Construction and Querying of Ancient Poet Knowledge Graph

Shuai Wang, Di Wang, Daikun Zhang and Fu Zhang*
School of Computer Science & Engineering, Northeastern University, Shenyang, Liaoning, 110819, China
*Corresponding author’s e-mail: neu414@163.com

Abstract. In recent years, with the development of knowledge graph techniques, how to construct various of knowledge graphs from different application domains has become important issues. This paper proposes a construction and querying approach for ancient Chinese poet knowledge graph. Firstly, the paper presents an approach for completing the construction of knowledge graph based on the techniques of knowledge acquisition, knowledge fusion, ontology generation and knowledge reasoning. The corresponding algorithms are provided. Then, the paper further investigates and realizes a query system for querying the poet knowledge graph through D3 data visualization query. Finally, several cases are carried out on the query system, and the results show that the approach and the prototype system are effective. The constructed ancient poet knowledge graph may be useful for users to query the poet information more accurately.

1. Introduction
With the continuous development of the information society and the rapid progress of search technology, people begin to expect to obtain the required information more quickly and accurately [1]. Knowledge graphs can provide more intelligent acquisition and management of information through more orderly and regular organization of knowledge for users [2-3]. As a new method of knowledge representation, the goal of knowledge graph is to describe the entities and concepts existing in the real world, as well as the relationship between the entities and concepts, and to capture and present the semantic relations between the concepts of the domain. Therefore, how to construct various of knowledge graphs from different application domains has become important issues [4].

Up to now, the number of recorded Chinese ancient poets is numerous, but there is no uniform and comprehensive way to get access to the information about these poets. Only when users need these information, do they have to search for relevant information from books or Internet, and the process of inquiry is not only wasting lots of time and energy but the results will also probably be tedious and fragmented or even wrong. This paper aims at applying the knowledge graph technology to describing Chinese ancient poets, and builds a knowledge graph of ancient Chinese poets, which can sort out and summarize the information of ancient poets in a more comprehensive way, and provide more convenient and accurate results for related queries.

In order to complete the construction of ancient poet knowledge graph and provide the query functions, this paper proposes a construction and querying approach for ancient Chinese poet knowledge graph. The paper first presents an approach for completing the construction of knowledge graph based on the techniques of knowledge acquisition, knowledge fusion, ontology generation and knowledge reasoning. Also, the corresponding algorithms are provided. Further, the paper investigates and realizes...
a query system for querying the poet knowledge graph through D3 data visualization query. Moreover, several cases are carried out on the query system, and the results show that the approach and the prototype system are effective. The constructed ancient poet knowledge graph may be useful for users to query the poet information more accurately.

The remainder of this paper is organized as follows. Section 2 gives the construction process of ancient poet knowledge graph, and provides several corresponding algorithms. Section 3 implements a prototype tool for querying poet information from the constructed knowledge graph. Section 4 shows conclusions and the future work.

2. Construction of ancient poet knowledge graph

This section gives the construction process of ancient poet knowledge graph and provides the corresponding algorithms. In order to construct the ancient poet knowledge graph, the paper uses top-down and bottom-up methods, which include the following several steps: (i) defining the basic class hierarchy and properties in the ontology; (ii) acquiring knowledge and making knowledge fusion and necessary data processing; (iii) enriching and expanding the ontology through knowledge reasoning.

2.1 Ontology definition

In order to construct an ancient poet knowledge graph, the paper first defines three basic classes, including poets, poems, and characters. Then the paper further adds a "create" attribute, its domain is the class "poet", and its range is the class "poetry". This attribute definition is the basis for knowledge reasoning. The rest of the attributes will be added in the knowledge acquisition process.

2.2 Knowledge acquisition and fusion

This paper acquires the poet information from the ancient poetry web and baidu encyclopedia. Firstly, the paper crawls the poets' names from the data collected in the "Ancient Poetry encyclopedia" in the ancient poetry web, and a total of 2094 poets are found. Then, the paper further crawls the poets' attributes. The method is as follows:

- investigating all the poets’ encyclopedia entry information, crawling all the attribute names that describe the poet's information, screening the attribute columns and selecting some of the most frequently occurring attributes, synthesizing synonyms with similar meanings, and finally obtaining the target attribute columns that need to be crawled;
- entering the relevant poet entry in the encyclopedia and traversing the poet's attribute list. If there is an attribute on the target attribute list, cleaning the dirty data under the attribute label, and saving the attribute and the attribute value to the dictionary describing the poet;
- storing the dictionary type data into the MongoDB database [5], and looping the steps above until the poet list is traversed. Since there are synonyms in the attributes of the crawl, the attributes are merged and finally 46 poet attributes are obtained.

Figure 1 shows a part of the poet information in the MongoDB visualization tool Robo 3T.
Secondly, the paper further crawls the poets’ kinships and poetries. The source of the poet's kinship is the content under the headings of the encyclopedia's corresponding poetry catalogue with the words "kinship", "relative members", "family members" or "family members". The hierarchy of kinship before crawling is done in the Protégé Ontology Editor [6]. Then, the target relationship attribute column is filtered by traversing the poet entries. The method of crawling poet kinship data is as follows:

- traversing all poet entries and crawling all title names in the entry directory. In the title list, if a title may describe the kinship relationship, it will be selected and a target title list is obtained;
- entering the poet entry, locating the content under the title in the target title list, and if the content is displayed as a table, extracting the relationship-name key-value pair. If it is not a table, skipping to the next step. Looping the steps above until the poet list is traversed.

Finally, the data is stored in the MongoDB database.

2.3 Ontology generation and query

This section further fills the acquired knowledge into the knowledge graph. Here the paper chooses RDFS [7] as the ontology description language. On this basis, the encapsulation of the SPARQL query statement is used to pave the way for the subsequent query system.

2.3.1 RDFS ontology. The generation of RDFS ontology relies mainly on the Jena open source toolkit, using the ontology model to create the ontology, and using the methods defined in the ontology model to operate the model based on the following steps:

i). creating ontology. Creating a ontology requires the use of a ModelFactory, a class that Jena provides to create various models. The class defines the member data for the concrete implementation model and more than twenty methods for creating the model. Here, the paper calls the read method in the ModelFactory to read the ontology file saved on the local disk.

ii). defining classes and subclasses. As mentioned above, the paper creates four basic classes. For the four basic classes, the paper calls the createClass() method to add the corresponding class to the ontology, and uses the addSubClass() method to define the subclass relationship. The key fragment of the algorithm for defining classes and subclasses is shown in Table 1.

| Table 1 A fragment of the algorithm for defining classes and subclasses |
|---|
| Input: four basic classes |
| Output: class hierarchy |

String NS = "http://someone/#";
//creating classes and subclasses, and defining relationships
OntClass person = ontModel.createClass (NS+"character");
OntClass poet = ontModel.createClass (NS+"poet");
OntClass poetry = ontModel.createClass (NS+"poetry");
//poet class is the subclass of the character class
person.addSubClass(poet);
person.addSubClass(emperor);

iii). defining attributes. Attributes can be divided into data property and object property. The role of a data property is to assign values to the class, such as adding a property "dynasty" for poets, the property type is string; an object property is to define the relationship between classes, such as adding the property "create" to the poet class and the poetry class, the relationship represents that poets create poetries. The key fragment of the algorithm for defining classes and subclasses is shown in Table 2.

| Table 2 A fragment of the algorithm for defining attributes |
|---|
| Input: four basic classes |
| Output: class hierarchy |

iv). adding individuals and property values. After defining classes and attributes, the paper finally adds the individuals and property values to the classes and attributes. The key fragment of the algorithm for adding individuals and property values is shown in Table 3.

v). generating ontology. Based on the steps above, an ontology is basically generated. Finally, the OntModel write function is called to generate the ontology file in the RDFS format. A fragment of the generated ontology is shown in Figure 2.
Table 2 A fragment of the algorithm for defining attributes

| Input: acquired poet data in section 2.2 |
|-----------------------------------------|
| Output: classes and their attributes    |

```java
String inpath1 = "C:\...\poet\poet_infos3.xls";
InputStream stream1 = new FileInputStream(new File(inpath1));
Workbook rwb1 = Workbook.getWorkbook(stream1);
Sheet sheet1 = rwb1.getSheet(0);
int rows1 = sheet1.getRows();
int column1 = sheet1.getColumns();
// add data properties
for(int j = 1; j < column1; j++){
    String property = sheet1.getCell(j, 0).getContents();
    OntProperty newProperty = ontModel.createOntProperty(NS+property);
    newProperty.setRange(XSD.xstring); 
}
// add object property "create"
OntProperty doProperty = ontModel.createOntProperty(NS+"create");
doProperty.setDomain(poet);
doProperty.setRange(poetry);
```

Table 3 A fragment of the algorithm for adding individuals and property values

| Input: acquired poet data in section 2.2 |
|-----------------------------------------|
| Output: RDF triples                      |

```java
//adding a poet's triple relationship
for(int i = 1; i < rows1; i++){
    Cell name_cell = sheet1.getCell(0, i);  //The name of each poet
    String personURI = "http://www.recshop.fake/cd/poet/" + name_cell.getContents();
    //creating a new individual for each poet
    Individual indi = poet.createIndividual(personURI);
    for(int j = 1; j < column1; j++){
        String property = sheet1.getCell(j, 0).getContents();  //data property
        String value =sheet1.getCell(j, i).getContents().replace("\n", "");  //property value
        if(value != ""){
            OntProperty newProperty = ontModel.getOntProperty(NS+property);
            indi.addProperty(newProperty, value);}}}
```
2.3.2 Ontology query with SPARQL. SPARQL [6] provides rules for querying RDF data. This paper uses the SPARQL query language to mainly accomplish the following four functions as shown in Table 4:

- **Entity query**: Enter the poet's name and return the poet's knowledge card.
- **Attribute query**: Enter the poet name and attribute name and return the attribute value.
- **Conditional query**: Retrieve the poet who satisfies the query conditions.
- **Poetry query**: Enter the poet name and the poetry name and return the verse.

| Input: RDFS ontology | Output: query result |
|----------------------|----------------------|
| //Entity query       |                     |
| String queryString = "select ?x ?y" + " Where" + "{" + "<http://www.recshop.fake/cd/poet/" + poetName + "> ?x ?y +"}"; |
| //Property query     |                     |
| String queryString = "select ?x" + " Where" + "{" + "<http://www.recshop.fake/cd/poet/" + poetName + "> <http://someone/#" + propertyName + ""> ?x " +"}"; |
| //Condition query    |                     |
| String queryString = "select ?x" + " Where" + "{" + "?x <http://someone/#" + propertyName + ""> " + "" + propertyValue + "" + "}"; |

QueryExecution qexec = QueryExecutionFactory.create(queryString, model);
ResultSet resultSet = qexec.execSelect();

Based on the query methods above, Figure 3 provides an example of the entity query.
2.4 Knowledge reasoning

The RDFS ontology establishes some basic model constraints on the basis of RDF, and defines the constraints on attributes in classes, properties, and values. Although some simple semantics can be expressed through RDFS, in more complicated scenarios, RDFS semantic expression ability is still weak, which lack many common features. For example, RDFS cannot declare certain features of an attribute, such as transitivity, functionality, symmetry, and declaring that one property is the inverse of another property. Therefore, this paper only conducts knowledge-based reasoning tests on the basis of simple rules on the ontology without extending the complex semantic rules. Here, taking Li Wei as an example, the inference rules can be constructed in Protégé Ontology Editor [6] as follow:

\[(\text{for rdfs:domain poet, rdfs:range poetry}) \land (\text{Li Wei creates poetry}) \rightarrow \text{Li Wei rdf: type poet}\]

In this way, Li Wei is the reasoning result of the poet.

3. Experiments

In this section, as a proof-of-concept for the proposed construction approach in section 2, the paper developed a prototype tool and carried out some query cases. In the following the paper briefly introduces the design and implementation of the prototype tool.

The prototype tool includes several modules, i.e., data initialization, layout, and drawing. The paper uses two ways to initialize the data, one is that all the poet node data is stored in JSON format and the edges are constructed on the basis of nodes; and the another one is initializing the necessary related nodes in the program, and then traversing and looping to create nodes and edges according to the result data returned by the SPARQL queries. Moreover, the paper provides the function of force-oriented graph layout as shown in Figure 4. In addition, once the layout definition is complete, one can draw the graph on the canvas, and typically need to draw elements such as nodes, lines, arrows, and text describing the nodes.
Note that, in Figure 4 a search button is placed in the middle of the page, and the search box is popped up by clicking the button. The query can be made by double clicking on the node or by entering a search command in the search box. Double-clicking on a poet's node allows the poet's entity query to be performed. In addition, the "collection" is the entrance to the collection of poems made by the poet, and the collection can be viewed by double-clicking the event, as shown in Figure 5. Among them, each node represents a poem, and the description text on the side identifies the index number of the respective node. By double-clicking on the poetry node, one can query the basic information of the selected poem, including the poem name, dynasty, author and verse.

![Figure 5 The visual poetry collection query example](image)

In general, the queries can be achieved by entering an instruction in the search box. Table 5 gives several common query forms.

| Function         | Instruction format | Example                  |
|------------------|--------------------|--------------------------|
| Entity query     | (poet name)        | Li Bai                   |
| Property query   | (name of the poet) (property name) | Li Bai's main achievements |
| Conditional query| (property) is the poet of (property value) | Gender is a female poet |
| Conditional query| (property) = (property value) | Gender = female |
| Poem query       | (Poet name), (poetry name) | Li Bai, Deng Jinling Phoenix Terrace |

Since the properties and property values entered by the user may be synonymous in the database, in our work the corrections in the program make the input properties and property values match the database as much as possible, but there is still a case where the matching is unsuccessful. Moreover, due to the addition of the function of poetry query in the visual query system, the huge amount of ontology data inevitably causes the impact on the query speed, and the link of the force-oriented graph exacerbates this influence. In our work, lots of tests have shown that it takes an average of 18 seconds to query and plot a force-directed graph containing 2095 nodes.
4. Conclusions
This paper proposed a construction and querying approach for ancient Chinese poet knowledge graph. The paper presented an approach for completing the construction of knowledge graph based on the techniques of knowledge acquisition, knowledge fusion, ontology generation and knowledge reasoning. The corresponding algorithms were provided. Then, the paper further investigated and realized a query system for querying the poet knowledge graph through D3 data visualization query. Finally, several cases were carried out on the query system, and the results showed that the approach and the prototype system are effective. The constructed ancient poet knowledge graph may be useful for users to query the poet information more accurately. In future work, the paper aims at considering and adding more poet information into knowledge graph, and providing more query functions.

Acknowledgments
The authors thank the anonymous referees for their very valuable comments and suggestions, which improved the technical content and the presentation of the paper. The work is supported by the Innovative Training Program for College Students.

References
[1] Berners-Lee, T., Hendler, J., Lassila, O. (2001) The Semantic Web. The Scientific American, 284 (5): 34-43.
[2] Pan, J.Z. (eds.) (2017) Exploiting Linked Data and Knowledge Graphs in Large Organizations, Springer International Publishing Switzerland, ISBN 978-3-319-45654-6.
[3] Liu, Q., et al. (2016) Knowledge Graph Construction Techniques. Journal of Computer Research and Development, 53(3): 582-600.
[4] Xu Z.L., et al. (2016) Review on Knowledge Graph Techniques. Journal of University of Electronic Science and Technology of China, 45(4): 589-606.
[5] Chodorow, K. (2010) MongoDB: The Definitive Guide. ACM.
[6] Glimm, B. (2011) Using SPARQL with RDFS and OWL Entailment. In: Reasoning Web. Semantic Technologies for the Web of Data. Springer Berlin Heidelberg. pp.1-38.
[7] RDF 1.1 Primer, W3C Working Group (2014), http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140225/