Eighth International Conference on Intelligent Systems Design and Applications

Compute the Term Contributed Frequency

Cheng-Lung Sung[1,2], Hsu-Chun Yen[1], Wen-Lian Hsu[2]
[1]*Dept. of Electrical Engineering, National Taiwan University*
[2]*Institute of Information Science, Academia Sinica*
{clsung, hsu}@iis.sinica.edu.tw; yen@cc.ee.ntu.edu.tw

Abstract

In this paper, we propose an algorithm and data structure for computing the term contributed frequency (tcf) for all N-grams in a text corpus. Although term frequency is one of the standard notions of frequency in corpus-based natural language processing (NLP), there are some problems regarding the use of the concept to N-grams approaches such as the distortion of phrase frequencies. *We attempt to overcome this drawback by building a DAG containing the proposed data structure and using it to retrieve more reliable term frequencies. Our proposed algorithm and data structure are more efficient than traditional term frequency extraction approaches and portable to various languages.*

1. Introduction

Term frequency (*tf*), the standard notion of frequency in corpus-based natural language processing (NLP), counts the number of times that a term, word, or N-gram appears in a corpus. N-grams are commonly used in statistical natural language processing; including Text Summarization, Information Retrieval, etc. However, one major problem is that t*f* cannot extract valid phrases that do not occur sufficiently frequently [3, 7]. Although this problem can sometimes be resolved by taking advantage of the inverse document frequency [8, 11], it still lacks the correct information about the actual frequency of a phrase's occurrence.

The distortion of phrase frequencies was first observed in the Vodis Corpus when the bigram "Rail ENQUIRIES" and trigram "BRITISH RAIL ENQUIRIES" were examined and reported by [6]. Both of them occur 73 times, which is a large number for such a small corpus. "ENQUIRIES" follows "RAIL" with a very high probability when it is preceded by "BRITISH." However, when "RAIL" is preceded by words other than "BRITISH," "ENQUIRIES" does not occur, but words like "TICKET" or "JOURNEY" may. Thus, the bigram "RAIL ENQUIRIES" gives a misleading probability that "RAIL" is followed by "ENQUIRIES" irrespective of what precedes it. This problem happens not only with word-token corpora but also with corpora in which all the compounds are tagged as units since overlapping N-grams still appear. Thus, term weighting in various NLP applications, such as Unknown Word Detection and Key Phrase Extraction [1], requires additional metrics [7].

The unknown word problems in Chinese, Japanese, and Korean (CJK) languages have increased in the last decade [1, 5, 12]. Many researchers have overcome the problems by using N-gram language models along with smoothing methods [10]. In addition, frequent strings are used in many NLP applications [9].

We define a new data structure called the *TCF-Node* and show how it can be used to construct a directed acyclic graph (DAG) in optimal time; i.e., the upper-bound of time complexity is bound to the building time of the term suffix array. After the term suffix array has been constructed and sorted, the *TCF-DAG* can be built in *O(n)* time.

As long as the *TCF-DAG* is created, we can traverse the whole *TCF-DAG* and extract the term contributed frequency (*tcf*) in linear time (*O(n)*).

2. The properties of the *TCF-Node*

Throughout this article we use T_x to denote any single node of *TCF-DAG*.

The *TCF-Node* has the following properties:

- It has three variables, *Word*, *Sum*, and *Self*;
- It has one prefix parent node at most;
- It has one suffix parent node as most;
- It has links to the parent nodes.

The *TCF-Node* also has the following characteristics:

- Every *Node* has exactly one common grandparent node if and only if it has a prefix parent node and a suffix parent node.
- A *Node* can only affect the *tcf* of its prefix, suffix and common grandparent nodes.

We discuss these properties and characteristics in the following subsections.

978-0-7695-3382-7/08 $25.00 © 2008 IEEE
DOI 10.1109/ISDA.2008.152

2.1. *Word*, *Sum* and *Self* Variables

We consider a finite nonempty string x = x[1..*n*] of length n ≥ 1. The term stored in the *TCF-Node* is represented by *Word*, which is a string value. We define *Sum* as the original term frequency, and *Self* as the term contributed frequency (*tcf*). In this paper, *Sum* does not change.

The *tcf* is the actual term frequency extracted from the given term. For example, in Vodis Corpus, the *Sum* of the term "RAIL ENQUIRIES" is 73. However, the *Self* value of "RAIL ENQUIRIES" is 0, since all of the frequency values are contributed by the term "BRITISH RAIL ENQUIRIES". In this case, we can see that 'BRITISH RAIL ENQUIRIES' is really a more frequent term in the corpus, where "RAIL ENQUIRIES" is not.

2.2. Definition of Prefix and Suffix parent nodes

For a given string x[1..*n*], we define its prefix string as x[1..*n*-1], and its suffix string as x[2..*n*]. Let us consider two *TCF-Nodes*, T_1 and T_2. If the value of T_1.Word is equal to the prefix of T_2.Word, we can link T_1 as the prefix parent node of T_2. On the other hand, if the value of T_2.Word is equal to the suffix of T_2.Word, we can link T_1 as the suffix parent node of T_2. Since the difference between prefix/suffix parent nodes and the child node is one character (or one word in word-based suffix arrays), there will be no room for any other parent nodes. Thus, a TCF-Node can have at most one prefix parent node and one suffix-parent node.

Hereafter, for simplicity, we refer to the value of Word for a specified TCF-Node as "TCF-Node."

2.3. Characteristics of the *TCF-Node*

In this section we explain how the *TCF-Node* can be used to compute the *tcf* value. First, we show that for any single *TCF-Node*, there is at most one common grandparent node respect to its prefix-parent, suffix-parent nodes. Then we show that for any single TCF-Node, it only affect the frequencies of its prefix-parent, suffix-parent and common grandparent nodes.

Lemma 1. If a given *TCF-Node* T_c has a prefix parent node T_{cp} with a suffix parent node T_{cps} and suffix parent node T_{cs} with a prefix parent node T_{csp}, then *i)* $T_{cps} = T_{csp}$, which is defined as T_{cg}; and *ii)* every T_c has at most one T_{cg}.

Proof. We prove this by using a simple diagram:

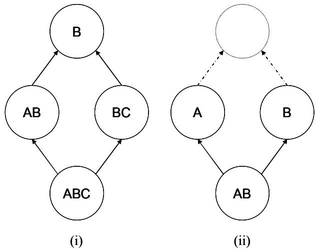

Figure 1. Parent nodes linking of *TCF-Node*

Figure 1 illustrates the parent nodes linking of *TCF-Node*. In Figure 1 (i), the values of T_c, T_{cs}, T_{cp} are "ABC", "BC", and "AB", respectively. We can easily see that $T_{cps} = T_{csp}$, where we denote the common grandparent node as T_{cg}. In Figure 1 (ii), since T_{cs} and T_{cp} contain no parent nodes, therefore there is no common grandparent node for T_c.

The result shows the *TCF-Node* can have links to its parent nodes. And for a *TCF-Node* contains both prefix and suffix parent nodes, it will at most has one common grandparent node. However, there are no back links (i.e.,. links to child nodes). Since for any string x[*i..j*], there are too many possible candidates to concatenate in either prefix or suffix substrings.

Lemma 2. The total frequency (*Sum*) of a given *TCF-Node* T_c will only affect the contributed frequency (*Self*) of its prefix parent node T_{cp}, suffix parent node T_{cs} and common grandparent node T_{cg}.

Proof. It is easy to show that the frequency of any *TCF-Node* T_c will accumulate to the frequency of T_{cp}. Intuitively, when we add the same word of T_c to the *Text*, the frequency of T_c will increase one. Since T_{cp} is the prefix of T_c, the same word of T_{cp} is also added. Likewise, we observe that the frequency of T_c affects the frequency of T_{cs}.

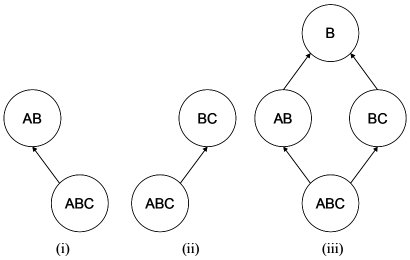

Figure 2. Frequency contribution of *TCF-Node*

According to Lemma 1, there is no extra parent node, so the increased frequency of T_c will only affect its two parent nodes. However, when we examine the common grandparent node T_{cg}, we find that the frequency of T_{cg} will increase twice, even though we only add T_c once. This is because, for prefix and suffix nodes, the grandparent node is also their suffix or prefix parent node. To resolve the problem, we simply need to subtract the added frequency or add the subtracted frequency from T_{cg} according to the operation on T_c.

Here we also explain the reason why the TCF-Node T_c. will not affect the frequencies of other nodes, such as the prefix parent of T_{cp} (prefix parent of prefix parent node) or suffix parent of T_{cs} node (suffix parent of suffix parent node), namely T_{cpp} and T_{css}, respectively.

As illustrated in Figure 3. We suppose there exists the node T_{cpp}, we can see the frequency of T_{cpp} is contributed by T_{cp} and itself. If we add the same word of T_c to the *Text*, the frequency of T_{cp} and T_{cpp} will both increase one. However, since there is no other link from T_c to T_{cpp}, we can treat the link between the T_{cp} and the T_{cpp} as the link between T_c and T_{cp}. Thus, the situation is just as Figure 2 (i), i.e. the frequency of T_{cpp} is affected by T_{cp} instead of T_c. Following the same procedure, we can also prove the frequency of T_{css} is affect by T_{cs} instead of T_c.

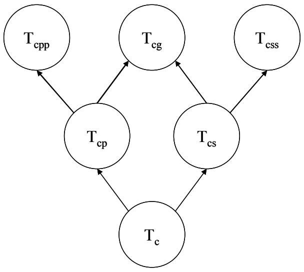

Figure 3. Two additional *TCF-Nodes*: T_{cpp} and T_{css}

The above two characteristics show that when *TCF-DAG* is constructed, it is easy to calculate the *tcf* for every *TCF-Node*. Since we only need to observe at most three parent nodes for each *TCF-Node*. We describe our implementation in the next section.

3. Implementation

In this paper we use a word-based suffix array approach [2]. We denote *Text* to be a text of length *n* over a constant-sized alphabet Σ. We further assume that certain characters from a constant-sized subset *W* of the alphabet act as word boundaries; thus, they divide *Text* in a natural sense into *k* tokens, called *terms* hereafter. We define the set of all suffixes of *Text* starting at the word boundaries as Suffix_I (*Text*) = $\{Text_{i..n} : i \quad I\}$. Then, the word suffix array *SA[1..k]* is a permutation of *I* such that $Text_{SA[i-1]..n} < Text_{SA[i]..n}$ for all $1 < i \leq k$; *that is*, *SA* represents the lexicographic order of all suffixes in Suffix_I (*Text*). We also adopt Yamamoto's approach [11] to compute the original term frequency.

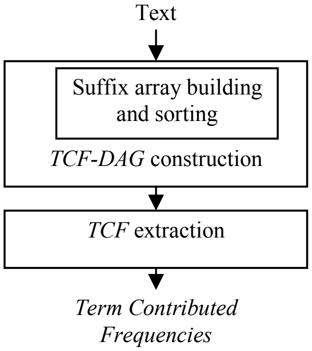

Figure 4. The process of extracting *TCF* from Text

Figure 4 presents the TCF calculation process, which comprises two phases: i) TCF-DAG construction and ii) TCF extraction.

Algorithm 1 (Construction of *TCF-DAG*).

```
SA[1..k] = buildSuffixArray (S)
SA'[1..k] = extractTermFreq(SA)
T[1..k] = {SA'[1] ...SA'[k]}
for i := 1 to k do
  T[i].Sum = T[i].Self = SA'[i].Freq
  j := 1
  while $i+j \leq k$ and isPrefix(T[i],T[i+j])
    j := j+1
    T[i+j].prefixParent ← T[i]
T[1..k] = sortBySuffix(T[1..k])
for i := 1 to k do
  j := 1
  while $i+j \leq k$ and isSuffix(T[i],T[i+j])
    j := j+1
    T[i+j].suffixParent ← T[i]
```

Algorithm 1 details how we construct the *TCF-DAG* from a suffix array *SA*. Once again we use [11] method to build the suffix array *SA* and retrieve the corresponding term frequency array *SA'* for a given text *S*. After *SA'* is created, where *SA'* is also sorted in lexicographic order, we assign the values in *SA'[1..k]* to *T[1..k]* sequentially. Before starting the iteration process, we assign the initial *Self* and *Sum* to the extracted term frequency from *SA'*.

To construct the *TCF-DAG*, we first iterate the whole of *T* and check if there exists any element that matches the prefix string of later elements. If *T[i]* is equal to the prefix string of T*[i+j]*, we link *T[i]* to *T[i+j]* as the prefix parent node of *T[i+j]*. When the whole of T has been traversed, we sort *T* by suffix strings of the words. Then we traverse *T* again. In contrast to previous steps, we check if any element in *T* matches the suffix string of subsequent elements. If *T[i]* is equal to the suffix string of T*[i+j]*, then we link *T[i]* to *T[i+j]* as the suffix parent node of *T[i+j]*. Once these two iterations are finished, the *TCF-DAG* is constructed.

Algorithm 2 (Extracting *TCF*).

```
T[1..k]
for i := 1 to k do
  $T_c$ := T[i]
  if T.prefixParent exists then
    $T_{cp} = T_c.prefixParent$
    $T_{cp}.Self = T_{cp}.Self - T_c.Sum$
  if $T_c$.suffixParent exists then
    $T_{cs} = T_c.suffixParent$
    $T_{cs}.Self = T_{cs}.Self - T_c.Sum$
  if $T_c$.grandParent exists then
    $T_{cg} = T_c.grandParent$
    $T_{cg}.Self = T_{cg}.Self + T_c.Sum$
for i := 1 to k do
  output T[i].Self
```

Algorithm 2 details how to extract the term contributed frequency from *TCF-DAG*. Although the *TCF-DAG* is a linked graph, we do not need to traverse it by a breadth-first search (BFS) or depth-first search (DFS). Our proposed method only iterates the *T[1..k]* array, which contains all *TCF-Node*s.

First, we check if any parent nodes of T_c exist. If any parent nodes (say T_{cp} or T_{cs}) do exist, we subtract the total frequency (*Sum*) of T_c from the contributed frequency (*Self*) of T_{cp} or T_{cs}. Second, if there exists a grandparent node T_{cg}, which implies T_c has both suffix-parent and prefix-parent nodes, we add the total frequency of T_c to the contributed frequency of T_{cg} once. We add one copy of the total frequency of T_c to compensate for the extra loss of T_{cg}, which has the contributed frequency, subtracted the value twice via its two child nodes (T_{cp} and T_{cs}).

Note that the *Self* value of any T_x can be reduced by several child nodes. However, by using the proposed algorithm, in the iteration, we only need to consider every T_x with its parent nodes. The Algorithm 2 simply iterates in a for-loop so one can easily estimate the time complexity as $O(n)$.

4. Conclusion

We have proposed a robust method that extracts *tcf* efficiently by creating a *TCF-DAG* via suffix array approaches and traverse the *TCF-DAG* in linear time $O(n)$. The upper bound of the time complexity depends on the sorting algorithm used to sort the suffix strings of the suffix array. In the future we will focus on the N-grams related NLP applications, such as Named Entity Recognition, Key-phrase Extraction and Text Summarization.

The work of [4] is similar to our proposed method in that it uses a database to store all terms' frequency and computes Chinese frequent strings in $O(n^2)$ time. However it is time-consuming because it needs to search every possible string in the database; thus, its string-length is limited.

Our proposed algorithm and data structure are more efficient than traditional term frequency extraction approaches and portable to various languages.

5. References

[1] L. F. Chien, "PAT-Tree-Based Adaptive Keyphrase Extraction for Intelligent Chinese Information Retrieval," *Information Processing and Management,* 1998.

[2] P. Ferragina and J. Fischer, "Suffix Arrays on Words," in *Combinatorial Pattern Matching*, 2007, pp. 328-339.

[3] Q. H. Le, P. Hanna, D. W. Stewart, and F. J. Smith, "Reduced n-gram models for English and Chinese corpora," in *Proceedings of the COLING/ACL on Main conference poster sessions* Sydney, Australia: Association for Computational Linguistics, 2006.

[4] Y.-J. Lin and M.-S. Yu, "Extracting Chinese Frequent Strings Without a Dictionary From a Chinese Corpus and its Applications," *Information Science and Engineering,* vol. 17, pp. 805-824, 2001.

[5] Y.-J. Lin and M.-S. Yu, "The Properties and Further Applications of Chinese Frequent Strings," *Computational Linguistics and Chinese Language Processing,* vol. 9, pp. 113-128, 2004.

[6] P. L. O'Boyle, "A study of an N-Gram Language Model for Speech Recognition." PhD thesis, Queen's University Belfast, 1993.

[7] T.-H. Ong and H. Chen, "Updateable PAT-Tree Approach to Chinese Key Phrase Extraction using Mutual Information: A Linguistic Foundation for Knowledge Management," in *Asian Digital Library Conference*, Taipei, Taiwan, 1999, pp. 63-84.

[8] P. Soucy and G. W. Mineau, "Beyond TFIDF Weighting for Text Categorization in the Vector Space Model," in *IJCAI*, 2005, pp. 1130-1135.

[9] B. Suhm and A. Waibel, "Toward Better Language Models for Spontaneous Speech," in *ICSLP*. vol. 2, 1994, pp. 831-834.

[10] J. Wu and F. Zheng, "On Enhancing Katz-Smoothing Based Back-Off Language Model," in *Spoken Language Processing*. vol. I, 2001, pp. 198-201.

[11] M. Yamamoto and K. W. Church, "Using suffix arrays to compute term frequency and document frequency for all substrings in a corpus," *Computational Linguistics,* vol. 27, pp. 1-30, March 2001 2001.

[12] K. C. Yang, "Further Studies for Practical Chinese Language Modeling," in *Department of Electrical Engineering*. vol. Master: National Taiwan University, 1998.