Crawling Strategy Based on Domain Ontology of Emergency Plans

Junjie Wang[1, a], Depeng Dang[2, b], Pengxia Zhou[3], Hongjie Wang[4], Xue Jiang[5] and Shihang Huang[6]

[1 2 3 4 5 6] College of Information Science and Technology, Beijing Normal University, Beijing, 100875, China

[a]email: wangjunjie@mail.bnu.edu.cn, [b]email:ddepeng@bnu.edu.cn

Keywords: Ontology; Emergency plan; Theme crawler

Abstract. In order to build an effective emergency plans crawler, this paper affords a new algorithm for emergency plans identification and a new idea of URL predict using URL pattern library. Through the experiment, we found the crawler achieves efficient collection of emergency plans from the web. The emergency plan crawler is proper for emergency plan collection.

Background and significance

Emergency plans guide precaution of potential disasters. There is no emergency plan base which makes the emergency plans be easily shared. In the research we crawl emergency plans and then save them to the server. It is the foundation to share and search emergency plans.

There are different kinds of web pages analysis methods. In TF/IDF it is thought that words appear in less documents, and appear in one document more times represent the document better[1]. Naive Bayesian classifier method assumes each feature of the samples cannot be associated with other features [2]. C4.5 decision tree needs be given a bunch of samples[3]. Ehrig uses the ontology into the subject crawling in his work [4]. Ye Yuxin [5] introduces ontology reasoning into theme crawler. BestFirst [6] believes that if a web page A is related to a theme, then the page pointed by the web page A should also be related to this theme. PageRank algorithm is based on network topology [7]. HITS assesses the quality of the web by two weights—the Authority and the Hub [8]. In this paper we learn from the HITS and construct our own page crawling method.

Theme crawler of emergency plans domain

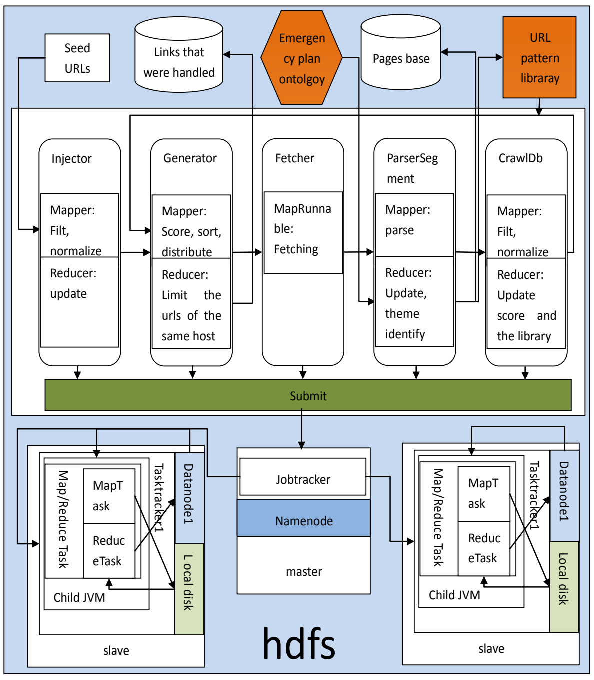


Fig.1. Theme crawling framework of emergency plans

This paper establishes domain ontology through the research on emergency plans, and conducts the theme determination. Also finds the special nature of the link structure of the theme, and designs a new method based on the URL pattern library for the link prediction. This study extends the common crawler of Nutch, and uses the plug-in mechanism of Nutch.

Definition 1 **Target page** contains emergency plans and need to be crawled into storage.

The crawling framework used in this paper, as shown in Fig.1 has two special modules. The emergency plans ontology records the knowledge in emergency plans. This ontology supports calculating how much a page relevance to the theme. The URL pattern library is a library of URL patterns. URLs satisfied with these patterns have more probabilities to point to the target pages. The URL in accordance with the pattern in the queue will be crawled preferentially.

Emergency plans topics identification

We doing topic identification using domain ontology of emergency plan. Ontology is a new field of research in information technology areas, and is an important mean to describe the semantic model. We can use ontology to store knowledge of emergency plan in this paper.

Definition 2: **Emergency plan crawling ontology** can be described as a four-tuple: Oemergency = {C, K, E, M}.

C represents a collection of concepts, K means a collection of key words, E denotes the inclusion relation among the concepts, and M represents a mapping function from keywords to concepts.

We explored emergency plans, and we collected 8 concepts in the plans. The concepts are put into a tree structure according to the inclusion relation. In fact, there are many other concepts. In the identifying of emergency plan, only the 8 concepts in the top layer are useable.

Algorithm 1. The algorithm for emergency plans identification

```
input:
    words' frequency statistical tables WT(<word, count>pair)
    emergency plans ontology $O_{emergency}$
output: topic identification results True.False
Step1. initialize the concept set CE empty
Step2. If there is no unvisited element in the WT, go to Step 7.
Step3. Took out a pair of < $word_i$ $count_i$ >, and look for $word_i$ from the keywords table K of the $O_{emergency}$. If not find, go to Step 2.
Step4. Based on the relationship in the mapping relationship table M of keyword to the concept, get the corresponding concept $c_j$ of $word_i$. If the
    $c_j$ is in CE, go to Step 6.
Step5. set the frequency of $c_j$ $count_j = count_i$ , and put < $c_j$, $count_j$ > into the CE, then skip to Step 2.
Step6. Update $count_j = count_j + count_i$, and skip to Step 2.
Step7. According to the concept set C and concept inclusion relation E in the $O_{emergency}$, form tree structure $T_{emergency}$.
        Step7.1 Set the < emergency plans, $count_{emergency\ plans}$> as the root node of the $T_{emergency}$, and initialize the $count_{emergency\ plans}$= 0. And put the
            "emergency plan" into the queue.
        Step7.2 If the queue is empty, go to Step 8. Otherwise, put out an element < $c_i$, $count_{ci}$ > from the queue.
        Step7.3 Find such relationships $c_j \subset c_i$ in the E. If do not exist, skip to Step 7.2.
        Step7.4 Set $count_{cj} = 0$, make< $c_j$, $count_{cj}$ > as a child of < $c_i$, $count_{ci}$ >, and put $c_j$ into the queue, then skip to Step 7.2
Step8. Recursively calculate the count value of each node in the $T_{emergency}$. Set the root node as the current node.
        Step8.1 If the current node c has children c1, c2,...Cn, so $count_c=count_{c1}+count_{c2}+...+count_{cn}$, By the recursive process, we know that
            $count_{c1}$,$count_{c2}$,...,$count_{cn}$ will turn out to be the current node. Continue this step.
        Step8.2 Find whether c in CE. If not, set $count_c = 0$. If in, read to get the corresponding Pair< c, countc >of the concept c in the CE. Then
            update the concept frequency as $count_c$ for c in the $T_{emergency}$.
Step9. Check the nodes of "emergency preparation", "monitoring and early warning" and "emergency response" and " further disposal",
    whether the frequency of the four concept is more than the threshold value tp. If one item does not reach, it returns False.
Step10. Check the root node "emergency plan". If the frequency $count_{emergency\ plan} > \Theta$, return True. Otherwise , return False.
```

Using the word frequency statistics method, we did the segmentation, words frequency statistic, and selected keywords from the emergency. We determined the mapping function from keywords to concepts artificially. In the constructed emergency plan crawling ontology there are 8 concepts, 7 relations among concepts, 3894 keywords and 4077 maps from keyword to concepts.

After establishing the emergency plans domain ontology, we can get the score of each module of emergency plans. Emergency plans generally have four parts (emergency preparedness, monitoring and warning, emergency response and further disposal), two attributes (geographical information and domain information) and one group (related department). When using emergency plans ontology doing identify, we need to check each part and each attribute if it obtain a certain score.

The link prediction based on URL pattern library

The link relations between web pages have an important role for predicting web topics.

Definition 3: **Noise link** are links that not points to a target page.

Definition 4: **URL pattern** is a string which is used to describe a series of URL with a certain syntactic rule.

After have analyzed the emergency plans link structure in the network, we find that the URLs of different target pages from the same website have the same pattern. After realizing it, we do the URL prediction according to the links' structure, or pattern. We can extract the URL pattern of the target pages from the website. URL pattern should only be extracted from the same website.

Trie tree is used to store the URL pattern library in this paper. We add URL pattern or merge URL patterns by corresponding operate of the Trie tree. Trie tree's advantage is that minimizing unnecessary string comparisons. The defect is space consumption.

```
Algorithm 2: the algorithm for URL pattern
Initial state: the URL pattern library $T_{urlPattern}$ is empty.
Step1. Whether it runs up to the grab depth or the number of reserved pages, if reaches, just quit.
Step2. If the crawling queue is empty, go to Step5. Else select a URL url1 from the crawling queue , crawl the page and analysis, then stores
    the link information to crawldb. Analysis the text content, if it is not an emergency plan, then continue to repeat Step2 to grab the next
    page. Otherwise go to the next step.
Step3. Find url1,s hostname host, then in the URL pattern library find whether there is URL pattern deriving from the host. If not, put url1
    into the URL pattern library, and go to Step2; Otherwise go to the Step4.The following is the method of joining Url1 into the mode
    library URL:
    Step3.1 Set the current character currentChar1 as the first character of url1, and set the current node currentNode as the root node
        of $T_{urlPattern}$ tree.
    Step3.2 If currentNode had no child, go to Step3.3; Otherwise, scanning every child childNode of currentNode. If a childNode's
        characters = currentChar1, go to Step3.4. If there is no any character in a childNode equals currentChar1, go to Step3.3.
    Step3.3 New-built a node newNode , and the characters in the newNode are currentChar1, and add the newNode as a child of
        currentNode , then update currentNode = newNode. If url1 has a next character , it will be store the next character in the
        currentChar1, and repeat Step3.3. Otherwise the method of adding url1 to URL pattern library is over.
    Step3.4 Set currentNode = childNode, and currentChar1 is the next character of url1, and jump to Step3.2.
Step4. Merge Url1 with the corresponding to address mode urlPattern in the URL pattern library, and make the new URL pattern can
    match the url1 and urlPattern. Specific merging step is:
    Step4.1 Currently comparative character currentChar1, currentChar2 respectively is set to the first characters of url1 and the first
        character of urlPattern.
    Step4.2 If currentChar1 = currentChar2, go to Step4.4. Otherwise, go to the next step.
    Step4.3 At this time currentChar1! = currentChar2, if currentChar1 currentChar2 $\in D$, then update the current urlPattern current
        character as '/b '. If currentChar1 currentChar2 $\in A$, then update the current urlPattern current character as'/ r'.
    Step4.4 Check whether there is characters in the back of url1 and urlPattern . If there is no character in the back of urlPattern, skip
        to Step2. If there is character in the back of urlPattern, and no character in the back of url1, delete all children of urlPattern,s
        current node , then skip to the Step1. If all have, set the current character of currentChar1 as the next character of url1, and set
        the current character of currentChar2 as the next character of url2, then skip to Step4.2.
Step5. According to the URL pattern library, one by one check the links url1 in the crawldb. For a url1, if we can find a route from the root
    node to a leaf node in the $T_{urlPattern}$ tree , and the string in the route is url1 prefix, then the url1 matches a URL pattern. Particular way
    is:
    Step5.1 If in the crawldb there are no matching url, take out a article, and write for url1, then go to Step5.2; Otherwise, go to the
        Step6.
    Step5.2 Set the current character currentChar1 as the first character of url1, and set the current node currentNode as the root node
        of $T_{urlPattern}$ tree.
    Step5.3 If currentNode has no child, then the match is successful, and improve url1's score, then go to Step5.1. Otherwise, scan every
        child of currentNode childNode , if one childNode's character = currentChar1, the current matching is successful, then go to
        Step5.4.
    Step5.4 Set currentNode = childNode. If there is no next character in the url1, go to Step5.5 t; Otherwise go to Step5.6.
    Step5.5 If currentNode's child is empty, the match is successful, and improve url1's score, then go to Step5.1. If currentNode's child is
        not null, the match fails, and skip to Step5.1.
    Step5.6 Update currentChar1 as the next character of url1, and skip to Step5.2.
Step6. Putting the crawldb after updating into the queue, and skip to Step1.
```

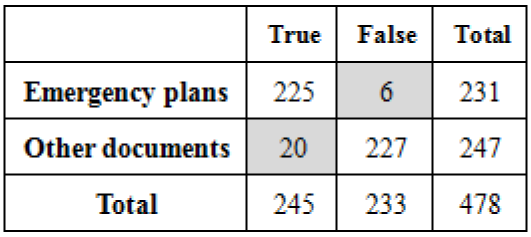
When we use Nutch to crawl emergency plans, we use distributed cluster. URLs are distributed to each task machine, and pages are crawled by the corresponding machine. In the process of distribution, the URLs from the same host are guaranteed to be distributed to the same machine. So, on each machine we can establish a Trie tree to represent a part of URL pattern library processed by the corresponding machine. The Trie tree residing on each machine in the correspond to one part of the entire URL pattern library. It won't affect the decision result of URL pattern , and also can effectively reduce the size of the Trie tree on each machine.

Definition 5: **The digital character set D**= {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '/b'}. For any numeric characters digitaChar, digitaChar = '/b' established.

Definition 6: **The alphabetic character set A** = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', '/r'}. For any alphabetic characters alphaChar, alphaChar= '/r' established.

Results and conclusion

In the experiment, we made the secondary development on Nutch 1.2.



	True	False	Total
Emergency plans	225	6	231
Other documents	20	227	247
Total	245	233	478

Fig.2. The result of theme identify

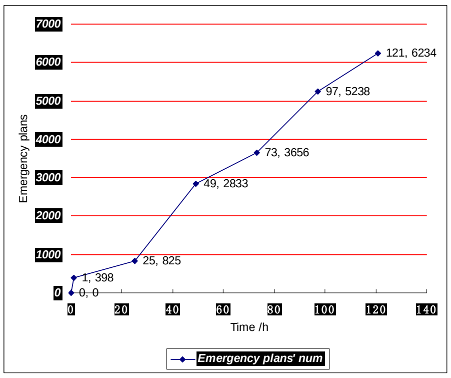


Fig.3. Emergency plans gathering

From Fig.2 we learn that, the positive correct rate is 225/231 = 0.97. But the negative correct rate is 227/247 = 0.91. It is because, there are documents of legal provisions or reports that related to emergency on the governments' websites. These documents are similar to emergency plans after segmentation. From Fig.3 we can find that, we got 398 emergency plans in the first hour. It is a sharp speed, it is mainly due to our choice of the seed URLs.

Through the experiment, we find that, the emergency plans identification algorithm performs well of the emergency identification. Using the URL pattern library made use be able to gather about 50 emergency plans per hour. The emergency plan crawler is proper for emergency plan collection.

Acknowledgement

In this paper, the research was sponsored by the Nature Science Foundation of Henan Province (Project No. 201112400450401) and Youth Fund Project of Luoyang Institute of Science and Technology (Project No. 2010QZ16).

References

[1] Salton, G. & Buckley, C. (1988). Term-weighing approaches in automatic text retrieval. In Information Processing & Management, 24(5): 513-523.
[2] E. Frank, R.R. Bouchaert. Naive Bayes for text classification with unbalanced classes. Proceedings of the 10th European conference on principles and practice of knowledge discovery in databases, Springer, Berlin (2006), pp. 503–510
[3] Peter D. Turney, Learning Algorithms for Keyphrase Extraction, Information Retrieval, v.2 n.4, p.303-336, May 2000
[4] Ehrig M, Maedche A. Ontology-Focused crawling of Web documents. In: Lamont BG, ed. Proc. of the 2003 ACM Symp. on Applied Computing. New York: ACM Press, 2003.
[5] YE Yu-Xin, OUYANG Dan-Tong. Semantic-Based Focused Crawling Approach. Journal of Software, 2011, 22(9)
[6] Pearl, J. (1984). Heuristics: Intelligent Search Strategies for Computer Problem Solving. Reading, MA: Addison-Wesley.
[7] L. Page, S. Brin, R. Motwani, T. Winograd. The PageRank citation ranking: bringing order to the Web Manuscript in Progress. 2010.
[8] J. Kleinberg. Authoritative sources in a hyperlinked environment. Proc. ACM-SIAM Symposium on Discrete Algorithms. 1998.