Automatic Construction of Building Code Graph for Regulation Intelligence

Yu-Cheng ZHOU[1], Jia-Rui LIN[2], Zhong-Tian SHE[3]

[1]Graduate Student, Department of Civil Engineering, Tsinghua University, Beijing 100084, China, Email: zhouyc19@mails.tsinghua.edu.cn
[2]Asssistant Professor, Department of Civil Engineering, Tsinghua University, Beijing 100084, China, Email: lin611@tsinghua.edu.cn, **corresponding author**
[3]Undergraduate Student, Department of Civil Engineering, Tsinghua University, Beijing 100084, China, Email: shezt17@mails.tsinghua.edu.cn

ABSTRACT

Building codes are composed of a set of requirements that govern the design, construction, and maintenance of buildings and structures. Currently, most building codes are stored in unstructured text-based documents. However, with the advance of artificial intelligence, these unstructured building codes are no longer meet the requirements toward regulation intelligence scenarios such as design compliance review, relation analysis, and so on. To address this problem, this research proposes a method to automatically collect and formalize building codes and transform them into a knowledge graph representation. The method mainly consists of three steps: 1) data collection, which automatically collects building codes by crawling data in the web; 2) data structuring, which automatically transforms text-based building codes (e.g., HTML and PDF) into XML structure; and 3) graph generation, which transforms XML-based building codes into Neo4j graph database. The proposed method is implemented and tested in a case study. The result demonstrates the feasibility of the method and shows that the generated knowledge graph can support multiple regulation intelligence scenarios such as regulation relation retrieval, regulation conflict analysis, design compliance review, and so forth.

INTRODUCTION

The entire lifecycle of a built environment is governed by a variety of regulations, requirements, and standards (Nawari, 2018). The manual process of regulatory compliance checking is time-consuming, costly, and error-prone (Zhang & El-Gohary, 2017). Currently, most building codes are stored in unstructured text-based documents (Zhou & El-Gohary, 2016), and usually cite each other which forms a complex network of codes and standards. However, the massive and complex building code system brings significant challenges to building design, construction, and management. Meanwhile, with the advance of artificial intelligence (AI), these unstructured building codes are no longer meet the requirements toward regulation

intelligence scenarios such as design compliance review, relation analysis, and so on. Thus, retrieve and use the knowledge and experience contained in building codes efficiently and conveniently are of great significance.

In recent years, the advance of AI represented by deep learning integrates and promotes many industries. The exploration of regulation intelligence such as AI-enabled law systems (Liang et al., 2011) and automatic compliance review (Lin & Guo, 2020) are growing. However, there is still a lack of study on data structuring and standardized database development. At present, building codes are still stored and managed in form of text-based documents (e.g., PDF, text) with low retrieval and utilization efficiency (Solihin et al., 2017), which has become the obstacle and bottleneck of the upgrading of the regulation intelligence.

To address this problem, this research proposes a method to automatically collect and formalize building codes and transform them into the knowledge graph representation. The method mainly consists of three steps: 1) data collection, 2) data structuring, and 3) graph generation. By using the generated knowledge graph, the proposed method can support multiple regulation intelligent scenarios such as regulation relation retrieval, regulation conflict analysis, design compliance review, and so forth.

The remainder of this paper is organized as follows. Section 2 illustrates the methodology of this research. Section 3 describes the data collection method. section 4 explains the data structuring method based on XML. Section 5 illustrates the graph generation method based on Neo4j. Section 6 provides a case study for extracting relationships in a knowledge graph. Finally, section 7 concludes the paper.

METHODOLOGY

This research aims to propose an automatic building code graph construction method, which automatically collects and formalizes building codes and transforms them into a knowledge graph.

Figure 1 shows the overview of the research methodology. The method consists of three steps: 1) data collection, which automatically collects building codes by crawling data in the web; 2) data structuring, which automatically converts text-based building codes (e.g., HTML and PDF) to XML structure; and 3) graph generation, which transforms XML-based building codes into a graph database Neo4j. In the first step, Chinese building codes are collected by crawling, and in the last two steps, a subset of collected building codes are selected for the study.

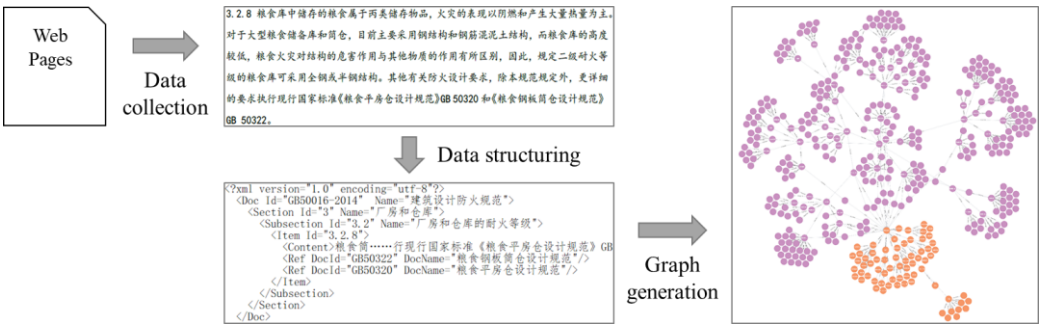


Figure 1. Overview of the research methodology.

DATA COLLECTION

Two types of building codes, HTML and PDF, are collected and transformed into plain text. Python language and several third-party packages are utilized for the automation. For HTML data sources, they are collected in websites (e.g., http://www.zhaojianzhu.com, https://www.soujianzhu.cn/Norm/) and parsed and transformed by BeatifulSoup package. For PDF data, they are read and transformed to plain text using pdfminer package. By crawling all links in the websites, webpages and documents of building codes are identified and collected.

In the data collection, some processes are applied to transform data into plain text and clean them. First, some meaningless or useless sentences are removed. For instance, texts in a webpage irrelevant to the building code, duplicated sentences, and garbled or unexpected texts. Second, figures and tables in building codes are collected and stored separately as files. In the main text, each place where a figure or table exists is replaced by the figure or table number surrounded by newlines. Note that parsing figures and tables are beyond the scope of this paper so the content in them will not be considered.

In the end, a total of 1560 building codes are collected and transformed into plain text. The collection process performs in less than 1 hour. The collected building codes have 112 MB of text and 1.82 GB of figures. Figure 2 shows an example of collected building codes.

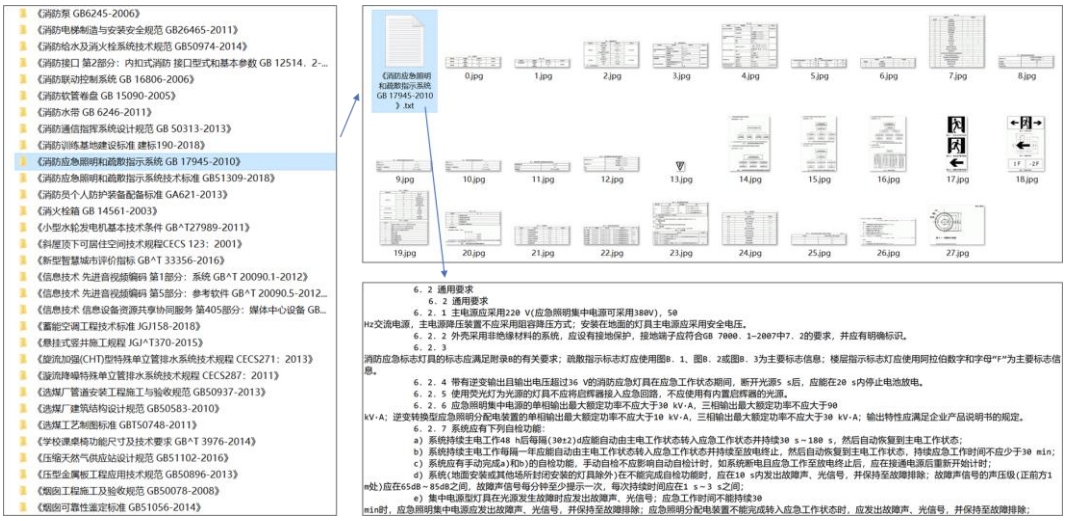


Figure 2. Example of collected building codes.

DATA STRUCTURING BASED ON XML

The first step in data structuring is preprocessing, which cleans and preprocesses the text for further processing. Two Python packages, linecache and re, are utilized. The linecache can cache the content of text files in memory to improve file reading efficiency, and the re package provides regular expression matching operations, which can match specific strings by user-defined rules. The preprocessing

mainly consists of two processes: 1) removing catalogs and descriptions in the text by pattern matching, and 2) clear blank lines, headers, footers, and other useless texts.

In order to realize the structured and formalized expression of building codes, the research develops an XML-based structure model, as shown in Figure 3. This model mainly includes four types of entities: document, section, subsection, clause (item), and reference. Among them, the document (Doc) includes attributes such as Name and ID and can contain several Sections. A Section can contain several Subsections or Items, and Subsection can contain several Items. An Item represents a regulation clause, and it is related to one or more Refs. A Ref contains the information of where the related Item is referred.

Therefore, based on the proposed XML-based structure model, this research uses regular expression-based patterns to recognize the corresponding metadata (e.g., document name, section name) in the texts of building codes. Considering the characteristics of the collected building codes, the document ID is represented by 2 letters and 5 digits with issue year (e.g., "GB50016-2014"), and the section and subsection ID are represented by the section serial number separated by "." (e.g., "3.2", "3.2.1"). In addition, the regular expression for recognizing issuing department, issuing date, etc. are also developed by analyzing the commonly occurred patterns such as prefixes or suffixes in a sentence. Thus, by using the extraction method, the regulation clauses, sections, etc., can be obtained and their relationships can be established.

Finally, based on extracted contents and relationships, the corresponding XML structure of a building code can be yield according to the following order: 1) output the basic metadata of a building code, 2) for each section in the building code and for each subsection in a section, output the regulation clause, and 3) output the reference relationships of regulation clauses.

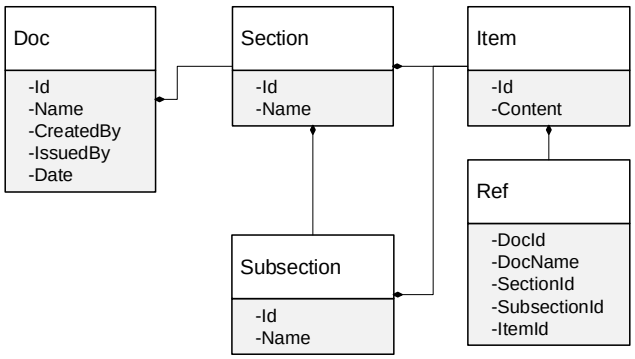


Figure 3. XML-based structure model for representing building codes.

GRAPH GENERATION BASED ON NEO4J

To achieve high efficiency and convince of information retrieval (query) and management for the structured building codes, the XML-based structured data is further transformed into a graph database. Neo4j, a high efficiency and open-source graph database implemented in Java (Fernandes & Bernardino, 2018), is utilized.

Figure 4 shows the graph structure model for representing building codes. In

this figure, the documents, (sub-) sections, and clauses are represented by nodes. The attributes of each node are consistent with the aforementioned XML model, the inclusion relationship (e.g., a subsection is included in a section) between these nodes is expressed by the "hasChild" attribute, and the reference relationship of clauses is expressed by the "refTo" attribute. In addition, considering the graph database can contain multiple building codes, the code_id is added to all nodes to indicate the ID of the belonging building code.

Based on the graph model, the graph generation consists of the following three steps. 1) Node creation. Visiting all elements in the XML document for each building code developed in the previous step, and generating Doc, Section, SubSection, and Item nodes in the meanwhile. 2) Inclusion relationship creation. Visiting all elements in the XML document, and creating hasChild attributes according to the inclusion relationship. 3) Reference relationship extraction. Visiting all elements in all XML documents, and creating refTo attributes by extracting reference relationships for each clause.

The graph generation is implemented based on Python packages py2neo, xml, and re. The xml package is used to parse XML files to obtain a structured tree-like representation, and the py2neo is used to import the generated graph data into the Neo4j database, which forms a knowledge graph.

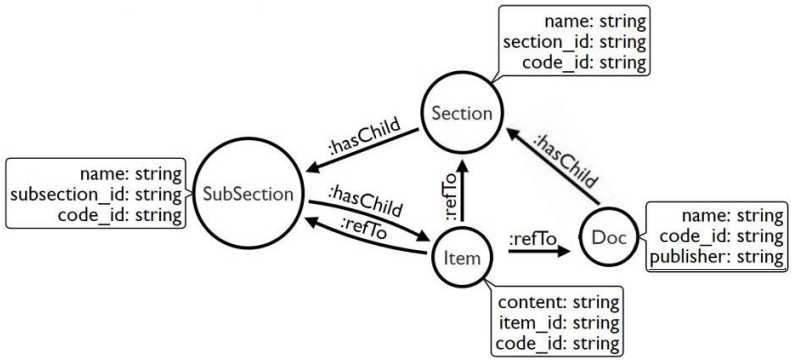


Figure 4. Graph structure model for representing building codes.

CASE STUDY: EXTRACTING RELATIONSHIPS IN KNOWLEDGE GRAPH

Based on the proposed method, a subset of collected building codes is selected for structuring and graph generation. The generated graph database finally contains 3964 nodes and 4377 relations (4047 inclusion relations and 330 references relations). To query the reference relationship of regulation clauses in a single document, the following sentence can be used:

```
MATCH p=(d:Doc)-[:hasChild*..3]->()-[:refTo]-()<-[:hasChild*..3]-(d)
                                RETURN p
```

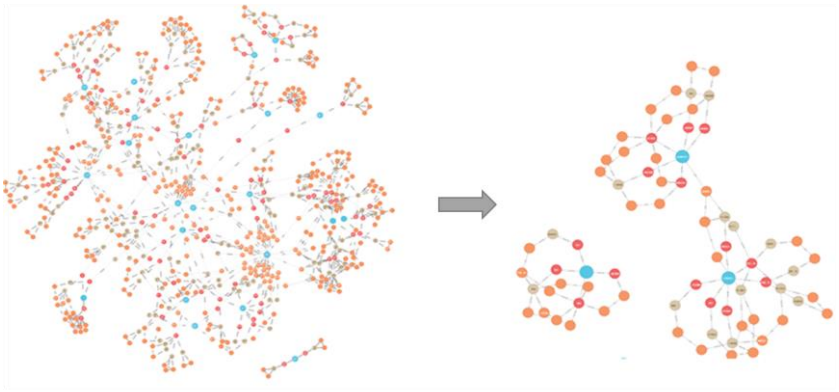
This match pattern in the query describes two nodes interconnected by "refTo" and both of them are linked to the same Doc node via "hasChild" relation with paths of length 3 or less. For more information about the query pattern syntax of Neo4j, the readers are referred to https://neo4j.com/docs/cypher-manual/current/syntax/patterns/.

Figure 5 shows a part of the generated knowledge graph and the result obtained from the query. As shown in the figure, the query result is a subset of the knowledge graph that satisfies the query matching pattern. Actually, by utilizing the power of the knowledge graph and flexible query patterns with high expressiveness, we can extract the data by describing the shape of the data we are looking for, which is very useful to analyze the structures and relations in the knowledge graph.

Generally, the reference relationships of regulation clauses cross different building codes grow rapidly as the number of regulation clauses increasing. The query of these reference relationships can support analyzing building code dependencies and discovering possible reference conflicts. To achieve this, the following query can be used:

```
MATCH p=(d:Doc)-[:hasChild*..3]->()-[:refTo]-()<-[:hasChild*..3]-(d1: Doc)
                        WHERE d<>d1 RETURN p
```

A part of the query result is shown in Figure 6. This result illustrates the feasibility of the method and the power of graph database in terms of information and relationship retrieval. The results also suggest that data stored in a knowledge graph can support multiple regulation intelligence scenarios such as regulation relation retrieval, conflict analysis, and design compliance review/checking. For instance, one can use the matching pattern to query nodes having specific relations to identify possible conflicts or extract compliance checking requirements.



a) b)

Figure 5. The knowledge graph and query result of relationships in a building code.

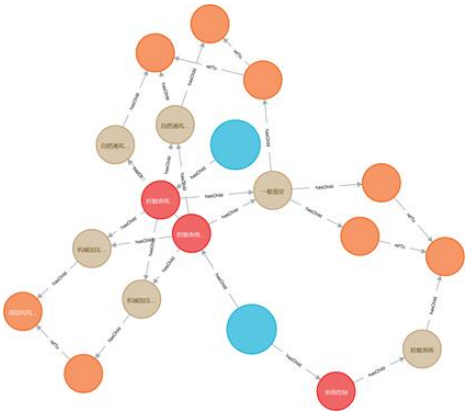


Figure 6. The query result of relationships in different building codes.

CONCLUSION

Currently, most building codes are stored in unstructured text-based documents, which are no longer meet the requirements toward regulation intelligence. To address this problem, this research proposes an automatic building code graph construction method consisting of three steps: data collection, data structuring, and graph generation. In the data collection, building codes in websites are collected in terms of HTML and PDF, and they are transformed into plain text. In the data structuring, text-based building codes are parsed by pattern recognition and converted into the proposed XML structure. Finally, in the graph generation, the XML-based structured data is transformed into Neo4j graph database. The experimental results demonstrate the applicability of querying specific and complex relations in a knowledge graph. The results also suggest that the knowledge graph of building codes can support multiple regulation intelligence scenarios such as regulation relation retrieval, regulation conflict analysis, design compliance review, and so forth. Comparing to conventional methods such as using a relational database, the proposed method has advantages of high expressiveness and flexible information retrieval.

Limitations in this research are also identified, which can be further improved by future work. First, the granularity in the graph construction is coarse. In the future, the construction of a more fine-grained knowledge graph can be studied, along with some advanced techniques such as natural language process, named entity recognition, and entity relationship extraction, to support a wider range of application scenarios. Second, the application scope of the proposed method needs to be further strengthened. To this end, more patterns or parsing methods in data structuring can be explored.

ACKNOWLEDGMENTS

The authors are grateful for the financial support received from the National Natural Science Foundation of China (No. 51908323, No. 72091512), and the Tsinghua University Initiative Scientific Research Program (No. 2019Z02UOT).

REFERENCES

Fernandes, D., & Bernardino, J. (2018). Graph Databases Comparison: AllegroGraph, ArangoDB, InfiniteGraph, Neo4J, and OrientDB. *Proceedings of the 7th International Conference on Data Science, Technology and Applications*, 373–380. https://doi.org/10.5220/0006910203730380

Liang, T., Li, B., & Zhao, G. (2011). The Summary of Environmental Protection Laws in Regulations Intelligence Retrieval System Research. *Shandong Chemical Industry*, *40*. https://doi.org/10.3969/j.issn.1008-021X.2011.09.012

Lin, J., & Guo, J. (2020). Automatic compliance checking in building and construction area. *Journal of Tsinghua University (Science and Technology)*, *60*(10), 873–879.

Nawari, N. O. (2018). *Building Information Modeling: Automated Code Checking and Compliance Processes*. CRC Press. https://doi.org/10.1201/9781351200998

Solihin, W., Dimyadi, J., Lee, Y.-C., Eastman, C., & Amor, R. (2017). The Critical Role of Accessible Data for BIM-Based Automated Rule Checking Systems. *Lean and Computing in Construction Congress - Volume 1: Proceedings of the Joint Conference on Computing in Construction*, 53–60. https://doi.org/10.24928/JC3-2017/0161

Zhang, J., & El-Gohary, N. M. (2017). Integrating semantic NLP and logic reasoning into a unified system for fully-automated code checking. *Automation in Construction*, *73*, 45–57. https://doi.org/10.1016/j.autcon.2016.08.027

Zhou, P., & El-Gohary, N. (2016). Domain-Specific Hierarchical Text Classification for Supporting Automated Environmental Compliance Checking. *Journal of Computing in Civil Engineering*, *30*(4), 04015057. https://doi.org/10.1061/(ASCE)CP.1943-5487.0000513