AUTOMATICALLY BUILDING A KNOWLEDGE-BASED Q&A SYSTEM

by

JINGLIN TAO

A dissertation submitted in partial fulfillment of
the requirements for the degree of

MASTER SCIENCE OF COMPUTER SCIENCE

WASHINGTON STATE UNIVERSITY
Department of Electrical Engineering and Computer Science

MARCH 2019

To the Faculty of Washington State University:

The members of the Committee appointed to examine the dissertation of JINGLIN TAO find it satisfactory and recommend that it be accepted.

Yinghui Wu, Ph.D., Chair

Assefaw Gebremedhin, Ph.D.

Larry Holder, Ph.D.

ACKNOWLEDGMENTS

I would like to thank ...

AUTOMATICALLY BUILDING A KNOWLEDGE-BASED Q&A SYSTEM

Abstract

by Jinglin Tao, M.S.
Washington State University
March 2019

Chair: Yinghui Wu

My abstract

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Dedication

dedicated to ...

CHAPTER 1. INTRODUCTION AND MOTIVATION

With the widespread collaborative spirit and the wide range usage of user-generated content, these concepts lead to the prosperity of wiki farms and make a great change to the ways of information retrieval and on-line education. Following the success and contribution of Wikipedia (January 15, 2001)[1], many other wiki hosting services were built in the past few years, like Wikia also known as Fandom (October 18, 2004)[2], Wikidot (August 1, 2006)[3], Huiji (March 2015)[4].

Compared with Wikipedia, like an encyclopedia, who has the DBpedia[5], a structured content collecting project from Wikipedia, many non-commercial and non-profit smaller wikis can go deeper into their specialty with more detailed information which can help the researchers find better supporting materials and open their minds with something they didn't know before. However, these wiki farms don't have enough

[1]https://en.wikipedia.org/wiki/Wikipedia

[2]https://www.fandom.com/

[3]http://www.wikidot.com/

[4]https://www.huijiwiki.com/

[5]https://wiki.dbpedia.org/

money and developers (many of them are volunteers) to build a better search engine for their database and do the maintenances, which affects their high-quality data collections not used to its full potential, especially for some wikis of literatures and arts including much terminology, like the search engine used by Huiji based on Elasticsearch[6], based on the Lucene library[7]. Elasticsearch is an excellent search engine, Lucene being a wonderful library as well, but somehow it still can't return the results we want, though the answers exactly appear in some pages. This situation becomes even worse when the size of wiki database grows sharply.

Facing the problem caused by an inefficient search engine and increasing data volume on those wiki farms, we decide to design another way for semi-structured data searching, combining the knowledge and skills learned in the past two years, which can also make some progresses in the Final Exam. These are the motivates and ideas the project begins with.

This work is called knowledge-based Q&A system, which consists of the technology with Natural Language Processing, Graph Discovering, Python programming, Subgraph Isomorphism and Data Visualization. The most important two aims are

- Running the project automatically, for they may not have enough developers

[6]https://www.elastic.co/

[7]http://lucene.apache.org/

to rebuild the search engine project; and

- Terminology friendly, which is essential to those literary wikis with fictional languages, neologism and other self-created words created decades ago but not having a widely use in the real world.

This paper contains 7 sections organized as follows. In the first Introduction and Motivation part, talking about the motivation of this project and what problem we need to be solved which are listed above. In the second part Standardize Graph Data, giving the structure of graph dataset model and the process of how to build it from semi-structured wiki data. In the third section NLP2SQL, explaining how to extract information from plain texts and understand natural language questions. In the forth section NLP2Graph, bringing in the subgraph isomorphism concept to do the Natural Language Processing graph searching. In the fifth chapter System Implementation, combining the work from second to forth parts and describing the frame of the whole project with flow charts. In the sixth chapter Experiment, giving several scenarios to show how the modules work together. The last one is the conclusion of the project and some future works.

CHAPTER 2. STANDARDIZE GRAPH DATA

In this part, we'll extract the wiki data from the public wiki API and design the graph structure from it.

For the original raw data, we choose the semi-structure data from Tolkien Gateway Website[8] also known as TG. This is a not-for-profit collaborative wiki which aims to collect and organize the works of J.R.R. Tolkien for fans and researchers, like a Tolkien encyclopedia, while Tolkien Estate holds the copyright over the literary texts[9]. What's more, Tolkien Gateway has become the largest and most well-built Tolkien-related encyclopedia on the World Wide Web since 2010 based on research by Mith[10], which gives [11]. It offers APIs[12] with documents in detail for users to craw data from its website. The APIs are a little bit different from the ones of Wikipedia,

[8]http://www.tolkiengateway.net

[9]https://en.wikipedia.org/wiki/Tolkien_Estate

[10]Noticing that LOTRO-Wiki on the top of the list with 86,649 articles is a game wiki exclusively "Lord of the Rings Online"-related. Thus, it isn't a good choice to do the research work based on a Massively Multiplayer Online Role-Playing Game set in Tolkien's Middle-Earth.

[11]LOTRO-Wiki, "Statistics" (retrieved 15 March 2019)

[12]Tolkien Gateway, "API" (retrieved 15 March 2019)

so we'll discuss them in the following subsections.

2.1 Problem Statement

From this subsection, we begin to state our problem definition. The data structure from Tolkien Gateway is like the one from Wikipedia. Raw data shared by GNU Free Documentation License[13] which we can get directly from request messages by using web crawlers through TG APIs and save the data into local text files for the next steps. Then, based on the paper of Zesch and Gurevych [2007], an article graph and a category graph can be set up from the raw data. As this search engine needs to meet the requirements of Graph discovery and natural language process, an undirected graph with labels and attributes is necessary to the experiment, which can also fit the structure of wiki pages (articles) with categories.

So, we define the input raw data as $D := (P, C)$, and the output graph as $G := (V, L, A, E)$. D represents raw data combined with P as wiki article/content pages and C as category pages. G refers to the undirected Graph we need to build from D to store these relationships, including V as vertices, a collection of nodes (article Ids); L as labels, a collection of nodes (category Ids), separated with V by different

[13] http://www.gnu.org/licenses/fdl-1.3.en.html

shapes and colors; A as attributes (option), a small set of nodes with attribute names described one article vertex $v \in V$ detailedly; E as edges, a group of relationships between nodes (four different kinds: V and V, L and L, V and L, V and A) (Newman [2003]).

2.2 Related Work

(unfinished)

Zesch and Gurevych [2007], taxonomy-like structure WCG and article graph **?**

2.3 Summary of Technique

To get the raw data from a wiki farm, clean the data and change the semi-structure files into a graph, we use these technique listed below.

2.3.1 Web Crawler Frame with Scrapy

Scrapy[14] is an open-source web-crawling framework which works with Python program using BSD License[15]. It can be used as a data extraction with APIs or as a general-purpose web crawler[16]. This framework can help us to get raw data from a wiki farm. The raw data can be set in many structures. Considering the convenience for next steps, the data is organized in Extensible Markup Language (XML). Scrapy also offers a demo frame which can help users build their own crawler.

2.3.2 Extract Data with Scrapy Selectors

Next, we need to extract data from the XML sources. BeautifulSoup is a popular web scraping python library though a little bit slow[17] and lxml is a XML parsing

---

[14]https://en.wikipedia.org/wiki/Scrapy

[15]https://en.wikipedia.org/wiki/BSD_licenses

[16]https://scrapy.org/

[17]https://www.crummy.com/software/BeautifulSoup/

library with ElementTree API[18] though not in Python standard library. Thus, we choose to use Scrapy Selectors for data extraction. Categories (id, name), category tree and articles (id, name, detail, inner link) can be stored after this step.

2.3.3 Output Category and Article Graph with igraph

Finally, we need a library to build an undirected graph with labels and attributes. igraph is an open source package with network analysis tools, well-known for its high efficiency and portability, supporting R, Python, Mathematica and C/C++.

2.4 Implementation

We use Scrapy demo frame to build three major crawlers to get the categories, articles and inner links betweens them. In Process of Standardizing Graph Data, the processes are explained in detail listed below.

---

[18] http://lxml.de/

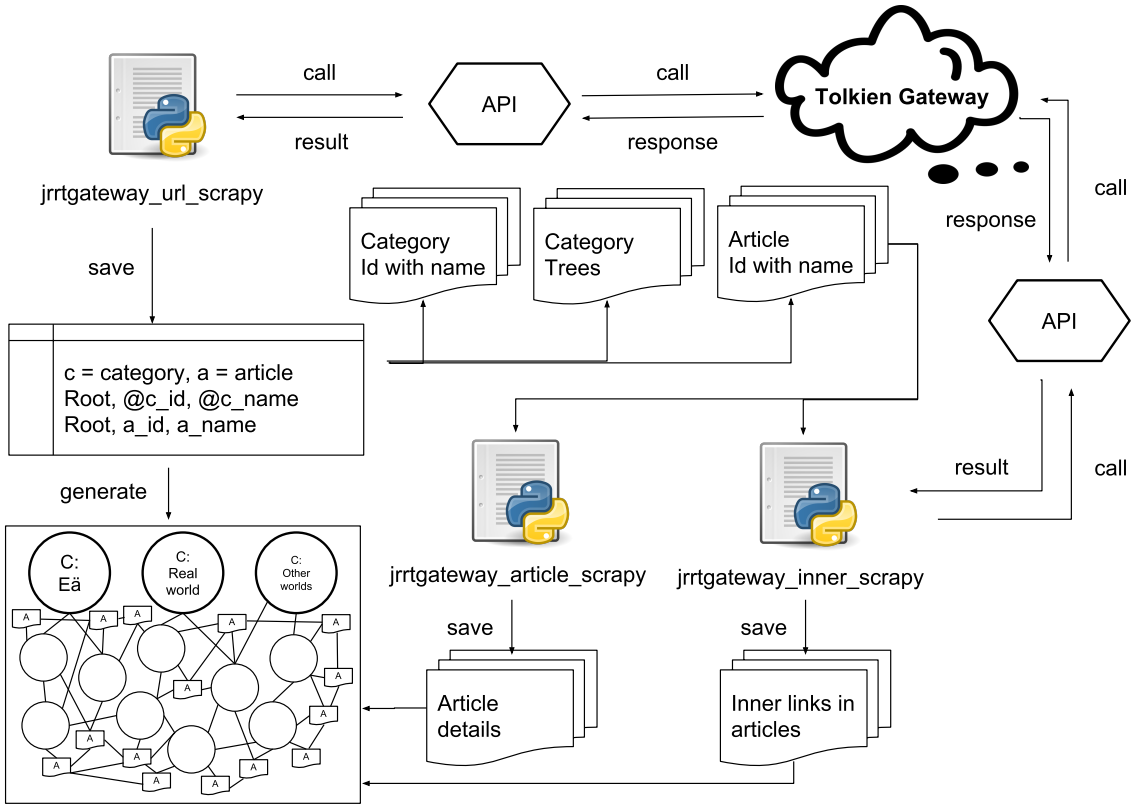


Figure 2.1: Process of Standardizing Graph Data

2.4.1 URL Crawler

This crawler is labeled as jrrtgateway_url_scrapy in Process of Standardizing Graph Data.

1. Focus three category roots ("E", "Real-world" and "Other fictional worlds") in this wiki. Take "E" as example. Set API parameters as action: "query", list: "categorymembers", cmlimit: 1000, format: "xml", cmtitle: "Category:E". Put "E", "Real-world" and "Other fictional worlds" in stack S and dictionary L.

2. Pop the last item i in the stack, change "cmtitle" to "Category:i" and run the crawler.

3. Extract data from XML files returned by API request and get children's ids and names (including categories, articles, files and other special pages). If a child is still a category, push the category name into the stack S, add category id to the key of L and set category name as value. If a child is an article, add article id to the key of V and set article name as value. Then, append the relationships between l and l or l and v to list E.

4. Repeat step 2 and 3, until stack S is empty.

2.4.2 Article Crawler

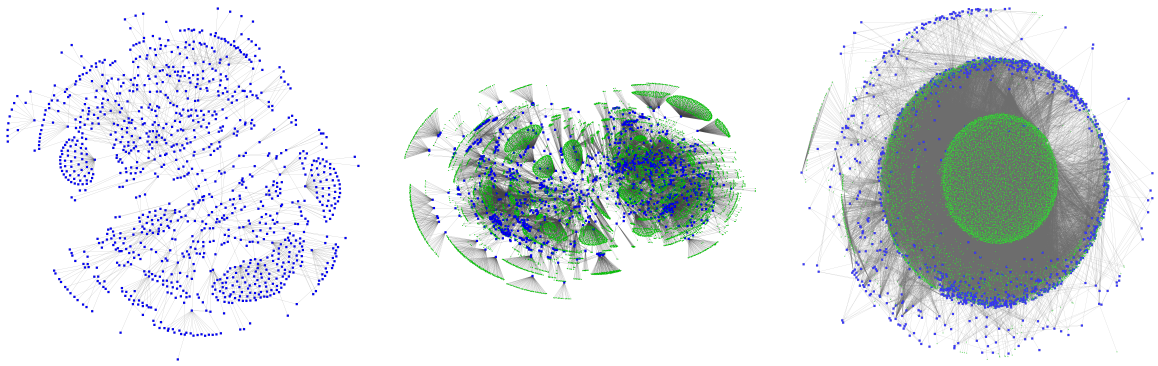
This crawler is labeled as jrrtgateway_article_scrapy in Process of Standardizing Graph Data.

1. Get article Ids K from V keys.
2. Set API parameters as action: "query", prop: "revisions", rvprop: "content", format: "xml", pageids: "k" and run the crawler.
3. Clean the detailed data and extract attributes a for each article.
4. Repeat Step 2 and 3, until get all article details and attributes.

2.4.3 Inner Links Crawler

This crawler is labeled as jrrtgateway_inner_scrapy in Process of Standardizing Graph Data.

1. Get article Ids K from V keys.
2. Set API parameters as action: "query", generator: "links", gpllimit: "max", format: "xml", pageids: "k" and run the crawler.
3. Extract inner link nodes and add the relationships between vertices to edges E.



(a) Category Graph **(b)** Article with Category **(c)** Category-Article Graph

Figure 2.2: Process of Standardizing a Graph from Raw Data

4. Repeat Step 2 and 3, until get all article inner links.

Finally, merge the clean data extracted from the results of three crawlers shown in Process of Standardizing a Graph from Raw Data.

2.5 Conclusion

(unfinished)

CHAPTER 3. NLP2SQL

In this chapter, we begin to set up the NLP module required in the Q&A system.

3.1 Problem Statement

Now we have a cleaned dataset which includes vertices V and their attributes A. Giving a better way to connect with the MySQL database of the wiki farm. Thus, we decide to change the natural language question to a SQL query. The relationships between V and A are changed into a relational database T. each attribute has its own table $t \in T$.

The graph with labels and attributes is still kept for the basic searching, giving the user an intuitive grasp of what they want. To enhance the comprehension of search engine, we annotate manually 100 questions with its corresponding SQL queries, like the WikiSQL from Yin et al. [2015].

3.2 Related Work

(unfinished)

Neural Enquirer: Learning to Query Tables in Natural Language Yin et al.

[2015]

Wikipedia-based Semantic Interpretation for Natural Language Processing Gabrilovich and Markovitch [2009]

fr2sql: Interrogation de bases de donnes en franais Couderc and Ferrero [2015]

3.3 Summary of Technique

3.3.1 SQLite with sqlite3

SQLite[19] can build lightweight disk-based SQL databases not requiring installing separate server process. However, SQLite is a C library which needs interface to interact with other programming language. Thus, we choose sqlite3[20] to finish this part of work. The sqlite3 module is a DB-API 2.0 interface for SQLite databases written by Gerhard Hring supporting Python language.

---

[19]https://www.sqlite.org/index.html

[20]https://docs.python.org/3.4/library/sqlite3.html

3.3.2 Seq2SQL

Seq2SQL is a deep neural network introduced in the paper Yin et al. [2015]. In this paper, the author gives a model of translating natural language questions to corresponding SQL queries. It is trained by the WikiSQL, a dataset of hand-annotated examples of questions with their corresponding SQL, which gives rewards to learn a better policy to generate the query. Also, Seq2SQL can offer an advantage of the SQL structure and simplify the original problem.

3.4 Conclusion

(unfinished)

CHAPTER 4. NLP2GRAPH

(consider to remove this chapter)

word phrase graph

4.1 Problem Statement

Input natural language questions

Output subgraphs

do subgraph isomorphism

4.2 Related Work

4.3 Summary of Technique

4.4 Conclusion

CHAPTER 5. SYSTEM IMPLEMENTATION

In this chapter, we'll combine the work of Standardize Graph Data and NLP2SQL. A data visualization task is also include in this chapter.

5.1 Frame of System

The flow chart The Structure of System Implementation illustrates the frame of whole project. The Q & A system is separated in three parts:

1. Data Process, including data crawling, data cleaning, data storing and data standardizing, referring to the work inStandardize Graph Data.

2. Search Engine, including subgraph generating, natural language processing and APIs with 3 databases, referring to the work inNLP2SQL.

3. Data Visualization, including user Interface, Q & A system module, APIs with search engine.

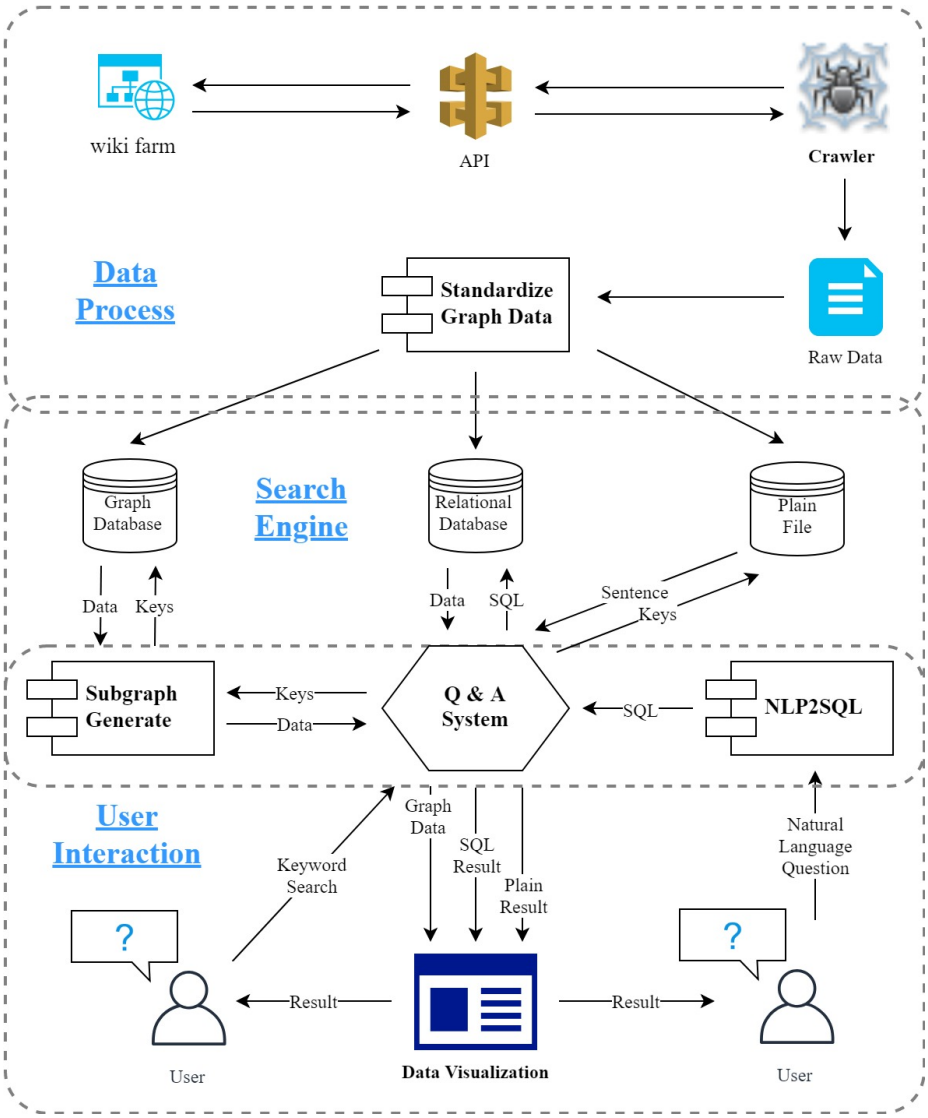


Figure 5.1: The Structure of System Implementation

5.2 Summary of Technique

We use Plotly[21] and Dash[22] for the data visualization task. They're modern analytics apps which can build beautiful web-based interfaces in Python.

At first, the project is built on Kivy with igraph.plot. However, the interaction between these two packages can't realize what we desire. Different purposes with different builders influence their compatibility. It's hard to build a dynamic graph which can interact with users. Thus, we decide to search for other tools to solve the data visualization tasks.

Scatter3d component is used for graph data plotting and user interaction. HTML components are used for listing text answers. Dropdown and input components are also used for user interaction.

5.3 Conclusion

(unfinished)

---

[21] https://plot.ly/

[22] https://plot.ly/products/dash/

CHAPTER 6. EXPERIMENT

(unfinished)

Set several scenarios below:

linked with a video url address: text

CHAPTER 7. CONCLUSION AND FUTURE WORK

In this dissertation ...

Bibliography

Benoît Couderc and Jérémy Ferrero. fr2sql: Interrogation de bases de données en français. In *22ème Traitement Automatique des Langues Naturelles*, 2015.

Evgeniy Gabrilovich and Shaul Markovitch. Wikipedia-based semantic interpretation for natural language processing. *Journal of Artificial Intelligence Research*, 34:443–498, 2009.

Mark EJ Newman. The structure and function of complex networks. *SIAM review*, 45(2):167–256, 2003.

Pengcheng Yin, Zhengdong Lu, Hang Li, and Ben Kao. Neural enquirer: Learning to query tables with natural language. *arXiv preprint arXiv:1512.00965*, 2015.

Torsten Zesch and Iryna Gurevych. Analysis of the wikipedia category graph for nlp applications. In *Proceedings of the Second Workshop on TextGraphs: Graph-Based Algorithms for Natural Language Processing*, pages 1–8, 2007.