International Journal of Database Theory and Application
Vol.8, No.2 (2015), pp.33-42
http://dx.doi.org/10.14257/ijdta.2015.8.2.04

Efficient Information Extraction Based on Signature Index

CanghongJin[1], Minghui Wu[1], Zemin Liu[2] and Shiwen Cheng[3]

[1]Department of Computer Science and Engineering, Zhejiang University City College, Hangzhou, China
[1]College of Computer Science and Technology, Zhejiang University, Hangzhou, China
[2]University of California, Riverside, CA, USA
chjin@zju.edu.cn, mhwu@zucc.edu.cn

Abstract

Information Extraction (IE) is an essential tool to retrieve structured information from at text (including web-content).Given a pattern query, an ideal IE application should be able to extract the matched target effectively and efficiently. However, as far as we know, efficiency and flexibility are major concern for typical IE tools since they either use brute-force document parsing for each query off-line or support on-line query in pre-extracted elements. In order to promote accuracy and efficiency of extraction, in this paper we propose a novel framework iExtractor that leverages In-formation Retrieval (IR) indexes to speed-up IE processes. We index text blocks with their signatures (presented as bit-strings) and propose efficient IE algorithms based on the signature index. Hence, iExtractor can validate query pattern in signature index without original text. The framework also supports on-line extraction through a general and flexible pattern extraction language. Our extensive experimental results on diverse real datasets show that our approach delivers stable efficiency and has outperforms baselines in terms of extraction accuracy.

Keywords: *iExtractor, signature index, information extraction*

1. Introduction

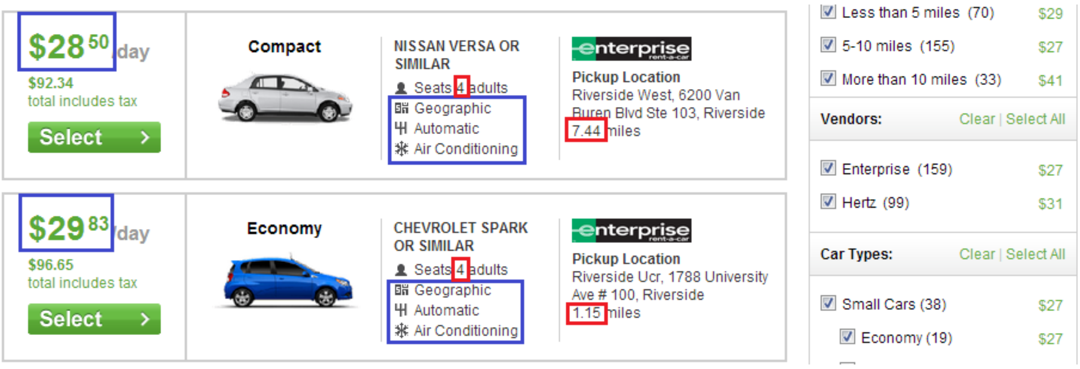
Information Extraction (IE) is the task of extracting assertions from massive corpora without requiring a pre-specified vocabulary. It has been a critical topic and attracted lots of attention from the areas of text understanding, data mining and machine learning. With ever growing and often changing document corpus such as world wide web (WWW), it is very challenging to execute IE queries over such large data sets efficiently. Furthermore, there is an increasing need to perform ad-hoc IE queries, in which the query patterns are not known before query time [13]. However, in some most current approaches, documents are processed for a set of pre-defined pattern queries to support fast online answer for these queries [1]. There is an obvious deficit of the approaches: for the queries in the pre-defined set they need to do a brute-force match involving a parse and check (using techniques such as wrapper-generation) over all the documents in the corpus. This limits their ability to support ad-hoc queries. Consider the following example, which demonstrates some of the current approaches to IE. For simplicity, we refer IE query as query.

Example 1: *Toward a collection of news documents where each document is labeled with a pre-defined news topic, the IE task at hand is to find titles for a given topic.* To extract the title related to a certain topic in Example1, The approach in article [1] required three critical components: relevant context, extraction pattern and a brute-force parse process. Relevant context is a part of whole document which related with a query. In this

ISSN: 2005-4270 IJDTA

paper, we consider proximity relationships in which the content is considered relevant to the query if it is near the input query pattern. Extraction pattern (also named text wrapper) is considered to be an information source that "wraps" the target content and is generally referred to as the extractor program which fetch data by parsing documents. If the user wants to find email subject with group termed NSWC, the first step is getting context surrounding NSWC. Then treat words before and after target data item as its extraction pattern. Finally, the relevant context is parsed to determine if there is a match. As a result, IE process is time consuming and in-efficient for on-line extraction task.

Example 2: *Car rentals is a car rent provider on web where the page includes car related information as rental fares, pick up location etc. The page structure is showed in Figure 1. Consider a user who is searching for "rent price or distance miles" based on the crawled documents from Car rentals website.*



$28 50 /day
$92.34
total includes tax
Select ›

Compact

NISSAN VERSA OR SIMILAR
Seats 4 adults
Geographic
Automatic
Air Conditioning

enterprise
Pickup Location
Riverside West, 6200 Van Buren Blvd Ste 103, Riverside
7.44 miles

$29 83 /day
$96.65
total includes tax
Select ›

Economy

CHEVROLET SPARK OR SIMILAR
Seats 4 adults
Geographic
Automatic
Air Conditioning

enterprise
Pickup Location
Riverside Ucr, 1788 University Ave # 100, Riverside
1.15 miles

☑ Less than 5 miles (70) $29
☑ 5-10 miles (155) $27
☑ More than 10 miles (33) $41

Vendors: Clear | Select All
☑ Enterprise (159) $27
☑ Hertz (99) $31

Car Types: Clear | Select All
☑ Small Cars (38) $27
☑ Economy (19) $27

Figure 1. Entity Information in Car Rental Web Page

In Example 2, some items like price are wrapped by HTML tags (blue boxes in Figure 1) whereas other items such as distance are wrapped by natural language content (red box).For those in blue boxes, since the target items are wrapped in in an HTML template, HTML parser could be used to extract them by CSS style or layout id or HTML tags. Wrappers could be classified as manually defined, auto-generated or based on machine learning [7] or classified as record-level, page-level or site-level wrapper [12]. All above mentioned wrappers, however, are defined offline phase and need work by doing an exhaustive scan of the corpus. Moreover, for those in red box, items are included non-HTML format and wrapper is designed as terms around target item, *e.g.* text wrapper of value 7:44's is Riverside and miles. Although entity recognition or NLP-based tools like WHISK [reference needed] can extract item from free text, it is difficult to determine all items the users might want to find in the web-corpus.

As we see from the above discussion, typical IE tools suffer from the following disadvantages: (1) Full text extracting requires high response time. *e.g.* an extraction on DBLife which draws 5-10 word length item takes several hours[4],and is therefore unsuitable for on-line extraction; (2) user can only find information on pre-structured data. *e.g.* in scenario 2, the data should be extracted and stored in structured formats; (3) with the additional or updating of rules, IE need to re-parse entire document collection, *e.g.* in example 2, if user defines a new wrapper to get the number of seats available for adults (the wrapper is Seats and adults).

In this paper, we focus on addressing the aforementioned drawbacks. Information Retrieval (IR) methods have ability to manage a huge number of corpuses. We propose a framework wherein IE is augmented with IR approaches to significantly improve extraction performance. Traditional IR indexes such as inverted indexes reorganize information by term-doc view which is not easy to fetch background for further extract. Suffix tree, a kind of Tire, labels string with tree edges and provides a linear-time solution for substring match problem. Therefore, it is one of solutions for our goal. Unfortunately, the cost of speedup is the storage of index with suffix tree is larger than the space to store content itself. In comparison with this index structure, signature file has advantages of supporting set-oriented queries efficiently and involving high processing and I/O cost [2]. In this article, we advocate a signature file based Information Extractor (iExtractor) to support on-line extraction with extraction pattern. Based on user given interesting items and sample documents, iExtractor generate a special wrapper and return all target items have the same pattern on-line efficiently. In addition, iExtractor support both tag-based or non-tag wrappers and regardless the format of original data. Our main contributions are as follows:

- We formalize the task of iExtractor online extraction by several novel definition models.
- We propose signature index for iExtractor and give efficient algorithms to match pattern in bit-strings file.
- We design an extraction language and express the overview of extract process.
- Experimental results of different format and size datasets validate the effectiveness of our framework.

2. Definitions

Pattern Query: A pattern query $PQ = (t_1, t_2, \dots, t_n)$ is an ordered list of query terms t$_i$. We define $Distance(t_i, t_j, D) = |Pos(ti, D) - Pos(tj, D)|$, where *D* denotes a document and $Pos(t, D)$ is the position of term t in *D*.

We determine if the document strictly contains the pattern query phrase using Strict Match.

Definition 1 (Strict Match): We say document D strictly match query PQ if (i) D contains all query terms in PQ.

$$SMM(Q,D) = \begin{cases} Pos(t_1,D) = k \\ Distance(t_{i+1},t_i) = Distance(t_{k+1},t_k) \end{cases}$$

Where $0 \le k \le Max(Pos(t,D))$

If document Sample is "*California is a state located on the west coast of the United States. Sacramento is the state's capital.*", pattern $SMM(United\ States, Sample)$ can be matched while $SMM(StatesUnited, Sample)$ fails.

Definition 2 (Block Model): $BM(t, D, R_l, R_r)$can be defined as follows. Given term t in document D and a distance constrain, denoted as (R_l, R_r), that relevant content must be in range of R_l terms preceding $t$and $R_r$succeeding$t$.

Definition 3 (Equal Length Block Array): given a document D and an integer N, Equal length Block Array$EBA(D, N)$ returns an array of blocks with the same number of terms.

3. I Extractor Components and Algorithms

Our framework iExtractor generates and links bit masks as fingerprints for text blocks and designs algorithms to use for pattern validation in index.

3.1 Signature File with Indexes

Off-line phase consists of two steps. First, for given text content, $EBA(D,N)$ returns an array of blocks as $< B >$. Second, map block B to bit masks BFP by signature function.

Algorithm 1*BuildBlockFingerprint(B,hf)*

```
Input: text block B,signature function hf.
Output: BFP.
1:  $N \leftarrow \# of B$ ; $BFP \leftarrow 0$ ;
2:  FOR $i = 1 \rightarrow N$ DO
3:  $t \leftarrow B_i$ ;
4:  $hv \leftarrow hf(t)$ ;
5:        $BFP \leftarrow BFP \vee PosTrans( hv,pos_t)$ ;
6:  RETURN BFP;
```

As described in Algorithm 1, to process a new block *B* is split into array of terms. Then the hash value *hv* for each term t is calculated with appropriate hash function *hf*. Further, function *PosTrans* is designed to merge additional position information (line 5) with two parameters: hash value of t and its integer position value post. There are could be several approaches to define *PosTrans*. In this paper, for simplicity, we define *PosTrans* as shifting post bit offset to original hash bit arrays. Finally, we do bit OR-operation to add t feature to block signature value (line 5).

To achieve an efficient and tractable hash based solution, storage cost and hash collision should be taken into account. Akin to the Bloom filter [8], block finger prints a space efficient probabilistic data structure designed to quickly look-up membership in a set. Query returns result that elements are either 'inside set (may be incorrect)" or 'definitely not in the set". However, Hash-based approach suffers from false positive. Let *m* is the size of bit for given hash function, n is the total number of elements in set and k is the number of hash function. The probability of a false positive f is estimated by formula 2 [9].

$$f = (1 - e^{-kn/m})^k = (1/2)^k \approx (0.6185)^{m/n} \qquad (2)$$

It is easy to see that as the number of elements in a set increases, so does the probability of false positives. The probability of false positive of block fingerprint, in practice, is somewhat different with value f. It is because, instead of using hash function directly, we redesign hash function to decrease collision. We use three 16-byte length hash functions that are perfectly random to do bit AND-operation for reducing the number of 1 in bit arrays. Consequently, Table3 shows the error-ratio with different size of set, which is better than f.

The second part of off-line stage is to build connection between query terms and block fingerprint generated by Algorithm 1. We propose a novel index structure called *Block Inverted Index (BII)*. In contrast to the typical inverted index, *BII* stores the pointer to the signature of block which query term belongs to rather than the location of term directly. For example, there is a block b consist of term arrays$< t_1, t_2, \dots, t_n >$. Let BFP be the signature of b. For given any term ti, 0 <i< n, BII returns the same BFPb. That is to say terms in the same block have the same signature value by BII. Since, we use EBA (defined in section 2) to split original text and use fixed-length BFP to present block feature, it is convenient to random access to another block's feature without additional pointer.

3.2 Pattern Matching with Signature File

In this section, by using signature file developed in Section 3.1, we define pattern match process as two consisting phases. First, we need to grab part of bit string, which is considered as feature of context for extracting, from signature file by Algorithm *Get Related Fingerprint*. Second, algorithm *Single Block Matcher* and *Multi Block Matcher* are employed to judge availability of query pattern in bit strings.

```
Algorithm 2GetRelatedFingerprint($t,R_l,R_r$)
Input: term $t$, left range $R_l$ , right range$R_r$.
Output: the array of signature $< BFP >$
1:  p ← findPos(t,BII)
2:  compare $R_l$ and $R_r$ with $N$ to check if the range cross multiple blocks;
3:  $< BFP > \leftarrow selectBFP;$
4:  return$< BFP >$
```

Algorithm 2 shows the procedure of extracting query related context signature content. Given a query term t and BII, we first find the start position of bit strings in signature file (line 1). Then we calculate how many blocks need to be extracted according to the left and right range. The window width depends on both range width and length of each block (line 2). If window width crosses several blocks, the result is an array of BFP. In common situation, query term belongs to more than one blocks, therefore the final results are list of bit stings, in which each bit string is created by Algorithm 2.

So far we have a fingerprint of blocks and inverted index structure to retrieve signatures. Next, we propose a solution to verify if a phrase exists in block by its signature. Assuming query phrase is in one block for simply, we give the process in Algorithm 3. First, convert a phrase to an array of terms $< t >$. Then we calculate the bit strings for phrase by Algorithm 1 on step 1. However, position value of term in query phrase is usually different from that in block.

```
Algorithm 3SingleBlockMatcher(qp,BFP)
Input: query pattern qp, block fingerprint BFP.
Output: if BFP match qp return true
1:  < t >← qp;
2:  step 1
3:  calculate off set pos;
4:  $QH = 0$
5:  FOR$i = 0 \rightarrow N$ DO
6:          $pos_t \leftarrow i + off\ set\ pos$
7:  $hv \leftarrow hf( t_i )$
8:  $hv' \leftarrow PosTrans( hv , pos_t );$
9:          $QH \leftarrow BFP \vee hv'$
10: Step 2
11: IF $BFP = ( BFP \vee QH )$ THEN
12: return true;
13: ELSE
14: return false;
```

For example, term at the beginning of query phrase might be positioned at the middle of block. Therefore, we need to get the position of first term in query as offset for block matching process (line 3). In order to gain off set pos, in this article, we try to estimate every possible position (from0 to *N-1*) by *PosTrans* method and record every passed value. In Step 2, we bit-ORed*QH* and *BFP* and see if the value is changed.

4. Extraction Process

In this section, we describe the overview of iExtractor which includes Extraction Language (EL) and operation process.

4.1. Extraction Language

This subsection makes a language to describe extraction pattern in iExtractor. As Figure 2 shows, if user wants to gain all the email senders related with topic “Princeton”. The procedure gets text around “Princeton" and extracts content between words “from" and “subject". As a result, email"strom@watson.ibm.com" is what we want.

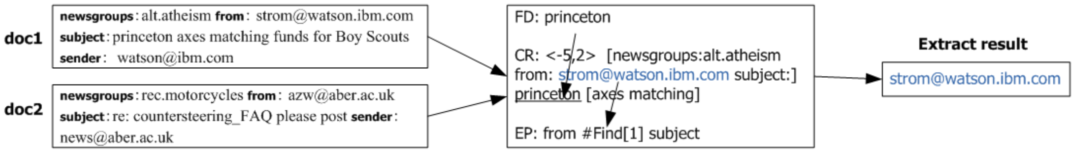

doc1
newsgroups: alt.atheism from: strom@watson.ibm.com
subject: princeton axes matching funds for Boy Scouts
sender: watson@ibm.com
doc2
newsgroups: rec.motorcycles from: azw@aber.ac.uk
subject: re: countersteering_FAQ please post sender: news@aber.ac.uk
FD: princeton
CR: <-5,2> [newsgroups:alt.atheism
from: strom@watson.ibm.com subject:]
princeton [axes matching]
EP: from #Find[1] subject
Extract result
strom@watson.ibm.com

Figure 2. Scenario for Extraction Language

For more details, we define three concepts in Extraction Language and give detail semantic as follows:

Focus Domain (FD) is terms which users are interested in. *e.g.* Princeton is FD means users want to find information entities related with Princeton.

Context Range (CR) defines the scope of text area as extract background. *e.g.* $< -5,2 >$ gives the window width with 5 words on left and 2 words on right of FD. Obviously, if CR is large enough, extract background is whole document.

Extract Pattern (EP) defines target element's pattern and its length. *e.g.* from find [1] subject means extract one word between term from and subject exactly.

Table 1 shows the notations of EL and give relevant their semantic.

Table 1. Extraction Language Notations and their Semantic

Notation	Semantic Description
@	mark term as focus domain
<Rl,Rr>	Return the range of extracted window
#Find[min,max]	Define the length of extracted item

Based on above concepts, we write a EL as from *#Find [1]subject Princeton <-5,2>* to extract senders.

4.2. Extraction Process

Now, we turn to express the overview of iExtractor and tell how these components above cooperate with each other in Figure 3. There are four essential components in iExtractor as following.

- ●GUI is a user interface to let user input Extraction Language.
- ●Model parser checks the grammar of EL and transfer EL to FD, CR and EP.
- ●Block Inverted Index Structure receives FD and CR and returns related array of bit strings by Algorithm2.

●Model Matcher uses Algorithm 3 to give the match probability of target element in blocks.

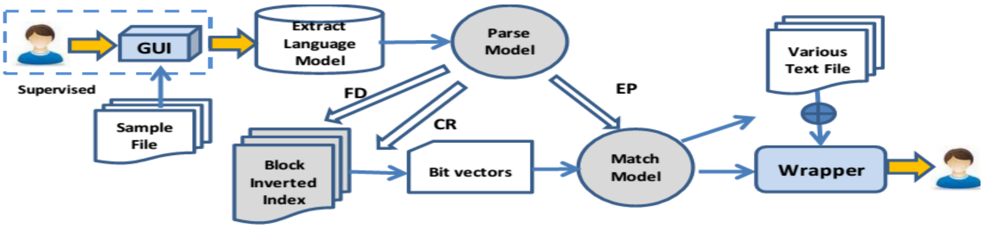


Figure 3. iExtractor Operators and Process

Obviously, we cannot obtain target like user name or phone number from signature index directly. However, iExtractor can supply a space-efficient structure to give the probability and help avoid string comparison in whole documents. Hence, iExtractor improves the efficiency of performanceand reduces the cost of I/O access.

5. Experiments

We conduct a set of comprehensive experiments to demonstrate the quality and efficiency of iExtrator. First, we analyze the storage overhead of datasets and indexes (Section5.1). Then, we evaluate compression ratio with false positive error (Section 5.2). Finally, we verify the efficiency of iExtractor in datasets with various structure type and size of data sets (Section 5.3). Table 2 introduces three datasets with their indexes.

There are three datasets as following:

1） 20 Newsgroups (NG) is a collection of documents which across 20 different categories and formatted as emails. There are two versions of dataset: full version has nearly 20k documents while mini version is the subset of full version and has 2000 documents.

2） University Web Site (UWS) is the dataset crawled from 2000 university web sites. UWS contains total216, 495 HTML formatted files. We split whole dataset into full (UWS3), medium (UWS2), and small (UWS1) subsets and describe detail information of them in Table 2.

3） DBLP dataset lists more than 2 million records in single XML file [14]. We select parts of file to generate dataset DBLP1, DBLP2 and DBLP3 with size in Table 2.

5.1. Implementation and Space Overhead

Our experiments are implemented in JDK 7 and the operating system is Cenos 6.3. Hardware environment for experiments is Dell server with a 2.27 GHZ 16-core CPU anda8G RAM.

Table 2. Dataset Description and Related Indexes

Data Source	Format	Data Set	# of file	Size	Lucence index	iExtractor index
20_Newsgroups	Unstructured data	NG1 NG2	2000 19997	4.46 MB 43.90 MB	6.25 MB 58.60 MB	7.03 MB 63.80 MB
University Web Site	HTML	UWS1 UWS2 UWS3	1583 60648 216495	13.0 MB 521 MB 1720 MB	9.62 MB 516 MB 1660 MB	10.86 MB 621 MB 2180 MB
DBLP	XML	DBLP1 DBLP2 DBLP3	1 1 1	20.40 MB 100 MB 1013 MB	20 MB 100 MB 990 MB	27.50 MB 123.70 MB 1420 MB

Table3 illustrate inverted index and iExtractor storage efficiency of different datasets by Formula $efficiency = datasetsize/indexsize$.We can see that iExtractor has similar or even better compression effectively like Lucerne.

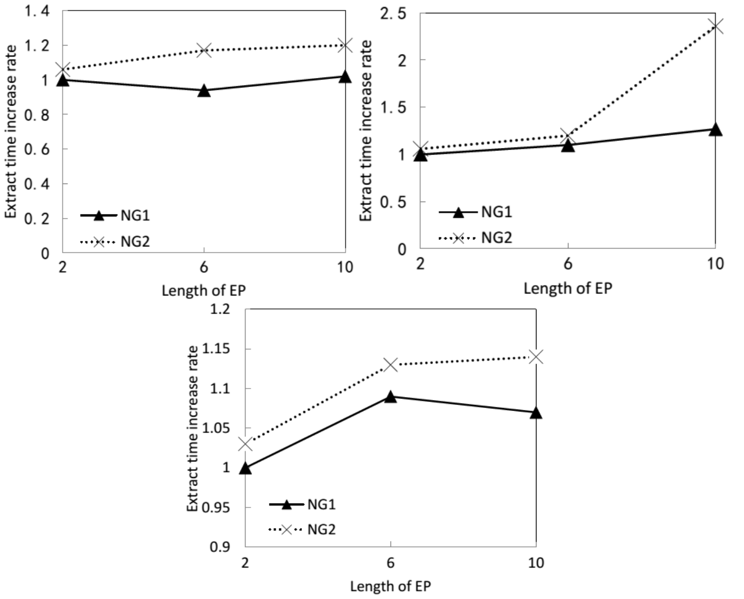
Table 3. Space Efficiency of Indexes

approach	NG1	NG2	UWS1	UWS2	DBLP1	DBLP2
Lucence	0.71	0.74	1.35	1.09	1.02	1.00
iEtractor	0.63	0.68	1.19	0.84	0.74	0.81

5.2. Extraction Efficiency Evaluation

In this section, we compare execute time efficiency of three extract models. (1) Text based Model (TBM) means extract item by exhausted parsing documents. We use Java regular pattern API to define extraction pattern and parse whole documents to grab result; (2) Index based Model (IBM) uses inverted index to gain term position values in query pattern and verify the correction of order by posting positions; (3) Signature based Model (SBM) uses iExtractor to find result.

To find the relationship between length of query pattern and extract performance, we choose 2, 6 and 10 terms EP to estimate execute time.

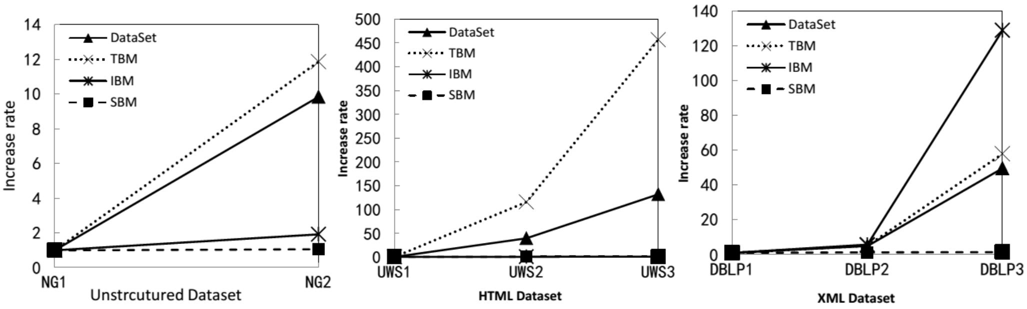


(a) TBM vs. EP (b) IBM vs. EP (c) SBM vs. EP

Figure 4. Execution Time Growth in Different Length of EP

Depicted Figures 4a and 4c, the length factor has less influence on TBM and SBM than on IBM. It is because IBM needs more indexes and calculation when EP is growing. TBM and SBM load content regardless EP length.

Next, we evaluate the influence of data format in three models. Execute time results are found with comparison of TBM, IBM and SBM and described. Motivated by the comparison of increment speed, we need to normalize result by Formulas size increment equals $Size_{target}$=$Size_{bench}$and Execute time increment equals ($Time_{target}$=$Time_{bench}$), where smallest value is treated as bench and other values are treated as target. Figure 5 gives the performance details in three types of dataset.



(a) Flat Data (b) HTML Data (c) XML Data

Figure 5. Execution Time Growth in Various Formatted Data Sets

As shown in Figure 5, we can see with the growth of dataset, execute time of SBM is low and stable in various data format and size. Execute time of TBM grows quickly with increasing data corpus size. IBM approach has good execute time in unstructured and HTML documents (Figures 5a and 5b). But growth rate becomes the worst in DBLP as Figure 5c depicted that is because DBLP file contains many high frequent tags which cost IBM more time to compare position for keeping correct order.

6. Related Work

There are several lines of work we build upon. In article [1], authors survey the major data extraction approaches and analyze relevant techniques. Typical IE tools mainly supply Brute-force parsing algorithms to gain data. For various sources like free web page or semi-structure XML file, text wrappers like Stalker [11] and WIEN [10] are useful to discover entities and relationships. Moreover, ontology and semantic web are used to extract information automatically [5].

IE tools with indexes could improve efficiency and support on-line extraction. For example, entity search [3] returns web entities instead of documents to user by indexes. TEXTRUNNER [6] use REVERB model to support relation and argument extraction. But element types in these approaches are limited and predefined off-line. Signature construct and compression methods are introduced in [2] to speed up file scanning and query evaluation.

7. Conclusion

Information Extraction provides a mechanism to extract items from free text. But the problem is that extractors need to scan the document brute-force to find interesting things resulting in poor performance. Moreover, text wrappers should be defined during off-line phase which makes the extraction process inflexible. In this paper, we propose a novel framework iExtractor, which is based on a signature file index, to speed up the text extraction process and promote flexible pattern extraction. Experimentally demonstrate that, in both unstructured and semi-structured datasets and with different data size, iExtractor provides efficient and stable performance than competing approaches.

Extensions of iExtractor with more complex extraction patterns and further compression of indexes could be interesting avenues for future research.

References

[1] C. H. Chang, M. Kayed, M. R. Girgis, and K. Shaalan, "A survey of web information extraction systems", Knowledge and Data Engineering, IEEE Transactions, vol. 18, no. 10, **(2006)**.

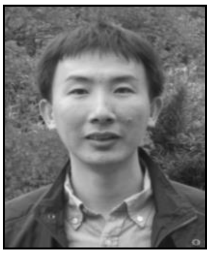
[2] Y. Chen and Y. Chen, "On the signature tree construction and analysis", Knowledge and Data Engineering, IEEE Transactions, vol. 18, no. 9, **(2006)**.

[3] T. Cheng, X. Yan, and K. Chang, "Entity rank: searching entities directly and holistically", Proceedings of the 33rd international conference on Very large data bases, **(2007)** September 23-27, Vienna, Austria.

[4] A. Doan, R. Ramakrishnan, and S. Vaithyanathan, “Managing information extraction, state of the art and research directions”, Proceedings of the ACM SIGMOD international conference on Management of data, **(2006)** June 26-29, Chicago, USA.
[5] D. W. Embley, Y. Jiang, and Y. Ng, “Record-boundary discovery in web documents”, ACM SIGMOD Record, vol. 28, no. 2, **(1999)**.
[6] A. Fader, S. Soderland, and O. Etzioni, “Identifying relations for open information extraction”, Proceedings of the Conference on Empirical Methods in Natural Language Processing, **(2011)** July 27-31, Stroudsburg, USA.
[7] C. N. Hsu and M. T. Dung, “Generating finite-state transducers for semi-structured data extraction from the web”, Information System, vol. 23, no. 8, **(1998)**.
[8] F. Keith, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese, “An improved construction for counting bloom filters”, In Lecture Notes in Computer Science, **(2006)**.
[9] A. Kirsch and M. Mitzenmacher, “Less hashing, same performance, building a better bloom filter,” Random Struct, Algorithms, vol. 33, no. 2, **(2006)**.
[10] N. Kushmerick, Wrapper induction for information extraction, University of Washington, **(1997)**.
[11] I. Muslea, S. Minton, and C. Knoblock, “A hierarchical approach to wrapper induction,” Proceedings of the third annual conference on Autonomous Agents, **(1999)**, New York, USA.
[12] S. Sarawagi, “Automation in information extraction and integration”, Tutorial of the 28th International Conference on Very Large Data Bases, **(2002)** August 20-23, Hong Kong, China.
[13] S. Liao and R. Grishman, “Filtered ranking for bootstrapping in event extraction”, Proceedings of the 23rd International Conference on Computational Linguistics, **(2010)** August 23-27, Beijing,China.
[14] M. Ley, “Dblp: some lessons learned”, Proceedings of the VLDB Endowment, vol. 2, no. 2, **(2009)**.

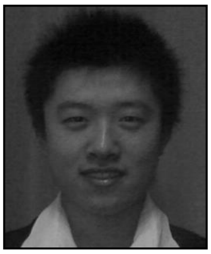
Authors

Canghong Jin, He get Ph.D. from Zhejiang University and works in Zhejiang University City College now. His research interests are information retrieval, data mining and big data.



Shiwen Cheng, He is a Ph.D. candidate in the Department of Computer Science and Engineering at UC Riverside. His research interests are web search, social network data, keyword search on structured data.

Minghui Wu, He is a Professor of Computer Science in Zhejiang University City College. His research interests are software engineering, program language and data mining.



Zemin Liu, He is a Master student of Computer Science in Zhejiang University. His research interests are streaming data, data mining and big data platform.