ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/45949392

Experiments with the Successor Variety Algorithm Using the Cutoff and Entropy Methods

Article *in* Information Technology Journal · January 2005

DOI: 10.3923/itj.2005.55.62 · Source: DOAJ

CITATIONS
3

READS
84

5 authors, including:

Riyad Al-Shalabi
Amman Arab University
43 PUBLICATIONS **435** CITATIONS
SEE PROFILE

Eyad Hailat
Wayne State University
14 PUBLICATIONS **30** CITATIONS
SEE PROFILE

Ahmad Ababneh
Prince Sattam bin Abdulaziz University
1 PUBLICATION **3** CITATIONS
SEE PROFILE

All content following this page was uploaded by Eyad Hailat on 22 August 2014.

The user has requested enhancement of the downloaded file.

Information Technology Journal 4 (1): 55-62, 2005
ISSN 1812-5638
© 2005 Asian Network for Scientific Information

Experiments with the Successor Variety Algorithm Using the Cutoff and Entropy Methods

Riyad Al-Shalabi, Ghassan Kannan, Iyad Hilat, Ahmad Ababneh and Ahmad Al-Zubi
Yarmouk University, Irbid , Jordan

Abstract: In the present study a system have developed that uses the Successor Variety Stemming Algorithm to find stems for Arabic words. A corpus of 242 abstracts have obtained from the Saudi Arabian National Computer Conference. All of these abstracts involve computer science and information systems. The study have set out to discover whether the Successor Variety Stemming Algorithm technique with the Cutoff Method can be used for the Arabic Language or not. In addition, the Successor Variety Algorithm have compared with the Cutoff and the Successor Variety with Entropy Method. Stemming is typically used in the hope of improving the accuracy of the search reducing the size of the index. The results of present research show that the Successor Variety Algorithm with the Cutoff Method is better than Successor Variety Algorithm with the Entropy Method. We have achieved an 84% level of correctness using the Cutoff Method, but a 64% level of correctness using the Entropy Method. These experiments were carried out using Visual Basic 6.0.

Key words: Successor, entropy, cutoff, stem, suffixes, prefixes

INTRODUCTION

Word stemming is an important feature supported by present day indexing and search systems. The idea is to improve recall by automatic handling of word endings by reducing the words to their word roots, at the time of indexing and searching. Stemming broadens our results to include both word roots and word derivatives. It is commonly accepted that removal of word-endings (sometimes called suffix stripping) is a good idea; removal of prefixes can be useful in some subject domains. A stemming algorithm is an algorithm that converts a word to a related form. One of the simplest such transformations is the conversion of plurals to singulars. One example is Porter's Algorithm. The Porter Stemmer is a conflation stemmer developed by Martin Porter at the University of Cambridge in 1980. The Porter stemming algorithm (or 'Porter stemmer') is a process for removing the commoner morphological and inflexional endings from words in English. It is the most effective and widely used. stemmer for English Porter's Algorithm works based on the number of vowel characters that are followed by a consonant character in the stem. This number (the Measure) must be greater than one for the rule to be applied. One of the limitations of this algorithm is that it can only be applied to text in the English Language.

Frequently, the user specifies a word in a query but only a variant of this word is present in a relevant document. This problem can be partially overcome with the substitution of stems for the words. A stem is the portion of a word that is left after the removal of its affixes (i.e., prefixes and suffixes). Stems are thought to be useful for improving retrieval performance, because they reduce variants of the same root word to a common concept. Furthermore, stemming has the secondary effect of reducing the size of the indexing structure because the number of distinct index terms is reduced. Many Web search engines do not adopt any stemming algorithm whatsoever. Frakes[1] distinguishes four types of stemming strategies: affix removal, table lookup, successor variety and n-grams. Table lookup consists simply of looking for the stem of a word in a table. Since such data is not readily available and might require considerable storage space, this type of stemming algorithm may not be practical. Successor variety stemming is based on the determination of morpheme boundaries, uses knowledge from structural linguistics and is more complex than an affix removal-stemming algorithm[2].

The goal of this study was to experiment with alternative stemming techniques using the successor variety approach. We all agree that a word in any language consists of a meaningful string of letters. An index in any language consists of a number of words in a related domain. A word may have nonstem letters in the beginning (prefix), in the middle (infix), or at the end (postfix) of the word. From the information retrieval point of view stemming is one technique to provide ways of finding morphological variants of search terms. It is used to improve retrieval effectiveness and to reduce the size of indexing files (Fig. 1).

Corresponding Author: Dr. Riyad Al-Shalabi, Yarmouk University, Irbid, Jordan E-mail: shalabi@yu.edu.jo

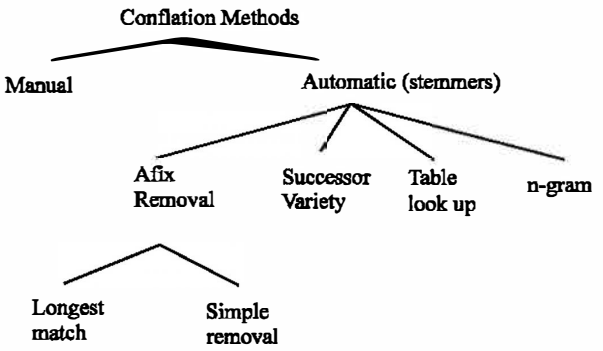

Fig. 1: Taxonomy of stemming algorithms[3]

We set out to test the Successor Variety Algorithm, which determines word and morpheme boundaries based on the distribution of phonemes in a large body of utterances. The successor variety of a string is the number of different characters that follow it in words in some body of text.

Successor variety stemmers[4] are based on work in structural linguistics, which attempts to determine word and morpheme boundaries based on the distribution of phonemes in a large body of utterances[5]. A stemming algorithm is a computational procedure that seeks to reduce all words with the same stem to a common[5]. Several algorithms have been developed to handle stems in English. Darwish[6] presents a good technique with an accuracy of 92.7% on 9,606 words. He stripped away the prefix and suffix. Beesley[7] presents a finite-state morphological analyzer for Arabic, which displays the root, pattern and prefixes/suffixes. The analyses are based on manually acquired lexicons and rules. Although his analyzer is comprehensive in the types of knowledge it presents, it has been criticized for its extensive development time and lack of robustness[6]. Most stemmers currently in use are iterative longest match stemmers, a kind of affix removal stemmer first developed by Lovins[8]. Increasing interest in the development of stemming algorithms has appeared for languages such as the Arabic language. The most notable of the efforts in this context are those reported by many authors[5,9-12]. In addition to Lovins[8] , iterative longest match stemmers have been reported by many authors[13-16].

There have been many experimental evaluations of stemmers. Salton and Young[3] examined the relative retrieval performance of fully stemmed terms against terms with only the suffix "s" removed. Hafer and Weiss[4] tested their stemmer against other stemming methods using the ADI collection and the Carolina Population Center (CPC) collection consisting of 75 documents and five queries. Van Rijsbergen[17] tested their stemmer[14] against the stemmer described by Dawson[16] using the Cranfield-1 test collection. Katzer *et al.*[18] examined the performance of stemmed title-abstract terms against six other document representations. Karen *et al.*[19] did a through review and study of stemming algorithms. Harman[20] used three stemmers-Porter, Lovins and S removal-on three databases-Cranfield 1400, Medlars and CACM and found that none of them significantly improved retrieval effectiveness in a IR system called IRX that ranks its output in order of relevance.

The latest natural language research focuses on building systems for the Arabic language with high performance and results relevant to the user needs. This gave rise to the idea of our project, to develop a system dealing with Arabic words. The system has been built based on the Successor Variety Stemming Algorithm, which determine the successor variety for a word, then uses this information to segment the word. We use the cutoff and entropy methods in our system to observe the effects of the successor variety approach on text in the Arabic Language and to discover whether it can be useful for special purposes for Arabic language applications.

Successor variety algorithm: The successor variety of a string is the number of different characters that follow it in words in some body of text. The successor variety of substrings of a term will decrease as more characters are added until a segment boundary is reached[2].

Successor variety stemmers[4] are based on work in structural linguistics, which attempted to determine word and morpheme boundaries based on the distribution of phonemes in a large body of text. The stemming method based on this work uses letters in place of phonemically transcribed utterances.

When this process is carried out using a large body of text, Hafer and Weiss[4] report 2000 terms to be a stable number), the successor variety of substrings of a term will decrease as more characters are added until a segment boundary is reached. At this point, the successor variety will sharply increase. This information is used to find the stem. When the successor varieties for a given word have been derived, the information must be used to segment the word. Hafer and Weiss[4] discuss four ways of doing this:

1. The cutoff method (this is the method we apply in our work):
- Some cutoff value (Threshold) is selected for successor varieties and a boundary is reached.
- The problem with this method is that if the threshold value selected is too small, incorrect cuts will be made; if it is too large, correct cuts will missed.

2. Peak and plateau method.
- A segment break is made after a character whose successor variety exceeds that of the character immediately preceding it and the character immediately following it.
- This method does not suffer from the problem of the cutoff method.

3. Complete word method
- A break is made after a segment if the segment is a complete word in the corpus.

4. Entropy method: (this is the other method we use in our experiments)
- Takes advantage of the distribution of successor variety letters. The method works as follows:
- Let $|D_{\alpha i}|$ be the number of words in a text body beginning with the *i* length sequence of letters α.
- Let $|D_{\alpha ij}|$ the number of words in $|D_{\alpha i}|$ with the successor *j* computed in step one.
- The probability that a word has the successor *j* is given by:
- The entropy of $|D_{\alpha i}|$ is: $\frac{|D_{\alpha ij}|}{|D_{\alpha i}|}$

$$H_{\alpha i} = \sum_{j=1}^{26} -\frac{|D_{\alpha ij}|}{|D_{\alpha i}|} \cdot \log_2 \frac{|D_{\alpha ij}|}{|D_{\alpha i}|}$$

- a cutoff value is selected and a boundary is identified whenever the cutoff value reached

A set of the above measures for predecessors can also be defined similarly.

In summary, the successor variety stemming process has 3 parts:
1. Determine the successor varieties of a word.
2. Use this information to segment the word using one of the methods above.
3. Select one of the segments as a stem.

The aim of Hafer and Weiss[4] was to develop a stemmer that required little or no human processing.

Full English Example

Test Word: *READABLE*

Corpus: ABLE, APE, BEATABLE, FIXABLE, READ, READABLE,
READING, READS, RED, ROPE, RIPE

The successor variety stem process is shown in Table 1.

Cutoff method
- segment when successor variety >= threshold
- consider threshold = 2 R|E|AD|ABLE

Table 1: Successor variety stem process

Prefix	Successor variety	Letters
R	3	E, I, O
RE	2	A, D
REA	1	D
READ	3	A, I, S
READA	1	B
READAB	1	L
READABL	1	E
READABLE	1	(Blank)

Peak and plateau method
- break at the character whose successor variety is greater than both its preceding and following character READ|ABLE

Complete word method
- break is made if the segment is a complete word in the corpus (READ)

Entropy method
- for *i* = 2, α = RE, $|D_{\alpha i}| = 5$
- for *j* = 'A', $|D_{\alpha ij}| = 4$
- for *j* = 'D', $|D_{\alpha ij}| = 1$
- *Haij* = - 1/5 * log2 (1/5) - 4/5 * log2 (4/5) = 0.46 + 0.26 = 0.72
  This value is low because "REA…" appears four times!

Our methodology Steps: The input data consists of a corpus of words and a reverse of each word.

Step one: Determine the successor varieties for a word. Take a word from the corpus and name it *FWord* and call the reverse *RevWord*:

1. (For Successor) Starting from *i*=1 to length of the *FWord*:
   a. For the rightmost *i* letters in *FWord*:
   i. Count the number of letters in the corpus that follow the first *i*th right most letters of *FWord*.
   ii. Store the successor value from the above step in a list called *SucList*.
2. (For Predecessor) Starting from *j*=length of the *RevWord*:
   a. For the rightmost *j* letters in *RevWord*:
   i. Count the number of letters in the Reversed corpus that follow the first *j*th right most letters of *RevWord*.
   ii. Store the Predecessor value from the above step in a list called *PrdList*.

Step two: Use this information to segment the word using:
1. Cutoff method
   a. Segmentation Process for Successor:

i. Set a variable named *SegSuc* of type string.
ii. For each character at position *k* in the *Fword*
   1. if the corresponding Successor value for *k* is greater than or equal to the threshold value (11) then add the contents of *SegSuc* to the list *SucSegList* and empty it.
   2. else add the character at position k to *SegSuc*.

iii. If the length of *SegSuc* is greater than zero, then add its contents to *SucSegList* and empty it

b. Segmentation Process for Predecessor:
i. Set a variable named *SegPre* of type string.
ii. For each character at position f in the *Revword*
iii. if the corresponding Successor value for f is greater than or equal to the threshold value (16) then add the contents of *SegPre* to the list *PreSegList* and empty it.
iv. else add the character at position f to *SegPre*.
v. If the length of *SegPre* is greater than zero, then add its contents to *PreSegList* and empty it.

Entropy method:

1. for each word in the corpus we find the following
   i. $|D_{\alpha i}|$ The number of words in a text body beginning with the *i* length sequence of letters α in *FWord* is computed and stored in step one.
   ii. $|D_{\alpha i}|$ The number of words in $D\alpha i$ with the successor *j* is computed in step one.
   iii. The probability that a word has the successor *j* is given by

$$\text{The entropy of } |D_{\alpha i}| \text{ is: } \frac{|D_{\alpha ij}|}{|D_{\alpha i}|}$$

$$H_{\alpha i} = \sum_{j=1}^{26} -\frac{|D_{\alpha ij}|}{|D_{\alpha i}|} \cdot \log_2 \frac{|D_{\alpha ij}|}{|D_{\alpha i}|}$$

The entropy value, calculated for each letter in the word, store the result in a list called *SucEntList*

Repeat this step for predecessors and store the resulting values in a list called *PreEntList*.

c. Segmentation Process for Entropy Successor:
i. Set a variable named *SegSucEnt* of type string.
ii. For each character at position *m* in the *Fword*
   1. if the corresponding Successor value for *m* is greater than or equal to the threshold value (2.7) then add the contents of *SegSucEnt* to the list *SucEntSegList* and empty it.
   2. else add the character at position *m* to *SegSucEnt*.

iii. If the length of *SegSucEnt* is greater than zero, then add its contents to *SucEntSegList* and empty it

d. Segmentation Process for Entropy Predecessor:
i. Set a variable named *SegPreEnt* of type string.
ii. For each character at position *n* in the *Revword*
   1. if the corresponding Successor value for *n* is greater than or equal to the threshold value (3.3) then add the contents of *SegPreEnt* to the list *PreEntSegList* and empty it.
   2. else add the character at position *n* to *SegPreEnt*.

iii. If the length of *SegPreEnt* is greater than zero, then add its contents to *PreEntSegList* and empty it.

Step three: Select one of the segments as the stem.

For each segment in *SucSegList* do the following:
1. If the segment occurs less than 16 times in words in the corpus then add this segment to the variable S1.

For each segment in *PreSegList* do the following:
1. If the segment occurs less than 16 times in words in the corpus then add this segment to the variable S2.

For each segment in *SucEntSegList* do the following:
1. If the segment occurs less than 16 times in words in the corpus then add this segment to the variable S3.

For each segment in *PreEntSegList* do the following:
1. If the segment occurs in less than 14 times in words in the corpus then add this segment to the variable S4.

Step Four: The first stem is the intersection of S1 and S2 and Store in variable *FirstStem*. The entropy stem is the intersection of S3 and S4 and Store in variable *EntropyStem*.

Step Five: Store the value of *FirstStem* in *FWord* and repeat steps One-to- Four using the new *FWord*, the resulting word is called *SecondStem*.

Step Six: If the length of the *SecondStem* is less than the length of *FirstStem*
then take the *SecondStem* as the stem.
else select either of them as a stem.

Example: The word "الحاسبات" is inserted in the text box. When the button labeled find, in the main form of our application, is pressed the program will perform the following steps:

Step one: Determine the successor varieties for this word: *FWord* assigned to "الحاسبات" and reverse it as

RevWord = "تابساحلا"

1. (For Successor) Starting from *i*=1 to length of the *FWord* (8):

-At i=1 the system searches the corpus and finds that, there are 16 letters following the *i*th segment and this is the successor variety we look for.. The Table 2 shows the successor variety for all *i*th segments:

Table 2: Successor variety for *i*th segments

I	Segment	Successor	Letters
1	ا	16	{ي,,ت}
2	اك	26	{,ص,ش,س,ر,د,خ,ح,ج,ث,ت,ب,ا,و ,ض,ظ,ط,ع,غ,ف,ق,ل,م,ن,هـ}
3	الح	6	{ل,ص,ر,د,ج,ا}
4	الحا	1	{س}
5	الحاس	1	{ب}
6	الحاسب	1	{ا}
7	الحاسبا	1	{ت}
8	الحاسبات	1	{Blank}

Store the successor value from the above step in a list called *SucList*.

2. (For Predecessor) Starting from *j*=length of the *RevWord* (8):

For the rightmost *j* letters in *RevWord*: Count the number of letters in the Reversed corpus that follow the first *j*th rightmost letters of *RevWord*.

-At *j*=1 the system searches the reversed corpus and finds that, there are 16 letters following the *j*th segment and this is the predecessor variety we look for. The Table 3 below shows the predecessor variety for all *j*th segments.

Table 3: Predecessor variety for all *j*th segments

I	Segment	Predecessor	Letters
1	ت	16	{ي,و,ه,ن,م,ل,ك,ق,ع,ص,ش,ر,خ,ج,ت,ا}
2	تا	17	{ي,و,ه,ن,م,ل,ك,ق,ف,غ,ع,ض,ر,د,ح,ج,ب}
3	تاب	3	{و,ن,س}
4	تابس	1	{ا}
5	تابسا	1	{ح}
6	تابساح	1	{ل}
7	تابساحل	1	{ا}
8	تابساحلا	1	{Blank}

Store the Predecessor value from the above step in a list called *PrdList*.

Step two: Use the information above to segment the word using:

1. cutoff method
   a. Segmentation Process for Successor:
   iv. Set a variable Named *SegSuc* of type string.
   v. For each character at position *k* in the *Fword*
      1. if the corresponding Successor value for *k* is greater than or equal to the threshold value (11) then add the contents of *SegSuc* to the list *SucSegList* and empty it.
      2. else add the character at position k to *SegSuc*.
   vi. If the length of *SegSuc* is greater than zero, then add its contents to *SucSegList* and empty it

The system will segment the word into three parts, the *SucSegList* looks like this:

Segment	Successor
ا	16 >= 11
ل	26 >= 11
حاسبات	6 < 11

b. Segmentation Process for Predecessor:
   vii. Set a variable named *SegPre* of type string.
   viii. For each character at position f in the *Revword*
   ix. if the corresponding Successor value for f is greater than or equal to the threshold value (16) then add the contents of *SegPre* to the list *PreSegList* and empty it.
   x. else add the character at position f to *SegPre*.
   xi. If the length of *SegPre* is greater than zero, then add its contents to *PreSegList* and empty it.

The system will segment the word into three parts, the *PreSegList* looks like this:

Segment	Predecessor
ت	16 >= 16
ا	17 >= 11
بساحلا	3 < 11

2. Entropy method: for each segment:
   Find the frequency of each letter after the *i*th most right segment in *FWORD*

   - When i=1, find the frequency of each letter that follow "ا". Find the sum of all the letters., Here the sum is 230.
   - For each letter:

Table 4: Frequency of each letter after the *i*th most right segment in *FWORD*

	Letter	Frequency	$Entropy_x$ formula	$Entropy_x$ value
1	ت	4	$-1*(4/230)*\log_2(4/230)$	-0.010166
2	ح	2	$-1*(2/230)*\log_2(2/230)$	-5.9526e-2
3	خ	1	$-1*(1/230)*\log_2(1/230)$	-3.411082e-2
4	د	2	$-1*(2/230)*\log_2(2/230)$	-5.9526e-2
5	ر	1	$-1*(1/230)*\log_2(1/230)$	-3.411082e-2
6	س	8	$-1*(8/230)*\log_2(8/230)$	-0.16850
7	ش	2	$-1*(2/230)*\log_2(2/230)$	-5.9526e-2
8	ض	2	$-1*(2/230)*\log_2(2/230)$	-5.9526e-2
9	ع	2	$-1*(2/230)*\log_2(2/230)$	-5.9526e-2
10	ق	1	$-1*(1/230)*\log_2(1/230)$	-3.411082e-2
11	ل	189	$-1*(189/230)*\log_2(189/230)$	-0.2327
12	م	3	$-1*(3/230)*\log_2(3/230)$	-8.1659e-2
13	ن	5	$-1*(5/230)*\log_2(5/230)$	-0.12007
14	ه	1	$-1*(1/230)*\log_2(1/230)$	-3.411082e-2
15	و	5	$-1*(5/230)*\log_2(5/230)$	-0.12007
16	ي	2	$-1*(2/230)*\log_2(2/230)$	-5.9526e-2
Sum		230		1.318368

- When i=2, find the frequency of each letter that follows "ال". Find the sum of all the letters. Here the sum is 189. For each letter.
- When i=3, find the frequency of each letter that follows "الح". Find the sum of all the letters Here the sum is 9. For each letter
- When i=4, find the frequency of each letter that follows "الحا". Find the sum of all the letters. Here the sum is 2. For each letter

	Letter	Frequency	Entropy$_x$ formula	Entropy$_x$ value
1	س	2	-1*(2/2)*log$_2$(2/2)	0
Sum		2		0

- When i=5, find the frequency of each letter that follows "الحاس". Find the sum of all the letters. Here the sum is 2. For each letter

	Letter	Frequency	Entropy$_x$ formula	Entropy$_x$ value
1	ب	2	-1*(2/2)*log$_2$(2/2)	0
Sum		2		0

- When i=6, find the frequency of each letter that follows "الحاسب". Find the sum of all the letters. Here the sum is 1. For each letter

	Letter	Frequency	Entropy$_x$ formula	Entropy$_x$ value
1	ا	1	-1*(1/1)*log$_2$(1/1)	0
Sum		1	0	

- When i=7, find the frequency of each letter that follows "الحاسبا". Find the sum of all the letters Here the sum is 1. For each letter

	Letter	Frequency	Entropy$_x$ formula	Entropy$_x$ value
1	ت	1	-1*(1/1)*log$_2$(1/1)	0
Sum		1		0

- When i=8, find the frequency of each letter that follows "الحاسبات". Find the sum of all the letters. A ssume that the sum is 1.

Find the frequency of each letter after the *i*th most right segment in *RevWORD*

- When *j*=1, find the frequency of each letter that follows "ت" in the reverse corpus. Find the sum of all the letters (sum of all words with this segment). Here the sum is 74.
- When *j*=2, find the frequency of each letter that follows "تا" in the reverse corpus. Find the sum of all the letters (sum of all words with this segment). Here the sum is 54.
- When *j*=3, find the frequency of each letter that follows "تاب " in the reverse corpus. Find the sum of all the letters. Here the sum is 5.
- When *j*=4, find the frequency of each letter that follows "تابس " in reverse corpus. Find the sum of all the letters (sum of all words with this segment). Here the sum is 2.

	Letter	Frequency	Entropy$_x$ formula	Entropy$_x$ value
1	ا	2	-1*(2/2)*log$_2$(2/2)	0
Sum		2		0

- When *j*=5, find the frequency of each letter that follows "تابسا " in the reverse corpus. Find the sum of all the letters. Here the sum is 2.

	Letter	Frequency	Entropy$_x$ formula	Entropy$_x$ value
1	ح	2	-1*(2/2)*log$_2$(2/2)	0
Sum		2		0

- When *j*=6, find the frequency of each letter that follows " تابساح " in the reverse corpus. Find the sum of all the letters. Here the sum is 1.

	Letter	Frequency	Entropy$_x$ formula	Entropy$_x$ value
1	ل	1	-1*(1/1)*log$_2$(1/1)	0
Sum		1		0

- When *j*=7, find the frequency of each letter that follows "تابساحل " in the reverse corpus. Find the sum of all the letters. Here the sum is 1.

	Letter	Frequency	Entropy$_x$ formula	Entropy$_x$ value
1	ا	1	-1*(1/1)*log$_2$(1/1)	0
Sum		1		0

- When *j*=8, find the frequency of each letter that follows " تابساحلا " in the reverse corpus. Find the sum of all the letters. Here the sum is 1.

	Letter	Frequency	Entropy$_x$ formula	Entropy$_x$ value
1	ت	1	-1*(1/1)*log$_2$(1/1)	0
Sum		1		0

Segmentation Process for Entropy Successor:

i. Set a variable named *SegSucEnt* of type string.
ii. For each character at position *m* in the *Fword*
   1. if the corresponding Successor value for *m* is greater than or equal to the threshold value (2.7) then add the contents of *SegSucEnt* to the list *SucEntSegList* and empty it.
   2. else add the character at position *m* to *SegSucEnt*.

The following table shows the *SucEntSegList*:

	Segment	Successor entropy
1	ال	4.0219≥2.7
2	حاسبات	2.41938<2.7

Segmentation Process for Entropy Predecessor:

i. Set a variable Named *SegPreEnt* of type string.
ii. For each character at position *n* in the *Revword*
    1. if the corresponding Successor value for *n* is greater than or equal to the threshold value (3.3) then add the contents of *SegPreEnt* to the list *PreEntSegList* and empty it.
    2. else add the character at position *n* to *SegPreEnt*.
iii. If the length of *SegPreEnt* is greater than zero, then add its contents to *PreEntSegList* and empty it The following table shows *PreEntSegLis*:

Step three: now, we attempt to select one of the segments as the stem:

1. For each segment in *SucSegList* do the following: If the segment occurs less than 16 times in words in the corpus then add this segment to the variable S1.

The comparison is shown:

	Segment	Occurs in	Comapison
1	ا	230	230 = 16
2	ل	50	50 = 16
3	حاسبات	1	1 < 16

The value if S1 is " حاسبات"

2. For each segment in *PreSegList* do the following: If the segment occurs less than 16 times in words in the reverse corpus then add this segment to the variable S2.

	Segment	Occurs in	Comapison
1	ت	74	74 = 16
2	ا	51	51 = 16
3	بساحلا	1	1 < 16

The value of S2 is reverse of " بساحلا " that is " الحاسب "

3. For each segment in *SucEntSegList* do the following:

If the segment occurs less than 16 times in words in the corpus then add this segment to the variable S3.

	Segment	Occurs in	Comapison
1	ال	189	189 = 16
2	حاسبات	1	1 < 16

The value if S3 is " حاسبات"

4. For each segment in *PreEntSegList* do the following:

If the segment occurs less than 14 times in words in the corpus then add this segment to the variable S4.

	Segment	Occurs in	Comapison
1	تا	54	54 = 16
2	بساحلا	1	1 < 16

The value of S4 is reverse of " بساحلا" that is " الحاسب"

Step four:

- The first stem is the intersection of S1 and S2 and Store in variable *FirstStem*. S1 is " حاسبات" and S2 is " الحاسب ". The intersection is " حاسب"
- The first stem is the intersection of S3 and S4 and Store in variable *EntropyStem*. S3 is " حاسبات" and S4 is " الحاسب". The intersection is " حاسب " and this is the stem.

Step five:

Store the value of *FirstStem*="حاسب" in *FWord* and repeat steps One-to- Four using new *FWord*, the resulted word called *SecondStem* = " حاسب". The result will be the same.

Step Six:

The length of the *SecondStem*(=4) is equal to the length of *FirstStem*

Then we can select any one of them.

The stem produced as " حاسب" and this is correct.

The implementation:

We have implemented this algorithm using the well-known programming language Visual Basic version 6.0. The purpose behind using this programming language is that it is easy to use, it facilitates the construction of an attractive user interface for our system and there are many manuals that describe the features of this language.

We focus mainly on the features dealing with the Arabic Language, also the ability to write applications that use an Arabic database. For our database, we used Microsoft Access 97 for storing and accessing the Arabic corpus. It is a good database engine and easy to learn and use.

Table 5: Computed stems for lists of words that share the same root

Test word	Computed stem cutoff method	Correct or not correct	Computed stem entropy method	Correct or not correct
الحاسب	حاسب	Correct	حاسب	Correct
احساب	حساب	Correct	حساب	Correct
الحاسبات	حاسب	Correct	حاسب	Correct
المحساب	حساب	Correct	حساب	Correct
حوسبة	حوسب	Correct	حوسب	Correct
محاسبة	حاسب	Correct	حاسب	Correct
والحسابات	حساب	Correct	حساب	Correct
والمحاسبة	حاسب	Correct	محاسب	Incorrect
الحواسيب	حواسيب	Correct	حواسيب	correct
حسب	حسب	Correct	حسب	Correct
حسيب	حسيب	Correct	حسيب	Correct
حاسبني	حاسبن	Incorrect	حاسبن	Incorrect
تحاسبا	حاسب	Correct	حاسبا	Incorrect
تحاسبوا	حاسبو	Incorrect	حاسبوا	Incorrect
حاسبه	حاسب	Correct	حاسب	Correct
حسابها	حساب	Correct	حساب	Correct
محسوب	حسوب	Correct	حسوب	Correct
الحاسبتان	حاسب	Correct	حاسبت	Incorrect
المحاسبتين	حاسب	Correct	حاسبت	Incorrect
المحاسبون	حاسبون	Incorrect	حاسبون	Incorrect

EXPERIMENTS AND RESULTS

we describe experiments done to test the correctness of our work. The results are shown in Table 5.

CONCLUSIONS

After applying these algorithms to 2000 Arabic words, we conclude that we can apply the Successor Variety Algorithm with the Cutoff Method to the Arabic Language since we have achieved an 80% level of correctness. On the other hand, we have achieved a 75% level of correctness by applying the Entropy Method.

Several advantages of the Successor Variety Algorithm can be observed; the most important one is the ability to find a stem without the need to use a dictionary. Another advantage is that it can be used in several domains; it is basically (domain independent).

REFERENCES

1. Frakes, W., 1992. Stemming Techniques. Chapter 6 in Frakes and Baeza-Yates, 1992.
2. Fox, C., 1992. Lexical Analysis and Stoplists. Chapter 5 in Frakes and Baeza-Yates, 1992.
3. Salton G. and C.S. Yang, 1980. Contribution to the Theory of Indexing. American Elsevier, New York, 1980.
4. Hafer, M.A. and S.F. Weiss, 1974: Word segmentation by letter successor varieties. Information Storage and Retrieval, 10: 371-385.
5. Mustafa Suleiman and Qasem Ahmad Al-Radaideh, 2001. Arabic word stemming using letter successor and Predecessor Variety, ACIT 2001.
6. Darwish, K., 2002. Building a shallow Arabic morphological analyzer in one day. Proceedings of the Workshop on Computational Approaches to Semitic Languages. ACL, Philadelphia, pp: 47-54.
7. Beesley, K. and L. Karttunen, 2000. Finite-state non-concatenative morphotactics. Proceedings of the ACL, Hong Kong, pp: 191-198.
8. Lovins, J., 1968. Development of a stemming algorithm. Mechanical Translation and Computational Linguistics, 11: 22-31.
9. Al-Fedaghi, S. and F. Al-Anzi, 1989. A new algorithm to generate Arabic root-pattern forms. In Proceedings of the 11th National Computer Conference and Exhibition, March, Dhahran, Saudi Arabia, pp: 04-07.
10. Al-Kharashi, I. and M.W. Evens, 1994. Words, stems and roots in an Arabic information retrieval system. J. American Soc. Inform. Sci., 45: 548-560.
11. Al-Shalabi, R. and M.W. Evens, 1998. A computational morphology system for Arabic computational approaches to semitic languages. Workshop, COLING 98, Montreal, Canada, pp: 58-65.
12. Khoja, S., 1999. Stemming Arabic text. Lancaster, U.K., Computing Department, Lancaster University. www.comp.lancs.uk/ computing/users/khoja/ stemmer.ps.
13. Paik, W., 1994. Chronological Information Extraction System (CIES), Dagstuhl-Seminar-Report: 79 on Summarizing Text for Intelligent Communication, Endres-Niggemeyer, B., Hobbs, J. and Jones, K.S. (Eds.), Wadern, Germany: IBFI
14. Porter, M.F., 1980. An algorithm for suffix stripping. Program, 14: 130-137.
15. Salton, G., 1968. Automatic Information Organization and Retrieval. New York, NY McGraw-Hill.
16. Dawson, J.L., 1974. Suffix removal for word conflation. Bulletin of the Association for Literary and Linguistic Computing, 2: 33-46.
17. Van Rijsbergen, C.J., 1979. Information Retrieval. London, UK: Butterworths.
18. Katzer, J., M.J. McGill, J.A. Tessier, W. Frakes and P. Das-Gupta, 1982. A study of the overlap among document representations. Information Technology: Research and Development, 2: 261-274.
19. Karen, S.J., S. Walker, S.E. Robertson, 2000. A probabilistic model of information retrieval: A development and comparative experiments-Part 2. Inf. Process. Manage., 36: 809-840.
20. Harman, D., 1991. How effective is suffixing? J. American Soc. Inform. Sci., 42: 7-15.

View publication stats