Knowledge-Based Systems 75 (2015) 41–51

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Schema matching based on position of attribute in query statement

Guohui Ding [*], Tianhe Sun

Shenyang Aerospace University, China

ARTICLE INFO

Article history:
Received 21 April 2013
Received in revised form 11 June 2014
Accepted 9 November 2014
Available online 11 December 2014

Keywords:
Schema matching
Database integration
Query log
Ant Colony Optimization
Attribute position
Query statement

ABSTRACT

Attribute-level schema matching is a critical step in numerous database applications, such as DataSpaces, Ontology Merging and Schema Integration. There exist many researches on this topic, however, they all ignore evidences about the positions of attributes in query statements, which are crucial to find high-quality *matches* between schema attributes. In this paper, we propose a novel matching technique based on the positions of attributes appearing in the schema structure of query results. The positions of attributes in query results embody the extent of the importance of an attribute for the user browsing the query results. The core idea of our approach is to collect the statistics about attribute positions from query logs to find correspondences between attributes (*matches*). Our method works in three phases. The first phase is to design a matrix to record the statistics about attribute positions. Then, we employ two scoring functions to measure the similarities between collected statistics of two schemas to be matched. Finally, we employ a traditional algorithm to find the optimal mapping. Furthermore, our approach can be combined with other existing *matchers* to obtain more accurate matching results. An experimental study shows that our approach is effective and has good performance.

1. Introduction

Schema matching is an essential building block in sharing multiple heterogeneous data sources through a unified access interface. The basic issue of schema matching is to find attribute correspondences between attributes in source schemas and attributes in target schemas, namely *matches*. *Matches* are very significant for creating a unified mediated schema over multiple source schemas, exchanging data from one schema to another schema and sharing data in similar domains. Significant attention has been paid to this topic in literatures, and a rich body of techniques have been proposed by works [41,40,38,24,23,21,19,13,14,5,4].

The schemas to be matched are typically designed by different developers which have different habits and experiences, so they often have diverse structures and representations, and this makes schema matching difficult. In addition, dozens of tables and thousands of attributes in schemas also increase the difficulty of schema matching. Even with some availability of domain expertise, the task of schema matching may not be easy. As a result, schema matching continues to be a challenge, and to be a valuable research problem in practice.

There are a multitude of techniques in schema matching area, also called *matcher*, e.g., [38,24,23,19,14]. However, there are yet no perfect *matchers* that can return 100% accurate *matches*, and consequently, additional efforts are needed to be contributed to this area. In this paper, we proposed a novel matching technique that uses the positions of attributes appearing in the schema structure of query results to find the *matches*. As is well known, people are used to reading words in books from the left side to the right side, which is a habit that people capture information. For example, given a spreadsheet listing some records of phones, people always begin with the first column to read, then the second column, etc. This kind of habit can present some advices for developers who work on applications associated with the structured information and are designing a schema structure. That is, the developers should arrange the more important columns at the positions closer to the left side of the schema structure. For example, the column “phoneModel” in the above spreadsheet will be arranged in the left-hand side of the column “phonePrice”; as such, the column “departure time” is more likely to appear in the left side of the column “arriving time” in a railway timetable. Sometimes, a default ordering rule of a industry is also contained in the schema structure. We browse four websites selling mobile phones and search a phone of a specific brand, and the schema structures of their returned results are shown in Fig. 1. We can see that the attributes of schemas from different websites almost have the same order in their respective query results. Obviously, developers for the websites are more likely to arrange the attributes close to the left side according to the reading habit above. However, the reading habit is slightly useless for arranging the order of attributes close

* Corresponding author.
E-mail addresses: dinguohui@sau.edu.cn (G. Ding), suntianhe2013@sina.com (T. Sun).

http://dx.doi.org/10.1016/j.knosys.2014.11.005
0950-7051/© 2014 Elsevier B.V. All rights reserved.

to the right side, but they still have the similar order and the reason is the default ordering rules in a specific industry. Thus, the habits including reading habit and default rules, which are typically conformed by schemas to be matched in the similar domains, should be used to find *matches*.

As the discussion above, we can see that different attributes have different importance of structuring query results to be shown to final users; that is, an attribute will hold its own position in the schema structure of query results. As a result, we can regard the statistics about the positions of an attribute in a large number of query results as the identification of its semantics. Actually, a query result derives from a corresponding query statement in query logs. Consequently, our core idea is to collect the statistics about positions of attributes from the query logs to find *matches*. There are three phases in our approach. Firstly, we collect the statistics about positions of attributes by scanning queries in query logs of schemas to be matched, and the matrix, called feature matrix, is used to record the statistics. Secondly, three kinds of cardinality constraints for mappings are considered, which are one-to-one mapping, onto mapping and partial mapping. Based on the constraints, we employ two scoring functions to measure the similarities of feature matrices of different schemas. Finally, given the scoring functions, Ant Colony Optimization algorithm is used to find the final optimal attribute mapping. Our approach can be used as an auxiliary technique for current main matching techniques, such as matching based on text, and matching based on instances. If our approach is combined with other existing *matchers*, the accuracy of matching results will be improved significantly. This paper makes the following contributions:

1. We discover that the positions of an attribute in schema structures of query results contain some semantic information that can be used in schema matching.
2. We exploit the feature matrices to record the statistics about positions of attributes, which are collected from query logs.
3. We consider two kinds of scoring functions to measure the similarities of the feature matrices of different schemas.
4. We perform an extensive experimental study and the experimental results show that the proposed algorithm has good performance.

The rest of this paper is organized as follows. Section 2 introduces how to extract features of attribute positions. Section 3 discusses the scoring functions and the searching algorithm. The experimental results are given in Section 4. A brief related work is reviewed in Section 5. Finally, we conclude in Section 6.

2. Features extraction of attribute position

The first phase of our work is introduced in this section. Given two schemas to be matched, we will scan each query in logs to collect the statistics about positions of attributes. Then, we design two types of matrices to record the statistics about positions of attributes.

As our motivation shows, the positions of an attribute in schema structures can be seen as the identification of its semantics in the same schema. Thus, the information of the positions can be collected to perform schema matching. In what follows, we will discuss how to collect the statistics about the positions. First, we will consider the clause types in query logs. Only attributes in "select" clauses will be outputted as schema structures of final results. Thus, an easy way is to scan only the "select" clauses, and ignore the "where", "group" and "order" clauses. This method is simple, but deficient. As the discussion in Section 1, different attributes have different importance when showing information to users. For example, when your friend introduces a person to you, he or she first tells you the person's name rather than his or her height. Similarly, for a car you are not familiar with, what you first want to know may be the brand or the name of the car rather than its engine model. These attributes, like name and brand, are actually the major attributes, which have more important or general information of a given object, while other less important attributes (like height and color) are the minor attributes. Major attributes often receive more attention from people. They are customarily placed in the front position by developers as the main query conditions. For example, if a person is designing a user query interface in a website selling mobile phones, the combox (graphical user interface widget) of the condition "brand" will be placed above the combox of the condition "color". When buyers use this interface to find mobile phones, the "where" clause of the query generated by servers is more likely to be "...where brand = '...' and color = '...'" rather than "...where color = '...' and brand = '...'"; that is, "brand" will appear in front of "color". It can be seen that the positions of attributes in the "where" clause also reflect the importance of attributes to some extent. This behavior is consistent with our motivation in Section 1. The cases for "order" and "group" clauses are similar. Thus, we consider four types of clauses "select", "where", "order" and "group" for collecting features in the query logs. Second, we need to consider the types of queries except for the types of clauses. As in [14], three types of queries are considered. The first is called SPJ query that is the single-block query with Select, Project, Join and optional "group" and/or "order" clauses. The second is called SPJU query that contains multiple SPJ queries connected by the set operator "union". The third is called SPJS query that is SPJ queries with nested subqueries (type: SPJ, SPJU or SPUS). Before scanning queries, we introduce a concept *appearance sequence*. Given a query, the *appearance sequence* is a sequence that consists of attributes in the query. Meanwhile, the attributes in the sequence are sorted according to the order that the attributes appear in the query. We can see that each query corresponds to an *appearance sequence* which embodies the reading habit of people or some default ordering rules of a industry mentioned in Section 1. Base on this concept, we will discuss how to scan queries and collect the statistics.

Each query in the query log will be scanned. For the SPJ queries, we just extract an *attribute sequence* from a query. For the SPJU and SPJS queries, we decompose the queries into separate subqueries each of which is a SPJ query. The position of each attribute in an *appearance sequence* is recorded in a feature matrix. Given a log,

A	brand	model	color	timeMarket	appearance	videoCall	system	smart
B	brand	model	color	timeMarket	appearance	system	smart	cpuModel
C	brand	model	color	appearance	videoCall	system	smart	cpuModel
D	brand	model	color	timeMarket	appearance	system	smart	cpuModel

Fig. 1. Schema structures of query results from four E-commerce websites A–D.