ISSN 1330-3651 (Print), ISSN 1848-6339 (Online) https://doi.org/10.17559/TV-20240212001325
Original scientific paper

Integration of Python Crawler and URL De-duplication Algorithm for Metrological Data Analysis

Xiaogang LUO*, Liang ZHOU

Abstract: In the field of network econometric data analysis, the analysis of massive URL data offers insights into the behavior of networks, the optimization of network structure, and the prevention of network attacks. Therefore, this study introduces Python web scraping technology to achieve data collection in the design of econometric data analysis software, and designs a hash split Bloom filter algorithm based on multiple eigenvalues to achieve de-duplication. The results demonstrated that the de-duplication time of the proposed algorithm was always lower than other algorithms. When the number of unified resource locators was 10,000, the de-duplication time was only 0.21 ms, which was 9.58 ms and 3.16 ms less than the other two algorithms, respectively. Meanwhile, the de-duplication accuracy of the research algorithm has always remained above 99.9%, reaching 99.95% when the number of unified resource locators was 35,000, which was 0.77% and 0.35% higher than the other two algorithms, respectively. In addition, the proposed algorithm had the lowest memory consumption. When the number of unified resource locators was 20,000, the memory consumption was only 0.28 MB, which was 1.18 MB lower than the Bloom filter algorithm. The research algorithms have shown outstanding performance in the field of network econometric data analysis, with high de-duplication efficiency, accuracy, and low memory consumption, providing reliable technical support for modern network econometric data analysis.

Keywords: bloom filter; HSDBF-ME; Python crawler; quantitative data analysis; URL deduplication

1 INTRODUCTION

The rapid development of the Internet has made data analysis an indispensable and important tool in various fields [1]. Quantitative Data Analysis (QDA), as an important branch, provides strong support and guidance for decision-makers through the collection, organization, analysis, and visualization of large amounts of data. However, in practical applications, due to the massive and dynamic nature of data on the Internet, data collection has become a challenging task [2]. Traditional manual data collection techniques require manual access to web pages, copying and pasting data, which is inefficient and prone to errors. Currently, obtaining information is mostly done manually, making it difficult to meet practical needs. Specifically, traditional manual data collection techniques have three main shortcomings. Firstly, the manual access of web pages, the copying and pasting of data, and the subsequent manipulation of this data requires a significant amount of time and effort. This can become cumbersome and inefficient when faced with large amounts of data. Secondly, manual operations are often limited by factors such as fatigue, resulting in a large number of errors. Thirdly, manual data collection techniques are inadequate for the purpose of large-scale data collection, and instead become a limitation on the scale and speed of data collection. Meanwhile, although traditional hash-based Uniform Resource Locator (URL) de-duplication techniques are simple and efficient, there may be issues with hash conflicts, leading to mis-identification of duplicate URLs [3]. When the URL is duplicated, parsing the same page multiple times results in wasted CPU resources, reducing the efficiency of the crawler system and wasting system storage space. Therefore, to improve the data collection efficiency of web crawling technology and further enhance the efficiency of metric data filtering and analysis, Python web crawling technology and improved URL de-duplication algorithms are introduced in the study. Firstly, the utilization of Python web scraping technology to simulate browser clicks and operations, and the subsequent acquisition of dynamically generated target URLs, represents a solution to the limitations of traditional collection methods. By employing techniques based on hash functions and Bloom Filters (BFs), the accuracy and efficiency of deduplication have been enhanced, thus resolving the issues associated with the collection, filtering, and analysis of voluminous measurement data. The main innovation of the research lies in simulating browser clicks and operations through Python to obtain dynamically generated target URLs, thereby solving the efficiency and accuracy issues of manual data collection. Meanwhile, using hash functions and BF methods can more accurately and efficiently remove duplicate URLs, improving the efficiency and quality of data collection. The research contributions are mainly divided into two points. One is to achieve good URL de-duplication effect while reducing the misjudgment rate of URL de-duplication and improving the misjudgment rate in data de-duplication. The second is to improve the efficiency of finding the required measurement data, solve the problem of screening and analyzing massive measurement data, and have broad application value in the field of network measurement analysis. The content mainly includes four parts. Part 1 provides an overview of URL recognition and Python. Part 2 introduces the design of QDA software based on URL algorithm and Python. The first section of the Part 2 proposes Python for QDA design. The second section proposes a Hash Split Distribution Bump Filter algorithm based on Multiple Eigenvalues (HSDBF-ME) to address the problem of duplicate URLs encountered by web crawlers when crawling websites. Part 3 mainly conducts experimental verification on the proposed technology. Part 4 discusses and summarizes the experimental results, and proposes future prospects.

2 LITERATURE REVIEW

URL detection has received a lot of attention from domestic and foreign researchers in ensuring the effectiveness and security of URLs. Luo S. et al. proposed a detection model based on neural networks to address the issue of SQLi attacks. This model consisted of a URL generator, a URL classifier, and a neural network model. This model could effectively detect malicious URLs and successfully classify them into different categories of SQLi attacks [4]. Zamir A. et al. designed a detection framework

based on a stacked model to address the problem of phishing attacks. By analyzing the characteristics of the phishing dataset and using multiple feature selection techniques, the model performed the best in classification accuracy, achieving an accuracy rate of 97.4% [5]. Shen M. and other researchers proposed an improved URL parsing strategy to enhance users' ability to correctly evaluate the host identity of URLs. This method could effectively improve the user's recognition rate of real URLs and reduce the misjudgment rate of confusing URLs [6]. Tan L's team has proposed a machine learning-based classification method for detecting and identifying malicious websites and URLs. By evaluating the performance of several well-known machine learning classifiers, this method could effectively improve the detection accuracy of malicious URLs and reduce the misjudgment rate [7]. Lee O. V. et al. discovered serious security vulnerabilities in the online world and developed a deep learning model to detect malicious URLs. It used the ISCX-URL-2016 dataset with over 110,000 URLs to evaluate the performance of the proposed detection algorithm. This deep learning algorithm has achieved significant accuracy performance in detecting malicious web pages [8]. Karve S. M. et al. investigated the significant cybersecurity threats posed by malicious URLs and studied methods for detecting malicious URLs using machine learning. They provided structured insights into various aspects and detailed analysis of harmful URLs for researchers and cybersecurity experts [9]. Tsai Y. D. et al. proposed an unbiased training strategy to enhance the performance of machine learning models in malicious URL detection, which can enhance deep learning-based models and alleviate the negative impact of biased features. The results demonstrated the strong generalization ability of this innovative strategy in the detection model [10].

Python crawler technology is widely used in data collection on the Internet, and many scholars have conducted research in this area. Yang S. and other scholars proposed a distributed web crawler system based on Python to address the problems of slow speed, low efficiency, and poor scalability of traditional single machine web crawlers. By analyzing the overall architecture of the system and the principles of each functional module, the system could effectively improve the speed and efficiency of web crawlers, solving the problems of traditional single machine web crawlers [11]. Hien N. L. H. et al. proposed a method of using Python language for web scrapers to meet the data extraction requirements of data factories. By stacking preferred web links, collecting required data, and storing it in CSV files for further reference, it was found that the Python language is very suitable for retrieving expected information from expected web sites [12]. Khan N's team discovered that the digital transformation of power companies has brought about explosive growth in information, so they proposed a data collection system based on tool libraries such as Python. This system could effectively extract valuable information from various sources and types of data [13]. Tao G. et al. proposed the use of web crawler tools for data acquisition and information extraction to meet the increasing demand for big data on the Internet and information retrieval. Python-based web crawling technology could effectively improve the accuracy and efficiency of information retrieval, providing strong support for various fields [14]. Osanyin Q. A. and other scholars have proposed a Python-based semi-automatic web crawler program for ICANN, an important global organization for domain name management and IP address allocation, to collect regular announcement information on its official website. This program could resolve the association relationships between hot security topics through association rules [15].

In conclusion, current research has demonstrated the efficacy of URL detection and Python crawling techniques in the detection and collection of network data. These techniques have the potential to enhance network data security and processing capabilities. However, the problem of large amount of data and insufficient memory in the Internet still exists. The algorithm for de-duplication also needs to be optimized. This study mainly introduces advanced URL de-duplication methods to achieve effective URL de-duplication to achieve more optimized QDA software design.

3 RESEARCH METHODOLOGY

Crawling technology and URL de-duplication play an important role in QDA. This chapter focuses on the design of QDA software. Firstly, Python is proposed for data collection in the Data Extraction Module (DEM), including web crawling of static and dynamic web pages. Subsequently, an improved HSDBF-ME algorithm is designed to address the issue of URL duplication in crawling to reduce false positives and improve the de-duplication effect.

3.1 Data Collection Based on Python

In QDA design, the DEM is very important. The function of this module is to collect and extract data from various data sources for subsequent analysis and modeling. The main goal of DEM is to obtain high-quality, complete, and accurate data for QDA. In the past, data extraction often required manual extraction of data from various sources (such as databases, websites, APIs, etc.) one by one, which was very cumbersome and time-consuming [16, 17]. To improve the efficiency and accuracy of data extraction and achieve automated data acquisition, Python is introduced in this study. Python mainly utilizes web crawlers to automatically retrieve data from the internet. By writing Python scripts, it is possible to automate accessing web pages, parsing web content, extracting required data, and saving it locally or in a database. The process of Python crawler is shown in Fig. 1.

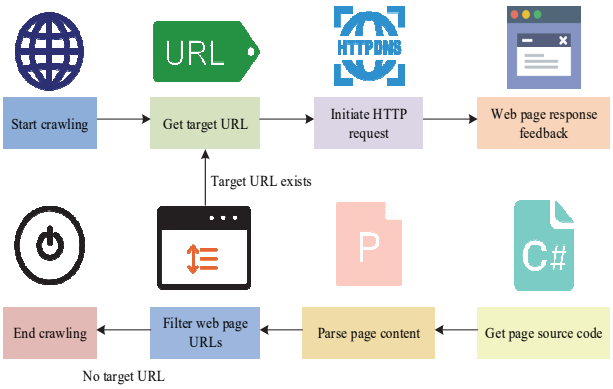


Figure 1 Python flow chart of the crawler

In DEM, it is necessary to complete the collection of web page data. Among them, web page formats include two types: static web pages and dynamic web pages. Static web pages are relatively simple, without the addition of Java. When writing, there is no need to consider hiding content in the source code, and the URL of the web page has a certain regularity, making it convenient for updating the URL queue and automatic loop crawling. Dynamic web pages are more complex, and due to the existence of Java, it is necessary to use third-party libraries that recognize Java to write code [18, 19]. In static web page data collection, it is necessary to find the URL naming convention of the static web pages to be crawled to automatically update the URL queue and loop through all web pages that comply with the convention. Due to the fact that numerical changes in the URL can cause changes in the entire webpage URL, this study sends requests through the urllib module in Python and uses a loop structure to change the numerical content in the URL for loop crawling. At the same time, to solve the problem of IP restrictions, it is necessary to add a delay code before sending requests, increasing the delay by 0.5 seconds each time to prevent IP restrictions caused by a large number of accesses in a short period of time. After the webpage responds, it will obtain a parseable webpage source code, which needs to be parsed in a timely manner to crawl the source code of the next URL webpage. After obtaining the source code of the first page and parsing it, it is necessary to close the webpage request to release system resources. The static webpage crawling process diagram is Fig. 2.

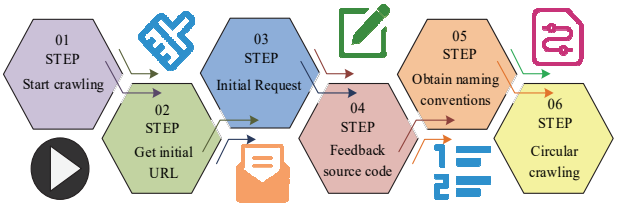


Figure 2 Flowchart of a static web page crawl

After completing the crawling of static web pages, a real-time monitoring program is introduced to update the URL queue and database content in a timely manner. The specific approach is to add conditional judgment code during the loop process and remove useless URLs. A real-time monitoring program is designed by comparing the total number of URLs captured currently with the total number of URLs captured in the previous time period. Real time monitoring of static web pages is achieved by adding new URLs to the queue and updating database content in a timely manner. In the data collection of dynamic web pages, due to the inability to find a corresponding naming pattern for URL URLs, dynamic web pages cannot send requests to web pages through the urllib module in Python. Therefore, after analyzing the source code of dynamic web pages, this study adopts a simulation browser click method to directly flip pages in the web page to obtain new target URLs and achieve the goal of loop crawling. Due to changes in the URL retrieval method, a third-party library from Selenium is selected for writing. Selenium mainly solves the problem of inability to execute JavaScript code in web page requests. Sending requests to dynamic web pages is achieved by simulating Google browser access through the web-driver module in the selenium library. At the same time, after sending the request, it is necessary to write a delay program to wait for network resources to load, and after a delay of 0.5 seconds, crawl the webpage source code. After the source code crawling is completed, it is necessary to change the URL for crawling, and simulate the browser through Selenium to perform the corresponding dropdown and click on the corresponding webpage part. On this basis, loop operations can obtain the entire URL queue, and repeat crawling until the dynamic webpage crawling is completed. The dynamic webpage crawling process is Fig. 3.

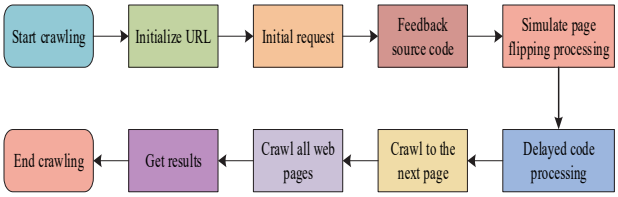


Figure 3 Dynamic web page crawl process

3.2 URL De-duplication Algorithm Based on BF

When crawling websites, crawlers will encounter a large number of URLs. Due to the large number of URLs, it is likely to catch duplicate URLs, resulting in resource waste [20]. To avoid this situation, it is necessary to make a judgment before crawling the URL. If the URL has already been crawled before, discard it, and this process is called URL de-duplication. For URL de-duplication, this study introduces the BF algorithm. BF consists of a bit vector and a set of hash functions. Assuming the length of the bit vector is m and all bits are 0 at the beginning. K represents the number of hash functions, $h()$ represents the hash function, and the set to be filtered is $\{x_1, x_2, ..., x_s\}$, which contains s elements [21]. BF maps each element in the set to a specific position in the vector through k hash functions. Each element x in the set is mapped through k hash functions to change the value of the corresponding position on the bit vector from 0 to 1. If it is found that the bit is already 1 during the setting operation, no more setting will be performed and the bit will be kept as 1. The probability calculation of a bit vector remaining 0 after the first hash mapping is Eq. (1).

$$P = 1 - \frac{1}{m} \tag{1}$$

When there are s elements in the set to be operated on, and each element needs to be hashed k times, the probability of a bit vector remaining 0 after hashing all elements is calculated as shown in Eq. (2).

$$P = \left(1 - \frac{1}{M}\right)^{ks} \tag{2}$$

After converting from the natural logarithm formula, the probability calculation is Eq. (3).

$$p \approx \mathrm{e}^{-\frac{ks}{m}} \tag{3}$$

On the basis of having already performed a hash operation on s elements, when a new element y arrives, if the position of the corresponding position vector has been set to 1 after k hash operations, then element y will be mistakenly considered to exist in the set, which is a misjudgment. The calculation of misjudgment rate is Eq. (4).

$$f=(1-p)^k \approx \left(1-\mathrm{e}^{-\frac{ks}{m}}\right)^k \tag{4}$$

In Eq. (4), there is a direct relationship between the misjudgment rate f and the number of set elements s, the size of BF bit vectors m, and the number of hash functions k. To minimize the misjudgment rate, this study first discusses the number of hash functions k in BF, and the calculation of k is Eq. (5).

$$k \approx -\frac{m}{s}\ln(p) \tag{5}$$

Due to $0 \le p \le 1$, Eq. (6) can be obtained.

$$\ln(p)\ln(1-p) > 0 \tag{6}$$

When $\ln(p)\ln(1-p)$ is maximum, the calculation of the minimum value of f is Eq. (7).

$$f=\left(\frac{1}{2}\right)^k \approx 0.6185 \text{ m/s} \tag{7}$$

Therefore, to maintain a low misjudgment rate, the size of the bit vector should be controlled below 50%. However, in practical applications, due to the gradual increase in the number of URL links, it is not possible to determine the size of the collection in advance. This may lead to the subsequent addition of elements being misjudged as already present in the collection, thereby having a certain impact on the crawler results [22]. To address this issue, this study proposes the HSDBF-ME algorithm. This algorithm uses multiple eigenvalues and elements to be filtered for hash mapping, and reflects the mapping results in the modification of bit vectors. The schematic diagram of multi eigenvalue BF mapping is Fig. 4.

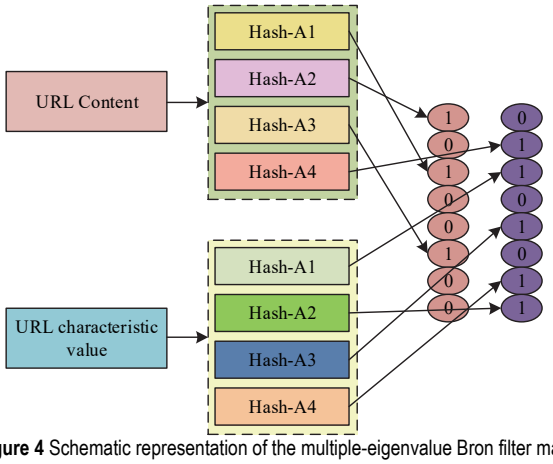


Figure 4 Schematic representation of the multiple-eigenvalue Bron filter map

In Fig. 4, this study mainly selects attributes or eigenvalues other than the element itself to form a multi-feature BF with the element to be filtered based on its properties and characteristics. The algorithm selects d feature values. In URL de-duplication, in addition to the URL content itself, the difference field of the URL is selected as another feature value to form a multi-feature value BF together with the URL. The schematic diagram of Hash splitting BF mapping is Fig. 5.

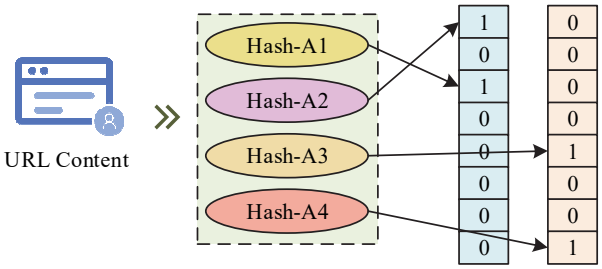


Figure 5 Schematic diagram of Hash split filter mapping

In Fig. 5, Hash function mapping splitting is aimed at the traditional BF algorithm's hash function group splitting. The original set of hash functions is divided into several groups, with each group mapping to the same bit vector. The split set of hash values is then mapped onto the same set of bit vectors. Assuming there are k hash functions, these k hash functions are split into L groups. The k/L hash mapping values of each group correspond to a bit vector, which requires L bit vectors. Compared to other techniques used in BF such as K-classification and complete combination, this method combines multiple eigenvalues and hash splitting. A certain degree of precision sacrifice reduces spatial complexity, overcomes the drawbacks of high spatial costs, and achieves higher spatio-temporal efficiency and lower misjudgment rates. In the misjudgment rate analysis of the HSDBF-ME algorithm, the probability of each element in the bit vector remaining 0 after the first hash mapping is calculated as shown in Eq. (8).

$$p=1-\frac{1}{mL} \tag{8}$$

In Eq. (8), L represents the number of bit vectors. When there are s elements in the set to be operated on, and each element needs to be hashed k times, and the probability of a certain bit vector remaining 0 after all s elements have been hashed is calculated as shown in Eq. (9).

$$p=\left(1-\frac{1}{mL}\right)^{ks} \tag{9}$$

After converting from the natural logarithm formula, the probability calculation is Eq. (10).

$$p \approx \mathrm{e}^{-\frac{ks}{mL}} \tag{10}$$

The improved algorithm calculates the misjudgment rate of a feature value as shown in Eq. (11).

$$f = (1-p)^k \approx \left(1 - e^{-\frac{ks}{mL}}\right)^k \quad (11)$$

The calculation of the misjudgment rate for d eigenvalues is Eq. (12).

$$f = (1-p)^k \approx \left(1 - e^{-\frac{ks}{mL}}\right)^{dk} \quad (12)$$

In the improved algorithm, the number of bit vectors is determined by the number of eigenvalues and the number of hash groups L. Each group of hash functions after grouping corresponds to a bit vector, that is, each eigenvalue corresponds to L bit vectors. Therefore, the bit vectors corresponding to d eigenvalues have d L groups. Compared to the basic BF algorithm, the improved HSDBF-ME algorithm successfully reduces the false positive rate f by increasing the number of eigenvalues and splitting the hash function, thereby improving the accuracy of determining whether elements exist in the set. The specific process of the HSDBF-ME algorithm is Fig. 6.

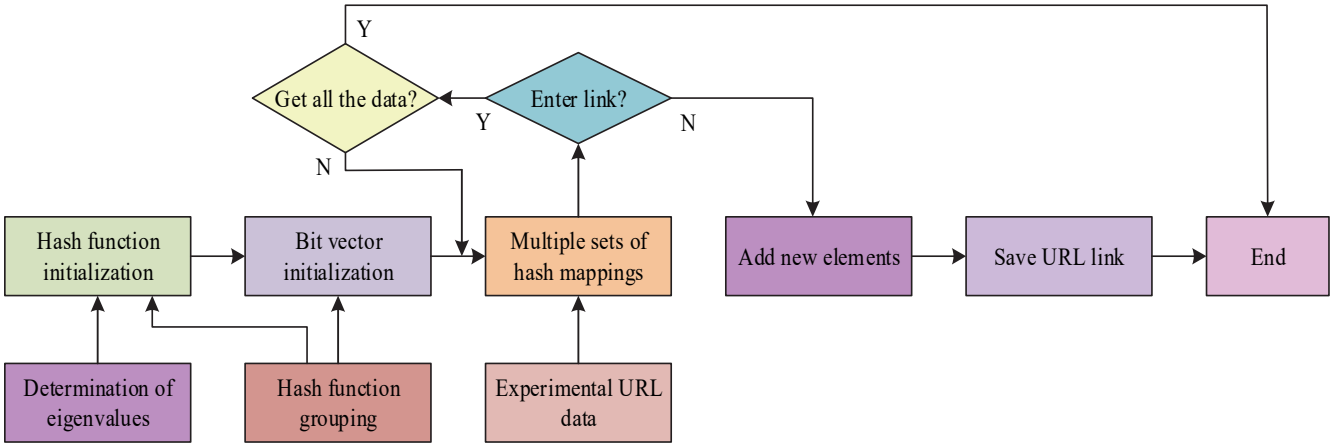


Figure 6 The specific flow process of the HSDBF-ME algorithm

4 RESULTS AND DISCUSSION

This chapter mainly conducts experimental analysis on the proposed HSDBF-ME algorithm and Python. Firstly, the anti-misjudgment performance of the HSDBF-ME algorithm is verified, and the misjudgment rates of each algorithm are analyzed based on the hash function value, bit vector size, and dataset size. Then, the time consumption and number of web pages crawled by different web crawling techniques are compared. Finally, the de-duplication performance of the HSDBF-ME algorithm is verified.

4.1 Analysis of Misjudgment Rate of URL

To verify the effectiveness of the proposed algorithm, a series of experiments are conducted in this study. Tab. 1 shows the parameters of the experimental environment.

Table 1 Setting of the experimental environment

Environment	Parameter setting
System	Windows 10 64 Bit operating system
Processor	Intel（R）Core（TM）i5-7200U CPU @2.50GHz 2.71GHz
Hard disk	240
JDK	JDK9.0.4
IDEA	IntelliJ IDEA Community Edition 2017.3.4
Language	Java

To verify the advantages of the HSDBF-ME algorithm in terms of misjudgment rate, this study conducted experimental comparisons between the basic BF algorithm and common multi-dimensional BF algorithms. Meanwhile, 20,000 obtained webpage link data were selected as experimental data, of which approximately 4,326 were duplicate URLs. The experiment would input these data into three different algorithms, observe and record their misjudgment rates. The experiment first verified the misjudgment rates of each algorithm under different hash function values k, where k values were 2, 4, and 6, respectively. To ensure the accuracy of the experiment, two tests were conducted and the average value was taken as the final result. The comparison results of the misjudgment rates of various algorithms under different k values are shown in Fig. 7. The HSDBF-ME algorithm achieved the lowest false positive rate at different k values. In Fig. 7a, in the first test, when k was 2, the misjudgment rate of the HSDBF-ME algorithm was only 0.51 $\pm$ 0.21%. The misjudgment rates of the basic BF algorithm and the multi-dimensional BF algorithm were as high as 4.24 $\pm$ 0.72% and 2.01 $\pm$ 0.73%, respectively. When k was 4, the misjudgment rate of the HSDBF-ME algorithm was only 0.28 $\pm$ 0.11%, which was reduced by 3.53% compared to the basic BF algorithm. When k was 6, the misjudgment rate of the HSDBF-ME algorithm was only 0.32 $\pm$ 0.23%, which was reduced by 1.42% compared to the multi-dimensional BF algorithm. In Fig. 7b, in the second test, the misjudgment rates of the HSDBF-ME algorithm at k = 2, 4, and 6 were only 0.81 $\pm$ 0.24%, 0.47 $\pm$ 0.25%, and 0.49 $\pm$ 0.19%, respectively, which were lower than those of the basic BF algorithm and the multi-dimensional BF algorithm. At the same time, when k was 4, the error rate of each algorithm was the lowest. The HSDBF-ME algorithm was significantly superior to other algorithms, and the optimal k value was 4.

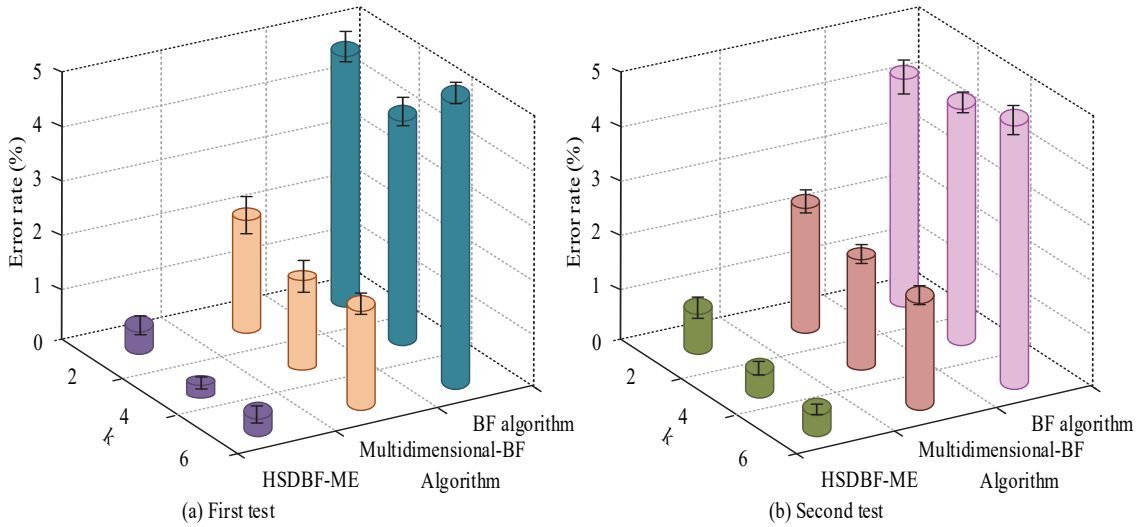


Figure 7 Miscalculation rate of each algorithm at different Hash function values *k*

The experiment then verifies the misjudgment rates of various algorithms under different bit vector sizes *m*, where the values of *m* are 20,000, 50,000, and 80,000, respectively. At the same time, the experiment is conducted twice and the average value is taken as the final result. The comparison results of the misjudgment rates of various algorithms under different bit vector sizes m are shown in Fig. 8. The HSDBF-ME algorithm achieves optimal results at different m values. In Fig. 8a, when *m* is 20,000, the misjudgment rate of the HSDBF-ME algorithm is only 8.33%. The misjudgment rates of the basic BF algorithm and the multi-dimensional BF algorithm are as high as 52.25% and 35.51%, respectively. In Fig. 8b, when *m* is 50,000, the misjudgment rate of the HSDBF-ME algorithm is only 4.51%, which is reduced by 35.91% and 41.87% compared to the basic BF algorithm and multi-dimensional BF algorithm, respectively. In Fig. 8c, when *m* is 80,000, the misjudgment rate of the HSDBF-ME algorithm is only 4.32%, which is 15.74% less than the multi-dimensional BF algorithm. Therefore, the misjudgment rate of each algorithm decreases with the increase of *m* value, and when *m* is 80,000, the misjudgment rate of each algorithm is the lowest. This indicates that the HSDBF-ME algorithm has significant performance advantages, and the optimal *m* value is 80,000.

The experiment continues to explore the misjudgment rates of various algorithms on different dataset sizes, where *n* values are 5,000, 10,000, and 20,000, respectively. At the same time, the experiment is conducted twice and the average value is taken as the final result. The comparison results of the misjudgment rates of various algorithms on different datasets n are shown in Fig. 9. The HSDBF-ME algorithm still achieves the optimal misjudgment rate at different *n* values. In Fig. 9a, when *n* is 5,000, the misjudgment rate of the HSDBF-ME algorithm is only 2.15%. The misjudgment rates of the basic BF algorithm and the multi-dimensional BF algorithm are as high as 9.27% and 4.16%, respectively. In Fig. 9b, when *n* is 10,000, the misjudgment rate of the HSDBF-ME algorithm is only 3.04%, which is reduced by 19.86% and 9.73% compared to the basic BF algorithm and multi-dimensional BF algorithm, respectively. In Fig. 9c, when *n* is 20,000, the misjudgment rate of the HSDBF-ME algorithm is only 7.28%, which is reduced by 43.05% and 27.69% compared to the other two algorithms. Therefore, the error rate of each algorithm gradually increases with the increase of n value, while the HSDBF-ME algorithm has the lowest increase in error rate, once again proving the superiority of the HSDBF-ME algorithm.

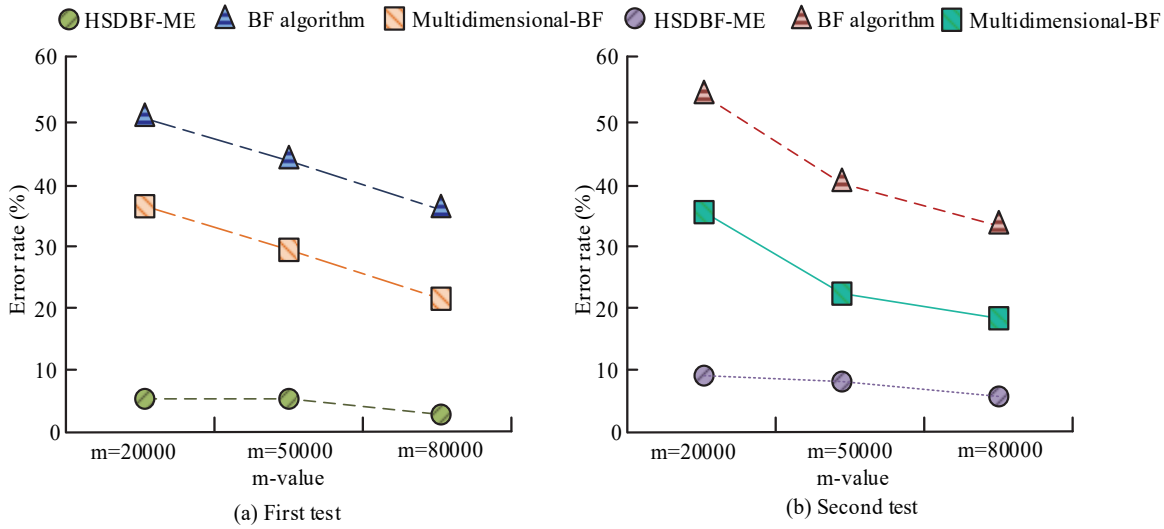


Figure 8 Miscalculation rate of each algorithm at different bit vector sizes *m*

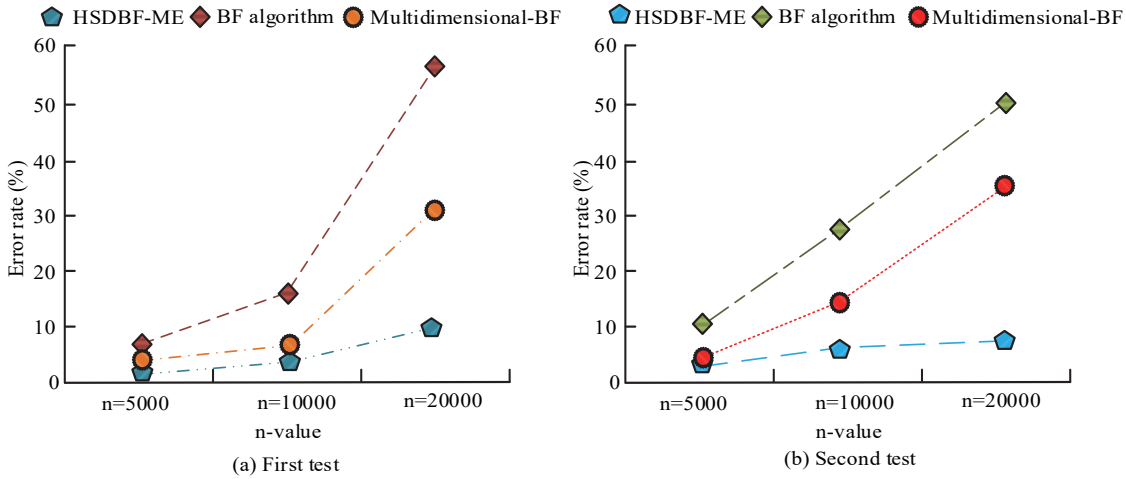


Figure 9 Miscalculation rate of each algorithm under different datasets *n*

4.2 Experimental Analysis of Python and URL De-duplication Algorithm

This study further validates the effectiveness of Python, using naive web crawlers, parsing web crawlers, and asynchronous web crawlers for experimental comparison. The experiment mainly compares the number of pages obtained under the same conditions and the time spent. The experimental results are shown in Fig. 10. The highest number of web pages obtained using Python is 1864, which increases by 712, 360, and 495 compared to the other three web crawling techniques. At the same time, this technology takes the lowest time, only 58.2 seconds, which is 31.9 seconds less than the naive web crawling technology. Overall, using Python performs well in terms of crawling efficiency and web page quantity, and has strong feasibility.

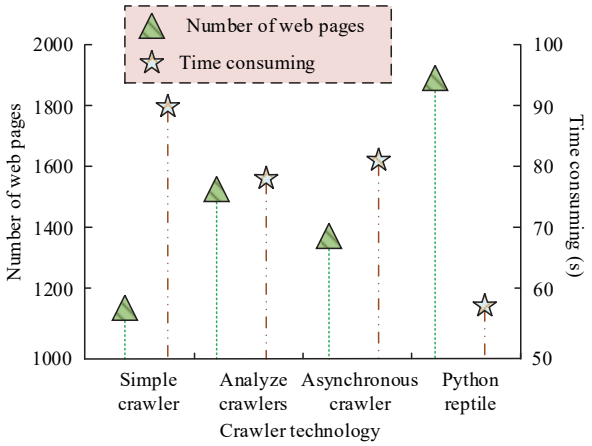


Figure 10 Number of pages obtained under equal conditions and time consumption

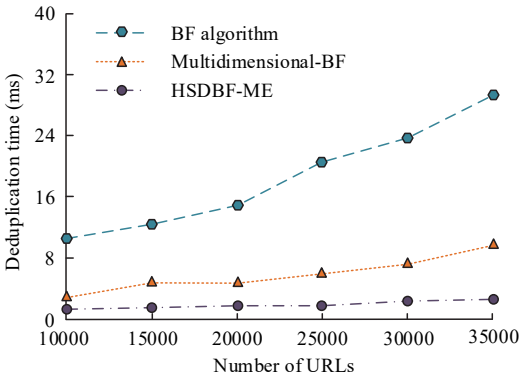
This study then compares the de-duplication time, de-duplication accuracy, and memory consumption of various algorithms when processing different numbers of URLs. The experimental results are shown in Fig. 11. In Fig. 11a, as the number of URLs increases, the de-duplication time of each algorithm also increases, and the de-duplication time of the studied algorithms is lower than that of other algorithms. When the number of URLs is 10,000, the de-duplication time of the HSDBF-ME algorithm is only 0.21 ms, which is reduced by 9.58 ms and 3.16 ms compared to the other two algorithms, respectively.

In Fig. 11b, as the number of URLs increases, the de-duplication accuracy of each algorithm decreases, but the HSDBF-ME algorithm has a de-duplication accuracy greater than 99.9%, which is significantly better than the other algorithms. When the number of URLs is 35000, the HSDBF-ME algorithm achieves a de-duplication accuracy of 99.95%, which is improved by 0.77% and 0.35% compared to the other two algorithms, respectively. In Fig. 11c, the memory consumption of each algorithm increases with the number of URLs, while the HSDBF-ME algorithm has the lowest memory consumption and the lowest growth rate. The memory consumption of this algorithm is only 0.28 MB when the number of URLs is 20,000, which is 1.18 MB lower than the BF algorithm, indicating that the HSDBF-ME algorithm has superior de-duplication performance.

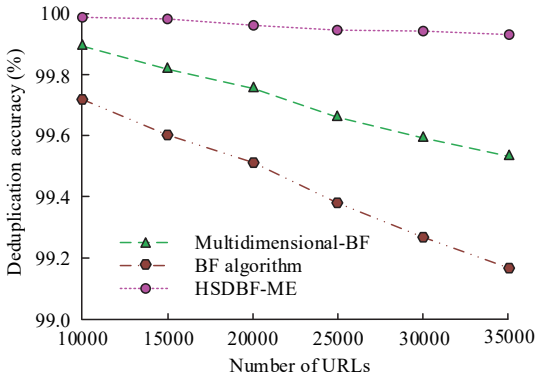
This result indicates that under different hash function values k and bit vector sizes m, the false positive rate of HSDBF-ME algorithm is significantly lower than that of basic BF algorithm and multi-dimensional BF algorithm. Especially in the case of $k = 4$ and $m = 80{,}000$, the HSDBF-ME algorithm has the lowest false positive rate. This indicates that the HSDBF-ME algorithm can more accurately perform URL de-duplication, improving the stability and reliability of the algorithm. This is consistent with the research results of Bhat R. et al., mainly because the HSDBF-ME algorithm uses a combination of multiple hash functions and multiple bit vectors, which increases the accuracy of deduplication [23]. At the same time, Python web crawling technology has the highest number of web pages obtained and the shortest time spent. This proves that Python web crawling technology has high efficiency and feasibility in practical applications, which can improve the speed and quantity of data crawling. The research of researchers such as Khan N. also confirms this point, because Python web crawler technology has the characteristics of simplicity and ease of use, and has rich third-party libraries and tool support, which can easily perform web page crawling and data extraction [24]. The use of Python crawler technology combined with HSDBF-ME algorithm for URL de-duplication has significant advantages in practical QDA applications, which can significantly reduce misjudgment rates and improve processing efficiency. Especially in large-scale data processing, this method performs well and has significant

improvements compared to traditional algorithms. Concurrently, the Python web crawler technology is capable of efficiently obtaining web page data and can be combined with the HSDBF-ME algorithm to leverage the strengths of existing technology and streamline the data acquisition and processing process. Furthermore, the enhanced efficiency of the algorithm enables more effective utilisation of computing resources, thereby reducing the system load.

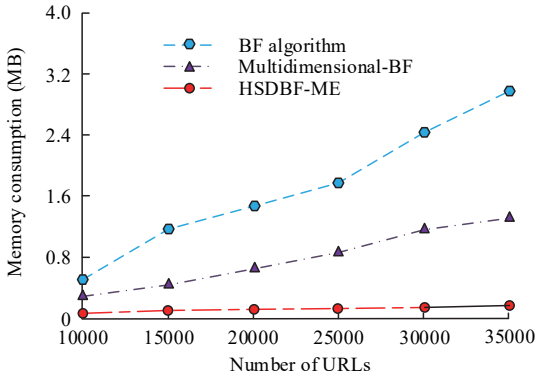
In summary, these research findings have important significance and reference value for practical applications in the fields of information security and web crawling. By improving algorithms and adopting efficient technical tools, the accuracy and efficiency of data processing can be improved, promoting the development and application of related fields.



(a) The deduplication time of different algorithms



(b) The accuracy of deduplication using different algorithms



(c) Memory consumption of different algorithms

Figure 11 The deduplication time, accuracy, and memory consumption of each algorithm

5 CONCLUSION

QDA is of great significance in various fields, as it can provide data-driven decision support and accurate predictions. Based on this, this study introduced Python for web crawling and designed the HSDBF-ME algorithm for URL de-duplication. The data showed that when k was 2, the misjudgment rate of the HSDBF-ME was only 0.51 ± 0.21%. When k was 4 and 6, the misjudgment rates of the HSDBF-ME were only 0.28 ± 0.11% and 0.32 ± 0.23%. When m was 20,000 and 80,000, the misjudgment rates of the HSDBF-ME were 8.33% and 4.32%. When m was 50,000, the misjudgment rate of HSDBF-ME was further reduced to 4.51%, which is 35.91% and 41.87% lower than the basic BF and multi-dimensional BF, respectively. When n was 5,000, the misjudgment rate of the HSDBF-ME was only 2.15%. When n was 10,000 and 20,000, the misjudgment rates of the HSDBF-ME were 3.04% and 7.28%, which were reduced by 19.86%, 9.73%, 43.05%, and 27.69% compared to the basic BF and multi-dimensional BF algorithms, respectively. In addition, Python had the highest number of web pages obtained under the same conditions and the lowest time spent, indicating that the proposed technology can effectively implement QDA software design. However, in practical applications, Python web crawling technology is limited by website anti-crawling mechanisms, resulting in incomplete data collection and the possibility of noise, missing information, or errors. Therefore, in the future, attention needs to be paid to network anomalies, changes in page structure, and other situations to ensure the reliability of data collection to the greatest extent possible. In addition, data cleaning and preprocessing should be carried out to improve feature extraction and similarity matching through machine learning, to further optimize the de-duplication method.

6 REFERENCES

[1] Johnson, C., Khadka, B., Basnet, R. B., & Doleck, T. (2020). Towards Detecting and Classifying Malicious URLs Using Deep Learning. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA)*, *11*(4), 31-48. https://doi.org/10.22667/JOWUA.2020.12.31.031

[2] Azeez, N., Salaudeen, B., Misra, S., Damaševičius, R., & Maskeliūnas, R. (2020). Identifying phishing attacks in communication networks using URL consistency features. *International Journal of Electronic Security and Digital*

Forensics, *12*(2), 200-213. https://doi.org/10.1504/ijesdf.2020.106318

[3] Liang, Y., Wang, Q., Zheng, X., Yu, Z., & Zeng, D. (2021). Robust Detection of Malicious URLs with Self-Paced Wide & Deep Learning. *IEEE Transactions on Dependable and Secure Computing*, *19*(2), 717-730. https://doi.org/10.1109/tdsc.2021.3121388

[4] Luo, S., Chen, X., Wu, Q., Zhou, Z., & Yu, S. (2020). Joint edge association and resource allocation for cost-efficient hierarchical federated edge learning. *IEEE Transactions on Wireless Communications*, *19*(10), 6535-6548. https://doi.org/10.1109/twc.2020.3003744

[5] Zamir, A., Khan, H., Iqbal, T., Yousaf, N., Aslam, F., Anjum, A., & Hamdani, M. (2020). Phishing web site detection using diverse machine learning algorithms. *The Electronic Library*, *38*(1), 65-80. https://doi.org/10.1108/el-05-2019-0118

[6] Shen, M., Liu, H., Zhu, L.,Xu, K., Yu, H., Du, X., & Guizani, M.(2020). Blockchain-assisted secure device authentication for cross-domain industrial IoT. *IEEE Journal on Selected Areas in Communications*, *38*(5), 942-954. https://doi.org/10.1109/jsac.2020.2980916

[7] Tan, L., Shi, N., Yu, K., Aloqaily, M., & Jararweh, Y. (2021). A blockchain-empowered access control framework for smart devices in green internet of things. *ACM Transactions on Internet Technology (TOIT)*, *21*(3), 1-20. https://doi.org/10.1145/3433542

[8] Lee, O., Heryanto, A., Razak, F., Raffei, M., Phon, D. N. E., Kasim, S., & Sutikno, T. (2020). A malicious URLs detection system using optimization and machine learning classifiers. *Indonesian Journal of Electrical Engineering and Computer Science*, *17*(3), 1210-1214. https://doi.org/10.11591/ijeecs.v17.i3.pp1210-1214

[9] Karve, S. M., Kakad, S., Amol, S., Gavali, A. B., Gavali, S. B., & Shirkande, S. T. (2024). An Identification and Analysis of Harmful URLs through the Application of Machine Learning Techniques. *International Journal of Intelligent Systems and Applications in Engineering*, *12*(17s), 456-468.

[10] Tsai, Y. D., Liow, C., Siang, Y. S., & Lin, S. D. (2024). Toward more generalized malicious url detection models. *Proceedings of the AAAI Conference on Artificial Intelligence*, *38*(19), 21628-21636. https://doi.org/10.1609/aaai.v38i19.30161

[11] Yang, S. & Abas, A. (2021). Data science talents mining from online recruitment market in China based on data mining technique. *Journal of Information and Communication Technology in Education*, *8*(2), 118-125. https://doi.org/10.37134/jictie.vol8.2.11.2021

[12] Hien, N. L. H., Tien, T. Q., & Hieu, N. V. (2020). Web crawler: Design and implementation for extracting article-like contents. *Cybernetics and Physics*, *9*(3), 144-151. https://doi.org/10.35470/2226-4116-2020-9-3-144-151

[13] Khan, N. & Haroon, M. (2023). A Personalized Tour Recommender in Python using Decision Tree. *International Journal of Engineering and Management Research*, *13*(3), 168-174. https://doi.org/10.2139/ssrn.4493066

[14] Tao, G., Shi-qi, W., Yi-Cheng, S., & Zhi-hang, T. (2020). Research on Online Review Based on LDA Subject Model. *International Journal of Advanced Networking and Applications*, *12*(3), 4606-4612. https://doi.org/10.35444/ijana.2020.12306

[15] Osanyin, Q. A. &Ajose-Ismail, B. M. (2020). A neural network language document representation technique for web-page classification. *International Journal of Computer Applications*, *176*(14), 38-46. https://doi.org/10.5120/ijca2020920071

[16] Asenahabi, B. M. (2019). Basics of research design: A guide to selecting appropriate research design. *International Journal of Contemporary Applied Researches*, *6*(5), 76-89.

[17] Soher, B. J., Semanchuk, P., Todd, D., Ji, X., Deelchand, D., Joers, J., & Young, K. (2023). VeSPA: integrated applications for RF pulse design, spectral simulation and MRS data analysis. *Magnetic resonance in medicine*, *90*(3), 823-838. https://doi.org/10.1002/mrm.29686

[18] Mokayed, H., Quan, T. Z., Alkhaled, L., & Sivakumar, V. (2023). Real-time human detection and counting system using deep learning computer vision techniques. *Artificial Intelligence and Applications*, *1*(4), 221-229. https://doi.org/10.47852/bonviewaia2202391

[19] Shin, S. (2021). A study on the framework design of artificial intelligence thinking for artificial intelligence education. *International Journal of Information and Education Technology*, *11*(9), 392-397. https://doi.org/10.18178/ijiet.2021.11.9.1540

[20] Wang, D., Zhang, Q., & Hong, S. (2021). Research on Crawling Network Information Data with Scrapy Framework. *International Journal of Network Security*, *2021, 23*(2), 326-331.

[21] Rachmad, Y. E, Mahendika, D., Lestari, N. C, Meisarah, F., & Rachman, R. S. (2023). The Relationship between Teachers Perception and Belief on Readiness to Plan a Learning of Early Childhood Education Program Teachers. *Edumaspul: Jurnal Pendidikan*, *7*(1), 154-157.

[22] Weirman, S. & Alexander, A. (2020). Hyperlinked sympathizers: URLs and the Islamic state. *Studies in Conflict & Terrorism*, *43*(3), 239-257. https://doi.org/10.1080/1057610x.2018.1457204

[23] Bhat, R., Thilak, R. K., & Vaibhav, R. P. (2022). Hunting the pertinency of hash and bloom filter combinations on GPU for fast pattern matching. *International Journal of Information Technology*, *14*(5), 2667-2679. https://doi.org/10.1007/s41870-022-00964-3

[24] Khan, N. & Haroon, M. (2023). A Personalized Tour Recommender in Python using Decision Tree. *International Journal of Engineering and Management Research*, *13*(3), 168-174. https://doi.org/10.2139/ssrn.4493066

Contact information:

XiaoGang LUO, Associate Professor
(Corresponding author)
Geely University of China,
Building 17, No. 175, Nanhu West Road, Huayang Sub-district,
Tianfu New Area, Chengdu City, Sichuan Province, China, 641423
E-mail: 2644310622@qq.com

Liang ZHOU, Engineer
Geely University of China,
Building 17, No. 175, Nanhu West Road, Huayang Sub-district,
Tianfu New Area, Chengdu City, Sichuan Province, China, 641423
E-mail: feixiangdedaizi@qq.com