ymlin@guet.edu.cn

Abstract. Knowledge graph is essential infrastructure of lots of intelligent Web applications. Recently, various types of knowledge graphs are designed and deployed to make the applications more smarter. However, the large amount and heterogeneity of product knowledge bring new challenges for managing such knowledge data. In this work, we propose a scalable framework for organizing large-scale product knowledge, which includes the objective product knowledge and the subject users' opinion knowledge. In order to improve the efficiency of knowledge query, we design a hybrid index structure with a learned model and several B-Tree indexes. Finally, a join strategy based on the variable combination of aspect and opinion is proposed to implement the query optimization. The experimental results show that the proposed method can improved the query efficiency significantly on a large-scale product knowledge compared with a states-of-the-art knowledge management system.

Keywords: Product knowledge · Organization · Query optimization

1 Introduction

With the rapid development and popularization of internet technology and E-Commerce, the count of data including product information increases dramatically on Web. However, it is till hard to meet the users' demand on acquiring accurate information on products. One of the fundamental reasons is that such massive information exists on Web in unstructured or semi-structured form, which limits severely them to be applied automatically and intelligently. On the other hand, the lack of effective management mechanism on such information makes users confront directly fragmented and redundant information, which exacerbates the problem of information overload.

Knowledge graph is an effective way to solve such problem above, which targets at extracting the knowledge from Web and managing these knowledge efficiently. Recently, various types of knowledge graph projects (such as YAGO [1], Freebase [2]) and knowledge management systems (such as RDF-3X [3],

W. Ni et al. (Eds.): WISA 2019, LNCS 11817, pp. 492–498, 2019.
https://doi.org/10.1007/978-3-030-30952-7_48

gStore [4]) have been proposed and developed. However, product knowledge includes the objective knowledge (such as product taxonomy) and the subject knowledge (such as user opinion) in general, both of them are important to many intelligent Web applications such as product recommendation [5]. This heterogeneity of knowledge brings new challenge for managing such knowledge. Moreover, existing systems often need to transform texts like URIs into ID values for query processing, which leads to extra cost for accessing the index frequently.

In this work, we propose a presentation framework for product knowledge to organize the objective product knowledge and the subject one uniformly at first. Then, we design a hybrid index structure to improve the retrieval efficiency based on the learned model and the traditional B-Tree index. Further, we propose join strategy based on the variable combination of aspect and opinion to accelerate the product knowledge retrieval. At last, a series of experiments carried on a large-scale dataset show that the proposed method can improve the query performance on product knowledge significantly compared with a state-of-the-art knowledge management system.

2 The Organization Framework of Product Knowledge

The product knowledge can be divided into two groups generally: the object knowledge and the subject one. The former describes the commonsense knowledge such as product taxonomy, product's aspects; the latter includes mainly users' opinion on a product or product's aspects (Fig. 1).

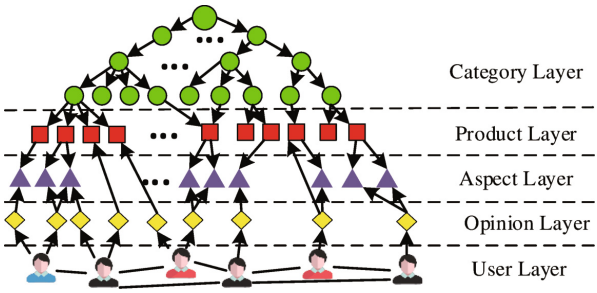


Fig. 1. An overview of product knowledge framework

Specifically, let product concept set $C = \{c_1, \cdots, c_{n1}\}$, product instance set $P = \{p_1, \cdots, p_{n2}\}$, and product aspect set $A = \{a_1, \cdots, a_{n3}\}$. Then, the entity set $E = C \cup P \cup A \cup T \cup F$, where $F = \{f_1, \cdots, f_{n5}\}$ is the set of facts, $f_i = <x, p_i, y>$ is a fact, $x \in E$, $y \in E$ and p_i is a predicate. We define some predicates used in this work in Table 1. Based on these symbols, we can present the product knowledge with the form of RDF (Resource Description Framework) triples.

Table 1. Some defined predicates and the corresponding descriptions

Predicate	Fact	Description
subCategory	<x, subCategory, y>	x is the sub category of y
productOf	<x, productOf y>	x is a product of y
aspectOf	<x, aspectOf, y>	x is an aspect of y
write	<x, write, y>	user x writes review y
reviewOn	<x, reviewOn, y>	review x is on product/aspect y

3 The Hybrid Mapping Index Structure

In existing knowledge management systems, lots of URI texts need to be transformed into ID values with mapping index for query processing. The B-Tree index is often used to speed up this process. However, as the size of index increasing, the query efficiency is gradually reduced. The learned index structure is a novel technique to build indexes by utilizing the distribution of data being indexed, which could provide benefits over state-of-the-art database indexes [6].

In order to applying machine learning model to index the URI texts, each URI is transformed into an one-dimensional array with ASCII code. Based on these sorted one-dimensional arrays, we design a two-layer mapping structure shown in Fig. 2, which combines a learned model and multiple B-Tree indexes.

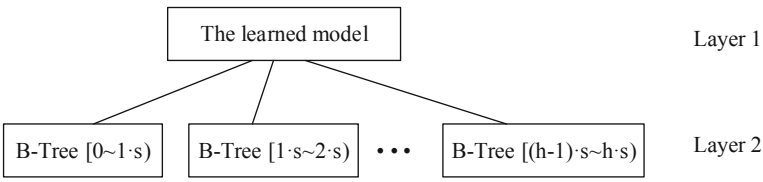


Fig. 2. An overview of product knowledge framework

The first layer is a learned model trained on the distribution of the sorted URI arrays, which tries to find the approximate location of each URI. Specially, we apply the neural network with one fully-connected hidden layer as the learned model, where the activation function is the *ReLu* function defined as follows.

$$RuLu(x) = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases} \tag{1}$$

In the training phase, we apply the variance of predicted value and true value of URI's location as the loss function. Since the product knowledge is coming from the Web, the URI of entity is relatively long. Then, it needs more nodes in input layer of neural network, which reduces the execution efficiency of model.

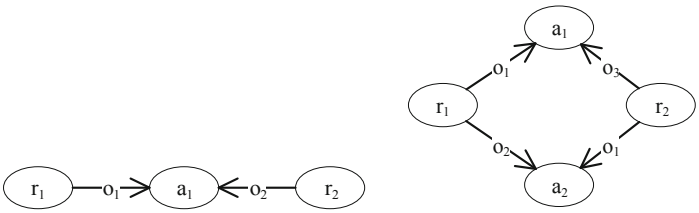
Considering the types of namespaces included in URIs is relatively few and fixed, we construct a prefix tree to compress the redundant namespace prefix. By this way, we can reduce the count of nodes in input layer greatly.

Ideally, the learned model would predict the exact value of each URI. However, there is a relatively large error between the predicted value and the true value of a URI's location because of the data distribution's complexity. Then, the second layer contains h B-Tree structures, which locate the URI accurately based on the approximate location predicted by the learned model. In order to improve the retrieval performance, we apply a threshold s to divide the URIs into multiple blocks for keeping the B-Trees even.

4 Query Optimization of Product Knowledge

The *join* operation is an important factor for influencing the query performance. When a SPARQL query is executed on product graph, the query is transformed into a query graph at first. Then, we will generate the candidates of each variable according to the adjacency of the variable node at first. At last, the *join* operation is executed to generate the result set by joining the candidate sets of variables based on the structure of query graph.

In the existing graph-based query systems, the query processes focus on the cases of the edge label being constant in query graph. When the query involves variables of edge label, the node variables are joined at first, then generate the values of edge variables. This strategy often leads to relatively low efficiency because of lots of intermediate results. However, user opinion is an important knowledge in product knowledge, on which users often query. There are lots of opinion expressions in product graph, which are in the forms shown in Fig. 3a and b. The former describes two reviews express different opinions on the same product aspect, the latter describes two reviews contain same opinion on different product aspects.



(a) Different opinion on a aspect (b) Same opinion on different aspects

Fig. 3. Two forms of opinion data in product knowledge

When user try to find out all reviews including same opinion, the existing methods treat the product aspect and opinion separately, which would generate lots

Algorithm 1. Joining on a variable combination of product aspect and opinion

Input: Current table of intermediate results IRT,
Variable combination (e, v)
Output: Final table of intermediate results $nIRT$

```
1:  if the candidate set of v is empty then
2:      return nIRT;
3:  for each intermediate result r ∈ IRT do
4:      tmp = ∅, tmpv = ∅;
5:      for each ele ∈ r do
6:          if there is not edge between ele and e then
7:              continue;
8:          tmpv ← ele;
9:          generating the other candidate set S of v by ele;
10:         if tmp = ∅ then
11:             tmp ← S ∩ C_v;          /* C_v is the candidate set of v */
12:         else tmp ← S ∩ tmp;
13:     if tmp = ∅ then
14:         continue;
15:     for each ele ∈ tmp do
16:         generating the candidate set tmpe of edge ele;
17:         for each edge ∈ tmpe do
18:             generating the candidate set S by ele and edge;
19:         if tmpv ⊆ S then
20:             creating r's duplicate r_c and r_c ← (edge, ele);
21:             nIRT ← r_c;
22:  return nIRT;
```

of noneffective intermediate results and reduce the query process effectiveness. In reality applications, product aspect and opinion are often retrieved together. Then, we should treat the aspect variable of and the opinion variable as a combination for query processing. The detail of the *join* strategy based on the combination of aspect variable and opinion variable is shown in Algorithm 1.

5 Experiments

We evaluate our method on a large-scale product knowledge, which is constructed with the Amazon data and the artificial data. This dataset includes 116,174,460 triples, 11,979,407 entities, 478,626 products, 10,573 product aspects, 1,000,000 users and 10,566 opinion terms. We compare our approach with the state-of-the-art knowledge management system *gStore*. Four queries are executed to evaluate the query performances:

- Q1: Querying the products in the product category c_1, of which users hold the opinion term o_1 on product aspect a_1.
- Q2: Querying users' opinion and corresponding aspects of the product p_1.

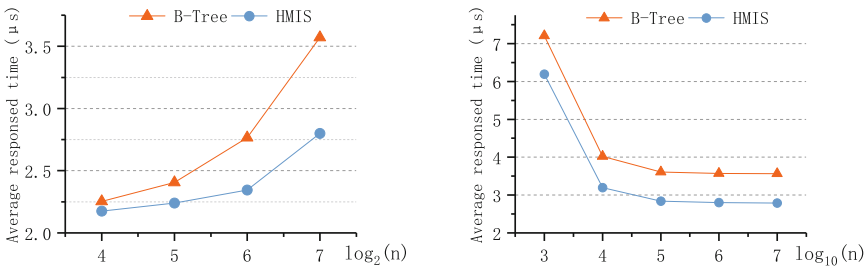
- Q3: Querying the count of product aspects users are more focused on and the products belong to category c_1.
- Q4: Querying the users who have bought the same product(s) and have the same opinion on certain product aspects as those of user u_1.

In the first experiment, we compare the query respond time of the proposed method and the gStore for the four queries Q1–Q4 above. As shown in Table 2, our method outperforms the gStore significantly on query respond time for the Q2, Q3 and Q4. That is because the proposed method avoids lots of useless intermediate results are avoided in the query processes. For the query Q1, our method's query performance is similar to gStore's. The main reason is that the proposed optimization strategy does not work for Q1, because Q1 does not involve the aspect variable and opinion variable.

Table 2. Query respond time

	Q1	Q2	Q3	Q4
gStore	290 ms	913 ms	>30 min	>30 min
Our method	287 ms	271 ms	2285 ms	2389 ms

In the following experiment, we verify the performance of the hybrid mapping index structure (HMIS). Firstly, we analyze the influence of B-Tree's order on respond time, where the count of querying URI is set to 100,000. As shown in Fig. 4a, We can find that HMIS takes less average respond time than B-Tree. Moreover, the performance of HMIS improves more significantly with the increase in B-Tree's order. The Fig. 4b shows that the average respond time of HMIS is less than that of the B-Tree for different numbers of random access.



(a) The order of B-Tree (b) The number of random access

Fig. 4. Performance comparisons of two index structures

Acknowledgements. This work is supported by National Natural Science Foundation of China (61562014), Guangxi Natural Science Foundation (2018GXNSFDA281049), the project of Guangxi Key Laboratory of Automatic Detecting Technology and Instruments (YQ17111), and the general scientific research project of Guangxi Provincial Department of Education (2017KY0195).

References

1. Hoffart, J., Suchanek, F.M., et al.: YAGO2: a spatially and temporally enhanced knowledge base from Wikipedia. Artif. Intell. **194**, 28–61 (2013)
2. Bollacker, K.D., Evans, C., et al.: Freebase: a collaboratively created graph database for structuring human knowledge. In: SIGMOD, pp. 1247–1250 (2008)
3. Neumann, T., Weikum, G.: The RDF-3X engine for scalable management of RDF data. VLDB J. **19**(1), 91–113 (2010)
4. Zou, L., Ozsu, M.T., Chen, L., et al.: gStore: a graph-based SPARQL query engine. VLDB J. **23**(4), 565–590 (2014)
5. Yu, J., An, Y., Xu, T., Gao, J., Zhao, M., Yu, M.: Product recommendation method based on sentiment analysis. In: Meng, X., Li, R., Wang, K., Niu, B., Wang, X., Zhao, G. (eds.) WISA 2018. LNCS, vol. 11242, pp. 488–495. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-02934-0_45
6. Kraska, T., Beutel, A., Chi, E.H., et al.: The case for learned index structures. In: SIGMOD, pp. 489–504 (2018)