1-1-2003

The design, implementation, and effectiveness of a Farsi word stemmer

Russell Beckley
University of Nevada, Las Vegas

THE DESIGN, IMPLEMENTATION, AND EFFECTIVENESS A FARSI WORD STEMMER

by

Russell Beckley

Bachelor of Arts - Economics
University of Montana, Missoula
1993

A thesis submitted in partial fulfillment
of the requirements for the

Master of Science Degree in Computer Science
Department of Computer Science
Howard R. Hughes College of Engineering

Graduate College
University of Nevada, Las Vegas
December 2003

UNLV

Thesis Approval

The Graduate College
University of Nevada, Las Vegas

August 11, 2003

The Thesis prepared by

Russell Beckley

Entitled

The Design, Implementation, and Effectiveness of a Farsi Word Stemmer

is approved in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

Examination Committee Chair

Dean of the Graduate College

Examination Committee Member

Examination Committee Member

Graduate College Faculty Representative

ABSTRACT

**The Design, Implementation,
and Effectiveness of a
Farsi Word Stemmer**

by

Russell Beckley

Dr. Kazem Tagva, Examination Committee Chair
Professor of Computer Science
University of Las Vegas, Nevada

A stemmer for the Farsi language has been designed, implemented, and evaluated. The stemmer uses Farsi morphology to remove affixes, producing effective stems. The implementation is written in C, using strings of unicode-encoded characters to represent Farsi words. It is meant to enhance the Farsi information retrieval system currently being developed at the Information Science Research Institute at the University of Nevada at Las Vegas. The effectiveness of the Farsi stemmer and stopword list on recall/precision was tested.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

Thanks to Drs. Kazem Taghva and Mohammad Sadeh, whose knowledge of Farsi made the stemmer and this thesis possible. I don't know enough Farsi to order a کباب

Thanks to Drs. Kazem Taghva, Tom Nartker, Ajoy Datta, and Emma Regenntova, for serving on my graduate committe and for reviewing this thesis.

Thanks to Dr. Kazem Taghva, Jeff Coombs, and Alan Condit for technical assistance and insight.

Thanks to Heidi and Zoë for putting up with three and a half years of this.

CHAPTER 1

INTRODUCTION

The *Farsi* language, also known as *Persian*, is spoken and written primarily in Iran and Afghanistan. The Farsi stemmer removes suffixes and prefixes from Farsi words, producing word stems. It was devised to improve information retrieval on a collection of Farsi language documents. It is a component of the Farsi information retrieval system developed at ISRI.

What is stemming?

To stem a word is to replace it with a more basic term, possibly its root. For example, stemming the term *interesting* may produce the term *interest*. Though the stem of a word might not be its root, we want all words that have the same stem to have the same root. For example,*interes* can be the stem of *interesting* if all terms whose stem is *interes* also derive from *interest*.

In an information retrieval environment, reducing words to stems allows the search engine to identify multiple derivations of given roots. If stemming enhances retrieval, it is because various words with the same root are used in similar contexts. For example, if you want to find a document about rabbits, a context in which *rabbit* occurs is as likely to interest you as a context in which *rabbits* occurs. A stemming search engine recieving a query that includes *rabbit* will look for all occurrences of *rabbit* and *rabbits*.

Why Build a Farsi Stemmer?

Researchers at ISRI have implemented an information retrieval system to search a Farsi document collection. Because of the morphological similarities of English and Farsi, and the fact that stemming benefits English document searches, we thought it would be worthwhile to develop a Farsi stemmer for Farsi document searches.

Like English, Farsi has an affixitive morphology. In other words, suffixes and prefixes are concatenated to Farsi words to modify the meaning. Like English nouns, Farsi nouns are appended to signify possession and plurality. Farsi verbs are modified to signify tense, person, negation, and mood. A verb may have scores of variations.

Stemmimg's effect on English document searches has been tested extensively. In some contexts, English stemmers such as Lovins and Porter improve precision/recall. [3] Similar results on a Farsi document collection require a Farsi stemmer.

Identifying Farsi Stopwords

Stopwords are words that do not correlate to any subject, e.g. *a*, *was*, and *we*. Most stopwords are very common. Many are effectively syntactic rather than semantic. Like English, Farsi has many stopwords.

Because stopwords do not aid retrieval, many information retrieval systems, including ours, identify them in order to screen them. Because several Farsi stopwords are verbs, each with dozens of variations, the stopword identification system uses the Farsi stemmer.

Some Technical Details

To accomodate the information retrieval system, the Farsi stemmer was implemented in the C programming language. Its input and output are strings of 32-bit values representing Farsi and Arabic characters defined by the unicode 3.2 standard. It has been tested on the Solaris operating system, but should work on a variety of platforms.

CHAPTER 2

INFORMATION RETRIEVAL BASICS

The term *Information Retrieval* denotes the search for specific information in a loosely structured collection of documents. Such documents are usually text files, but may be images, music files, or anything else. Information retrieval systems for text documents usually use an index, also known as an *inverted file*, to find occurrences of words in the collection. An index is a list of all the terms in a collection, each associated with the documents in which they are found. The index used by an information retrieval system, unlike a book index, lists every occurrence of every non-stopword, refers to multiple documents, and may associate any amount of information with each word.

The Problem of Information Retrieval

There are at least two reasons information retrieval is not an exact science. First, computers do not understand most documents. A loose collection of documents differs from a database, where information is rigidly structured in terms of first order formal logic, and so every piece of data, and every relation between pieces of data, have explicit interpretations. In contrast, most documents are written by humans for humans. To understand a document, one must have an *a priori* understanding of the world. While composing a document, most authors do not consider the eventual computerized interpretation of their document. The result is that most information retrieval systems reduce documents to weighted lists of words. Another reason that information retrieval is not an exact science is that computers cannot

understand a person's information requirements by reading a query. However, to give computers due credit, it is often difficult for humans, too, to understand documents and information needs.

To illustrate the difficulties of information retrieval, consider the process of creating queries. When a person queries a document collection, she believes that she will benefit from some of the documents, and believes that she will benefit more from some documents than from others. We can imagine that for each user, each document has a fixed utility, $U(u, d)$. If a retrieval system knew the utility of each document for the user, it would do well to sort them in desccending order and return the result. But an information retrieval system does not have this information. Even the user cannot know the utility of a document until she has read it.

To express her information needs, a user writes a query. But more than $U(u, d)$ determines the composition of the query. Two users with the same utility function may compose different queries. This difference results from differences in searching experience, differences in language proficiency, differences in personality, etc.

The goal of IR is to invert the query generation process, that is, to use a query to estimate a user's utility for each document. This is difficult because, as noted, query generation depends on more than $U(u, d)$. In reality, information retrieval systems use some measure of similarity between queries and documents to sort the collection.

$$Similarity : \{queries\} \times \{documents\} \rightarrow \Re$$

Usually, a crucial factor in computing similarity is the number of shared words relative to document size. Stemming increases the similarity measurements by equating a word variation with its root. The next chapter explores this concept more thoroughly.

The Farsi Information Retrieval System

Despite the ubiquity of the internet, search engines for semitic languages, such as Arabic, Farsi, and Hebrew, are scarce. To this end, researchers at ISRI have developed a Farsi

information retrieval system.

The lack of search engines for semitic languages results in part from the fact that the semitic alphabets have no representation in ASCII, the standard for computers and internet sites. Most PCs are not configured to compose queries in non-ASCII character sets. Uniform character mapping is essential to effective indexing and query processing. If the queries and the collection have different representations of the same term, their similarity will be underestimated. Unfortunately, Farsi internet documents use a variety of character mappings. To unify this variety, all text processed by the Farsi information retrieval system is translated to sixteen-bit unicode standard 3.2.

CHAPTER 3

FUNDAMENTALS OF STEMMING

Stemming enhances information retrieval primarily by allowing each query term to represent many terms with the same root. Secondarily, it saves index space and, therefore, saves query processing time. [1]

For example, suppose we have the following document, D_i: "He was running defensively down the hill" By stemming and removing stopwords, it becomes "run defense down hill". By putting it in vector form, the ordering disappears, and we get {defense, down, hill, run}. After the same process, the document D_j: "From the hill, we defended the downed runner." . becomes the same. The query "runs defensive" becomes {run, defense} and is equally similar to both documents.

We can think of stemming as a many-to-one mapping from any word to its stem:

$$stem : \{words\} \rightarrow \{stems\}$$

Or, we can formulate it as a many-to-one mapping from a set of words to the set of stems of all those words:

$$stem : 2^{\{words\}} \rightarrow 2^{\{stems\}}$$

If we want the second formulation to represent the stemming of queries and documents, we can tweak the definition of *set* to recognize duplicate elements. Under this definition, many information retrieval systems (including that on which the Farsi stemmer was tested) reduce queries and documents to sets of words. These functions are many-to-one because we expect some stems to represent multiple terms. Otherwise, the stemmer would not benefit the information retrieval system.

Terms

In this chapter, there are several key terms that deserve attention:

root: The character string from which a word derives.

de facto stem: The result of applying a stemming algorithm to a word. A *de facto* stem may or may not be a linguistic stem. A *de facto* stem may or may not be useful.

conflatet: To represent multiple terms with one stem, e.g., to represent *runner*, *run*, and *running* with the stem *run*.

conflation classt: A set of words represented by the same stem. For example, *Runner*, *run*, and *running* are members of the same conflation class. We can use this definition to approximate an inverse to the stemming function:

$$conflationClass : \{words\} \rightarrow 2^{\{words\}}$$

As we can for the stem function, we can extend this to operate on sets, or queries and documents:

$$conflationClass : 2^{\{words\}} \rightarrow 2^{\{words\}}$$

ideal conflation classt: For a given root, an ideal conflation class contains exactly all of its variations.

variationt: If r is the root of w, then w is a variation of r.

effective stemt: The *de facto* stem of a conflation class all of whose members have the same root.

ideal stemt: A symbol identifying exactly one ideal conflation class. A stemmer that always returns the ideal stem of the input word is a darn good stemmer.

contextt: A contiguous body of text with consistent subject matter.

contextual distributiont: In a given document collection, the set of contexts in which a word occurs.

theoretical contextual distributiont: Suppose that for any context, C, and any term, t, there is a constant probability,p, that t occurs in C; $p = P(t|C)$. The *theoretical contextual distribution* is the distribution of occurrences of t, over all contexts under infinite trials, determined by p. It is a general characterization of the contexts in which a word tends to occur. If stemming enhances retrieval, it is because various words with the same stem have similar theoretical contextual distributions. Two words with nearly identical theoretical contextual distrbution may, in a given collection, have somewhat different contextual distributions.

A crucial factor in computing similarity between documents and queries is the number of shared words relative to document size. Stemming effects similarity measurements by equating a word variation with its root. If a query q contains the word *monster* and document d contains the word *monsters*, consistent, good stemming will increase their similarity:

$$Similarity(q, d) < Similarity(stem(q), stem(d))$$

More generally, if d is any document, q is any query, and the stemmer is consistent:

$$Similarity(q, d) \leq Similarity(stem(q), stem(d))$$

This relationship holds because the stem function is many-to-one.

Given a root, the choice of variation scarcely correlates to the subject of the document in which it is found. For example, the probability that a document is relevant, given that it contains the word *bloviate*, is roughly equal to the probability that a document is relevant, given that it contains the word *bloviates* . Therefore, information retrieval systems do not typically recognize the syntactic role of a word in a sentence, only its presence.

We can model the situation more formally. Suppose each root, r, has one or more variations, r_i, with equivalent theoretical contextual distributions. Also, suppose the root occurs

if and only if one of its variations occur.. Further, suppose that the root occurs with a certain probability in a given context C, $P(r|C)$, and that each particular variation, r_i, has a probability independent of context, $P(r_i|(r \wedge C)) = P(r_i|r)$. Then the probability that the subject of C is S, given the presence of r_i, is equal to the probability that the subject of the document is S, given the presence of r_j , i.e., $P(S|r_i) = P(S|r_j) = P(S|r)$.

Now suppose there exists a root, x, with exactly one variation, x_1; i.e., $|variations(x)| = 1$. Then every time the root occurs, x_1 occurs. Without stemming, if x occurs in a query, we can access every occurrence of x in the collection.

However, if a root, y, has more than one variation, the occurrences of y do not have the same form. If y has several common variations, the probability of a given variation in a given context is much lower than the probability of y, i.e. $P(y_i|C) = P(y_i|y) \times P(y|C)$, (noting that $P(y|y_i) = 1$). When y occurs in a query, it will occur as one variation, say y_j. Without stemming, though all variations of y are equally relevant in this model, the information retrieval system will identify only occurences of y_j.

Now suppose we use a perfect stemmer—that is, one that always gives the true root—to index the collection and to process the queries. Then, even in the case where a root has several variations, information retrieval works as well as in the case where a root occurs in one form. Each occurrence of a variation of root r will be indexed as r. Also, each occurence of any variation of r in a query will be converted to r. Therefore, any variation of r in the query will reference every variation of r in the collection.

This model does not represent any reality of which I am aware. The contexts in which variations of the same root occur often differ. Also, finding the root of a given word is difficult.

Stemming Errors

In practice, stemming is error prone, though most errors are tolerable. Stemming errors generally fall under two broad categories: underconflation and overconflation. Underconflation denotes situations in which two words are variations of the same root, but do not have the same *de facto* stem. Overconflation occurs when two words share a *de facto* stem but do not have similar theoretical contextual distributions. These definitions are not symmetric.

Underconflation has multiple causes. There may be an affix the stemmer is not programmed to identify. If it is not programmed to look for the suffix *-ing* it will not conflate *pour* with *pouring*. Another cause is that many common word derivations are exceptional. For example, the plural for *man* is *men*; this problem leads to underconflation in many gender specific terms such as spaceman, doorman, policeman, etc,as well as woman. Another cause of underconflation is that a minimum stem length rule, enforced to prevent overconflation, may prevent valid suffix removal. The most severe underconflation is an absence of stemming, which, in most information retrieval instances, is no fatal flaw.

Overconflation, too, has multiple causes. Often, what appears to be a suffix is not a suffix. For example, to a machine, the term *ceiling* may appear to have the suffix *ing* to signify the continuous tense, though removing it leaves *ceil* which is not its root; if the stem of an unrelated term is *ceil*, the result is overconflation. Another problem is that two words may have the same root, but do not have similar theoretical contextual distributions. For example, *President* shares a root with *preside*, though their contextual correlation is slight. If the members of a conflation class have unrelated theoretical contextual distributions, the effect of stemming is probably undesirable. There are cases in which overconflation occurs without a stemmer, as when words with identical spelling have completely different meaning. i.e. homographs. Perfect overconflation is to represent every word in the collection with the same stem, which would render the infromation retrieval system useless.

Consider the conflation two words with similar theoretical contextual distributions, and

a query for which they are not similar enough. Overconflation might occur for one query but not another. For example, a query including *residential address* might conflate correctly to *reside address*, but a query with *residential area* might overconflate when stemmed to *reside area*.

When designing a stemmer, there is a tradeoff between overconflation and under conflation. For example, if your stemmer often underconflates, you might change the rules to remove suffixes in more cases. Plausibly, these new rules will lead to overconflation that did not previously exist. There are suffixes that are useful to remove in some cases, but harmful to remove in others. To always remove such a suffix will cause overconflation, and to always leave it will cause underconflation. In most cases there is no reliable way to distinguish good removal from bad removal. Every rule has unforeseeable exceptions. This may be why many stemmers overconflate and underconflate.

Moreover, successful stemming does not guarantee successful retrieval. For a stemmer to improve an information retrieval system action, we must have query q entered by user u, and documents d and e such that:

$$U(u,d) > U(u,e),$$

$$Similarity(q,d) < Similarity(u,e),$$

and

$$Similarity(stem(q), stem(d)) > Similarity(stem(q), stem(de)),$$

Given the uncertainties of the relationship between the user's information need and the word set comprising a document, some costs/benefits of stemming are accidental. This allows the possibility that, in a case where the correct order is produced without stemming, even perfect stemming will cause a less relevant document to score higher than a more relevant document. Without many trials, the utility of a stemmer is not known.

Stemming Practice

Usually, when an information retrieval system uses a stemmer, all terms in the collection are stemmed before being indexed, and all query terms are stemmed before comparison to the index terms. Therefore, a stemmed index is a record of occurrences of conflation classes, each represented by its *de facto* stem. In psuedo-mathematical jargon, the information retrieval system uses $Similarity(stem(query), stem(collection))$. Alternatively, an information retrieval system might index every word without stemming, and process queries by stemming each word, produce each variation of each stem, then search the index for each variation: $Similarity(conflationClassOf(query), collection)$. A third approach is to, for every occurence of stem s, index every variation of s, then use the raw query: $Similarity(query, conflationClassOf(stem(collection)))$.

CHAPTER 4

THE FARSI LANGUAGE

Farsi is the language of ancient Persia and modern Iran. Many words used by Farsi writers are Arabic, which fused with Farsi when Islam entered Persia. Of 32 letters in the Farsi alphabet, 28 are in the Arabic alphabet. Followers of Islam are expected to read the Koran in Arabic. Nonetheless, Farsi morphology is mostly distinct from Arabic morphology, though some of the affixes used in Farsi are Arabic.

Farsi morphology, like English morphology, is affixitive. Words are derived and inflected by adding prefixes and suffixes. The meanings of Farsi suffixes are similar to those of English; for example, there are Farsi suffixes to signify plurality, possession, comparison, and tense.

Farsi, like Arabic, is read from right to left. What appears to be the end of a word to an English reader is actually the beginning. Prefixes might at first appear to be suffixes.

Suffixes and Prefixes

I use the terms *suffix* and *prefix* in their linguistic sense, not as they are defined in the theory of languages, as initial and terminal substrings. A suffix is a string affixed to the end of a word to change its meaning, tense, or syntactic function. Therefore, a terminal substring is not necessarily a suffix; for example, any string is a terminal substring of itself. Furthermore, a suffix is not necessarily a terminal substring, as when a plural suffix precedes a possessive suffix. Likewise, a prefix is not always an initial substring.

Verbs

The roots of Farsi verbs are imperative forms derived from the infinitive forms, which end in تن or دن . To specify the person of a Farsi verb, the ending ن is replaced by other suffixes. For example, the infinitive گرفتن means "to take". Removing ن gives the past tense. The suffix یم specifies plural first person (*we*). Therefore, گرفتیم means"we took".

When encountering a verb, the Farsi stemmer does not output the infinitive form, therefore, it does not output the root. It removes the part that has replaced the ن in the word formation. For example, if گرفتیم is input, the stemmer outputs گرفت .

The suffixes of Farsi verbs indicate person, number, and tense. For example, to say "we went" we remove ن from رفتن ("to go"), yielding رفت ("went"), then add the suffix یم resulting in رفتیم . The verbal suffixes are :

هایم	past perfect plural first person
هاید	past perfect plural second person
هاند	past perfect plural third person
هام	past perfect singular first person
های	past perfect singular second person
هاست	past perfect singular third person
م	singular first person
ی	singular second person
د	singular third person
یم	plural first person
ید	plural second person
ند	plural third person

Farsi verbs are prefixed to specify tense, mood, and negation. The prefixes recognized by the Farsi stemmer are ن , ب, and می , which signify negation, imperative mood, and continuous tense, respectively. They are used in combination according to these rules:

Prefix ب is not prefixed to any word that is also prefixed by ن or می.

If ن is found in conjunction with می , ن precedes می.

Expressed more succinctly:

<stem>({می} {ن}) | (ب)

For example, the negative past continuous tense of داشت, ("had"), is نمیداشت . For another example, نمیرفتیم means "we were not going".

To complicate things, to add ن or ب to the beginning of a word that starts with a vowel, an ی is infixed between the prefix and the stem. For example, the negative of اندازید is نیاندازید ("Don't throw it".). The extended language of prefixes is expressed by:

<stem>({می} {{ی}ن})|({ی} ب)

Nouns

Nouns may have stacked suffixes, according to the pattern:

{non-plural/non-possessive}{plural}{possessive} <stem>

The possessive nominal suffixes, like verbal suffixes, signify person:

م	singular first person
ت	singular second person
ش	singular third person
مان	plural first person
تان	plural second person

شان plural third person

Farsi nouns can be made plural by adding one of the suffixes ات, ان, or ها. For example:

ده (village) > دهات (villages) دشان (hand) > دشان (hands) or دشها (hands)

Some Farsi nouns that end with a silent ه or a ی are pluralized by removing the end letter and adding ج to the end before adding ات For example:

میوه > میوجات (fruits)

Other Farsi nouns ending in و or ا are pluralized by appending ی before appending ان For example:

جنگیو > جنگیویان (warriors) دانا > دانایان (learned people)

Other Farsi nominal suffixes are:

کار agent

گار agent

نده agent

[6] [4]

CHAPTER 5

THE ALGORITHM

The Farsi stemmer receives a Farsi word and determines whether any of its terminal substrings are Farsi suffixes. If so, and if the input word has enough characters, the suffix is removed. Then, depending on what class of suffix is matched, the stemmer attempts to find a prefix, find another suffix, or do nothing.

Comparison to the Porter Stemmer for English

The Farsi stemmer is similar to the Porter stemmer [5]. Each is based on the morphology of its language. Both stemmers match words with a set of suffixes, and use multiple phases to conform to the rules of suffix stacking. Furthermore, they enforce a lower bound on how much information a stem retains.

However, there are important differences. For example, the Porter stemmer counts substrings of consecutive consonants and vowels, estimating the information content, before deciding to remove a suffix. In Farsi, many spoken vowels are not written, so the stemmer cannot count them. Therefore, the Farsi stemmer uses stem length to define a lower bound on information content (in the current version, minimum stem length = 3). Another difference is that the Farsi stemmer identifies prefixes, while the Porter Stemmer does not.

The first step of the algorithm is to try to find a terminal substring of the input word that is equal to a Farsi suffix from the following list:

ترین ‘ superlative

هاست past perfect singular third person

هایم	past perfect plural first person
هاید	past perfect plural second person
هاند	past perfect plural third person
ترین	superlative
جات	plural
گان	plural
هام	past perfect singular first person
های	past perfect singular second person
کار	agent
گار	agent
مان	first person plural possessive
تان	second person plural possessive
شان	third person plural possessive
اش	third person singular possessive
نده	agent
ار	verbal noun
یم	plural first person
ید	plural second person
ند	plural third person
ها	plural
ان	plural
ات	plural
تر	comparitive
ثر	comparitive
ت	singular second person
ش	third person singular possessive
ا	verbal noun

ه	object noun
م	singular first person
د	singular third person
ی	singular second person
گر	agent

If multiple suffixes are found, the stemmer chooses the longest suffix that would leave a sufficiently long stem. Consider the Farsi word دستشان ("their hands"). Both the plural suffix ان , and the plural possessive شان match the end of the word. Removing ان leaves four letters, and removing شان leaves three letters. Since both leave sufficiently long stems, the stemmer removes شان , the largest, producing دست (hand).

Suffix Classes

Each suffix belongs to a suffix class. The procedure for stemming the word is determined by the class to which its identified suffix belongs. The suffix classes are *verb, possessive, plural, other nouns*, and *other suffixes.*

Nouns

Recall that nouns may have stacked suffixes, according to the pattern:

{non-plural/non-possessive}{plural}{possessive}<stem>

Because each word has as many as three suffixes, the algorithm uses as many as three phases. If a possessive suffix is found in the first phase, it will go through a second phase. If, in the second phase, the stemmer finds a plural suffix, the word undergoes a third phase.

If the first identified suffix indicates a possessive noun, the stemmer removes the suffix then examines the new end of the word for a non-possessive noun suffix; if it finds one, it

applies the rules for that suffix. This makes sense because if the suffix is chosen correctly, the word is a noun, and, if we have removed a possessive suffix, another possessive suffix would not be valid. On the other hand, plural and other nominal suffixes may remain after removing a possessive suffix.

If the chosen suffix indicates a plural noun, the stemmer removes the suffix then examines the new end of the word for a non-possessive non-plural noun. If it finds one, it applies the rules for that suffix. This works because in a noun that has a plural suffix and a possessive suffix, the possessive suffix follows the plural suffix. Moreover, in a noun that has a plural suffix and a non-plural non-possessive suffix, the plural suffix follows the non-plural non-possessive suffix.

When a stemmer finds a non-possessive, non-plural, noun suffix, it removes the suffix and outputs the result. It does not start a new phase.

Verbs

If the identified suffix indicates a verb, the stemmer removes the suffix. Then it examines the front of the word for the common verbal prefixes ن ،ب, andمی. A removed prefix must conform to Farsi morphology. ب is not prefixed to any word that is also prefixed by ن or می. If ن is found in conjunction with می , ن precedes می. Furthermore, to add ن or ب to the beginning of a word that starts with a vowel, an ی is infixed between the prefix and the stem. The stemmer identifies and removes the longest valid prefix combination that leaves the minimum stem length.

Other Suffixes

For suffixes not classified above, the Farsi stemmer removes the suffix and output the result. It starts no additional phases.

Exceptions

Some suffixes require treatment that does not conform to the preceding general description.

When the stemmer finds the suffix تان preceded by س it ignores the suffix. The Farsi suffix ستان – "stan" – meaning "location of", is often used for countries and regions, e.g. "Kurdistan". The stemmer does not modify these words.

The stemmer finds verbal suffixes د and ت but does not remove them. Removing them causes much overstemming and complicates the process of finding stopwords.

For several suffixes that beging with ا , if the word to which they are attached ends with a ا or و , a ی is infixed between the root and the suffix. After removing these suffixes, the stemmer looks for a terminal ی preceded by a ا or و ; if it finds this pattern, it removes the terminal ی .

Imperfections

The stemmer does not remove suffixes without error. For some words, the stemmer removes more of the word than it should, because it removes every terminal substring that matches a valid Farsi sufffix. The stemmer can't be certain that the terminal substring is not part of the root. However, removing part of the root will not cause overconflation if it retains enough information to distinguish it from unrelated words.

On the other hand, the stemmer may fail to remove terminal substrings that are valid Farsi suffixes. There are three reasons for this. First, there are Farsi suffixes that are not in the list. Second, the verbal suffixes د and ت , though recognized, are purposefully not removed. Third, the stemmer will not produce a stem less than three characters long. If the shortest matching suffix leaves a root with less than three characters, the stemmer outputs the entire word. These might cause underconflation, but the worst underconflation is like using no stemmer.

CHAPTER 6

IMPLEMENTATION

The Farsi stemmer is implemented in the C programming language. Internally, *long int* variables represent the unicode values for Farsi characters.

Finding Suffixes

We want to find the longest suffix that matches a terminal suffix while retaining the minimum stem length. The simplest approach is to keep a list of valid suffixes, and for each input word, go through the list, comparing each suffix to the end of the word. Each time you find a matching suffix you compare the length of that suffix to the longest suffix previously matched; if the current suffix is larger, and would leave the minumum stem length (MIN), it becomes the new leading candidate. Repeat this until you reach the end of the list. In psuedocode:

```
for(each suffix in list)
   if(suffix matches end of word
     AND top_candidate.length < suffix.length
     AND word.length - suffix.length >= MIN)
        top_candidate = suffix;
return top_candidate;
```

The time complexity of this algorithm is $\mathcal{O}(\sum_{s \in \{suffixes\}} |s|)$.

If the suffix list is known to be in descending order of length, we can save some time by choosing the first valid match in the list. In psuedocode:

```
for(each suffix in list)
  if(suffix matches end of word
    AND word.length - suffix.length >= MIN)
      return suffix;
```

The time complexity is the same.

The Farsi stemmer uses a faster procedure than those above. It employs a finite state machine that accepts all Farsi words with valid suffixes. The final state specifies which suffix, if any, matches. The time complexity (worst, best, and average) of this approach is $\mathcal{O}(|w|)$ where w is the input word. The machine has 170 states, 82 of which are accepting states.

To identify suffixes, it is natural to design a state machine that reads input words last character first, and proceeds toward the first character as it applies the state-transition function. If the machine halts in an accepting state, we know that a suffix matches the end of the word, and that the input word is long enough to retain the minimum stem length after removing that suffix. A forward reading machine might or might not be preferable.

The first step in building such a machine is to draw a directed tree, with a root node that represents the starting state, and Farsi characters labeling each arch, such that the edges of each path, from the root node to a leaf, spell out a Farsi suffix.

Then, it gets a bit complicated, for which there are two reasons. First, because the length of any stem must be no less than three, every path to an accepting state must have at least three steps plus a step for each character in the suffix. To solve this problem, once a suffix is identified, the finite state machine must traverse three dummy states before reaching a final accepting state.

The second complication results because we want the *longest* matching suffix. If a terminal substring of one suffix is equal to another, shorter, suffix, then the machine may, as it traces the longer suffix, fail to make a match, when it could match the shorter suffix. There are two reasons we might have to revert to the shorter suffix. First, the word might be too short to remove the longer suffix. Second, once we have traveled past the node that represents the shorter suffix, a character of the word may fail to match the character of the longer suffix.

Consider this example: suppose the end of an input word matches a two-character suffix, SUF_2, and that the third-from-last character is consistent with a four-character suffix, SUF_4. Now, if the fourth-from-last character is not consistent with SUF_4, we must reject SUF_4, and go to the correct dummy state of SUF_2. In other words, when we abandon the hypothesis that the word has suffix SUF_4, we have to reconsider the hypothesis that it has suffix SUF_2. Likewise, when we have identified a longer suffix, and find that the remaining stem is too short, we must change to the path of the next shortest matching suffix.

The finite state machine is represented by MACHINE, a two-dimensional array of unsigned integers. Each column represents a character from the alphabet, and each row represents a state. If the current state is s, and the next character in the word is c, then MACHINE[s][c] specifies the next state. Therefore, the algorithm for finding a suffix is:

```
i = word size;
while(i > 0)
  state = MACHINE[state][characterAt(word, i)];
  i = i - 1;
endwhile
return state;
```

Where characterAt(word, n) is the nth letter of the word.

Once the finite state machine has processed the word, we use the number of the final state to determine a suffix group. Each suffix group is defined by the way words are processed after the suffix is found. If suffixes S_1 and S_2 belong to the same group, the stemmer modifies them the same. To determine the suffix group, there is an array, suffixPostFunnel, the length of which equals the number of states in the machine. If the final state of the machine, when processing word w, is s, then the suffix group to which w belongs is found at suffixPostFunnel[s]. If no suffix is found, and the ending state is s_x, then suffixPostFunnel[s_x] = 0 and the stemmer is done.

Stacked Nominal Suffixes

If the final state indicates that we have a possessive nominal suffix, we remove the suffix and feed the modified word to the suffix finding state machine. If the subsequent final state indicates a non-possessive nominal suffix, we remove it. If the final state indicates that we have a plural suffix, we remove the suffix and feed the modified word to the suffix finding state machine. If the subsequent final state indicates a non-possessive non-plural nominal suffix, we remove it.

```
subroutine process noun suffix:
  if(input suffix is possessive)
   remove suffix;
   if( non-possessive suffix now matches)
    process noun suffix;
  endif

  if(input suffix is plural)
   remove suffix;
```

```
    if(if non-possessive non-plural suffix now matches)
     process noun suffix;
    endif

   if(input suffix is non-possessive and non-plural)
    remove suffix;
   endif

end subroutine
```

Finding Verbal Prefixes

If a word is found to have a verbal suffix, the stemmer looks for prefixes. The procedure for finding prefixes is the same as that for finding suffixes, except that the prefix state machine reads the words from the first character to the last. The string remaining after prefix removal must have at least three characters. The stemmer finds any valid prefix combination, and the final state of the prefix machine indicates how much of the front of the word to remove.

```
i = 0;
while(i <= word size)
  state = PREFIX_MACHINE[state][characterAt(word, i)];
  i = i + 1;
endwhile
return state;
```

Removing *Yeh* if Necessary

Depending on the group to which it belongs, a suffix might have a ى (*yeh*) infixed between the root and the suffix.Rather than build this logic into the state machine, the stemmer uses a constant boolean array, RYIN[], to remember if the suffix group requires checking for an added ى (*yeh*) and uses a boolean condition to determine if the ى will be removed.

```
if(RYIN[suffix group])
  if((last letter is yeh) AND
     (2nd to last letter is alef or waw))
    {
     remove last character;
    }
  endif
endif
```

CHAPTER 7

STOPWORD IDENTIFICATION

To facilitate index-building and query processing, the Farsi information retrieval system removes stopwords. A stoplist was compiled using term frequency, common sense, English stopword lists, and automatic verb conjugation. However, due to frequent verbal affixation of verbal stopwords, we do not merely refer to a list for stopword identification. We use the Farsi stemmer in combination with the word list.

An initial list of stopwords was automatically compiled by identifying the five hundred most frequent words in the collection. This method was insufficient due to the narrow focus of our collection. For example, the term آیران (Iran) was among the fifty most frequent words in our mostly political collection. Though this may be a useful stopword for our test collection, it is probably not a good stopword for a general collection. Therefore, referring to well known English stop word lists and to common sense, we manually edited the result to remove words that, though frequent in our collection, should not be considered stop words in a general collection, to which we eventually want to apply the stopword list. However, an evaluation of this list revealed that it was incomplete.

Among the Farsi stopwords are 12 verbs, each with a past tense and imperative form. With all valid prefix and suffix combinations, verbs have as many as 91 variations. Though a given verbal root may occur frequently, most of its variations occur infrequently.Therefore, a large number of variations of stop words failed to make the frequency list, though each variation, like its root, is a poor search term.

However, we do not list all variations. Instead, we list the regular and irregular stems of each verb. To decide if a word is a stopword, we look for it in the list. If it isn't there, we stem the word and look again.

However, some verbal stop words are shorter than the minimum stem length, causing inconsistent stemming; there are several *de facto* stems per short verbal root. Therefore, we included in the stopword list every correct variation of the short verbal stop words. We did this with a mechanical verbal conjugator. More succinctly, if set F contains the five hundred most frequent words, set C contains those words that were frequent in our collection, but are not a general stopword, set $V3$ contains verbal roots with at least three letters, and set Jug contains all conjugations of verbal roots with less than three character, then our stop word set is: $F \cup V3 \cup Jug - C$.

The logic for identifying stopwords is, in psuedocode:

```
isStopWord( word )
{
  if(isInStopList(word)
    OR isInStopList(stem(word)))
     return true;
  else
     return false;
}
```

CHAPTER 8

TEST AND RESULTS

To evaluate the Farsi stemmer, we observed its effect on the recall and precision, using the Farsi information retrieval system, a fixed set of Farsi queries, and a fixed document collection.

Precision and Recall

Eleven point precision/recall is a standard measurement of how well an information retrieval system functions. It describes the relationship between the the number of relevant documents and the number of irrelevant documents in an output document set. Suppose we have a document collection D and that D_q, $D_q \subseteq D$, is the set of documents output by the search engine for a query q. Define R_q as the set of documents in D, $R_q \subseteq D$, that are relevant to q. Then the *precision* of D_q is

$$(|R_q \cap D_q|) \div |D_q|.$$

The *recall* of D_q is

$$(|R_q \cap D_q|) \div |R_q|$$

For our testing procedure, recall is truncated to one digit right of the decimal point.

Eleven point precision/recall is the average of the precision for eleven levels of recall: .0, .1, .2, . . . , 1.0. To determine precision for each level of recall, start with D_q empty and add documents one at a time, starting from highest ranked document and moving towards the least ranked document, computing recall and precision with each new document. For

each recall level, choose the highest corresponding precision. *Interpolated* precision at a given recall level is the highest precision achieved at that or any greater recall level. [7]

Test

To run the test, a collection of 1647 Farsi documents, primarily internet documents, was created. Native Farsi speakers compiled a list of sixty queries. For each each document in the collection, and for each query in the list, a native Farsi speaker determined whether the document was relevant to the query, resulting in a total of about 98,000 judgements. The Farsi collection was then indexed without using the stemmer, and without removing stopwords. We then processed each query in the list, using the vector space model of Salton [2] with the cosine measurement of Witten [7] as our similarity measurement. From each query's returned list, sorted by the cosine measurement, a precision/recall table was generated. Then the sixty tables were averaged into one precision/recall table.

This procedure was repeated using the stemmer and the stopword list. When processing the queries and building the index, all stop words were omitted and every non-stopword was stemmed. This produced new results including another average table.

Results

Test results suggest that the stemmer and stopword removal have a positive effect on information retrieval. The crucial numbers in tables 1 and 2 are the interpolated eleven point averages. The test in which the stemmer was used shows an increase in the interpolated eleven point average of .033, or 18%.

In addition to the data in tables 1 and 2, the test revealed that there were a few cases of overstemming. For example, the stemmer, mistaking the terminal substring ان for the plural suffix, removed it from ایران ("Iran"), yielding ایر. I do not know if this overstemming led to overconflation.

Table 8.1: Average precision/recall results using no stemmer and no stopword removal

recall	precision	interpolated
0	0.328	0.483
10	0.253	0.353
20	0.228	0.273
30	0.119	0.215
40	0.131	0.197
50	0.151	0.170
60	0.078	0.105
70	0.055	0.074
80	0.039	0.060
90	0.027	0.049
100	0.046	0.046

eleven pt avg	int eleven pt avg	three pt avg	int three pt avg
0.132	0.184	0.139	0.167

Table 8.2: Average precision/recall results using stemmer and stopword removal

recall	precision	interpolated
0	0.342	0.544
10	0.310	0.413
20	0.290	0.333
30	0.142	0.242
40	0.137	0.210
50	0.171	0.191
60	0.096	0.141
70	0.086	0.106
80	0.045	0.080
90	0.030	0.065
100	0.060	0.060

eleven pt avg	int eleven pt avg	three pt avg	int three pt avg
0.155	0.217	0.169	0.201

CHAPTER 9

CONCLUSION AND FURTHER RESEARCH

Conclusion

The results of the stemming test indicate that the Farsi stemmer improves retrieval. It will probably complement the Farsi search engine. This does not mean there is no room for improvement. Modifications that may improve the stemmer include editing the list of suffixes, changing the minimum stem length, and foregoing prefix removal.

Further Research

Further research will aim at understanding the effects of and improving the performance of the Farsi stemmmer.

A possible approach to understanding the effects of the stemmer is to determine whether overstemming is causing overconflation. If it is, there may be ways of diminishing it.

It may be worthwhile to seek a proper subset of the current suffix set that works better than the current suffix set. Due to the size of the current suffix set, the number of subsets is extremely large: $2^{36} \approx 65,000,000,000$, rendering exhaustive search unreasonable. To narrow the search for a good subset, one can rate specific suffixes by the performance of queries in which they are removed. Alternatively, one could run a precision/recall test for each suffix rule, and choose only those rules that individually improve performance.

Another possible modification is to increase the minimum stem length from three characters to four. This modification will decrease conflation, and, it is reasonable to expect, decrease overconflation. In the case of آیران, the three character limit does not prevent

stemming, while the four character limit does. However, we ran this test and found that setting a minimum stem length of four performed slightly worse setting than a minimum stem length of three.

To facilitate such development it may be beneficial to implement a more flexible source program to realize the algorithm. The DFA currently in use is handwritten and difficult to modify without extravagance. A source program for which the suffix set and minimum stem length are easily modified would be preferable.

BIBLIOGRAPHY

[1] William B. Frakes. Information Retrieval, Data Structures and Algorithms, chapter Stemming Algorithms. Prentice Hall PTR, 1992.

[2] A. Wang G. Salton and C. Yang. A vector space model for information retrieval. Journal of the American Society for Information Science, 1975.

[3] David A. Hull. Stemming algorithms - a case study for detailed evaluation. Technical report, Rank Xerox Research Centre, 1995.

[4] Ph.D Mohammad Sadeh. Stemming persian technologies. Technical report, Information Science research Institute, 2002.

[5] M.F. Porter. An algorithm for suffix stripping. Program, 14, 1980.

[6] Wheeler M. Thackston. An Introduction to Persian. Ibex Publishers, 1993.

[7] I. Witten, A. Moffat, and T. Bell. Managing Gigabytes: Compressing and indexing documents and i Morgan Kaufmann, 2nd edition, 1999.

VITA

Graduate College
University of Nevada, Las Vegas

Russell Beckley

Local Address :

2011 South Idaho Street
Pahrump, NV 89048

Degrees :

Bachelor of Arts, Economics, 1993
University of Montana, Missoula

Thesis Title :

The Design, Implementation, and Effectiveness of a Farsi Word Stemmer.

Thesis Examination Committee :

Chairperson, Kazem Taghva, Ph. D.
Committee Member, Dr. Thomas Nartker, Ph. D.
Committee Member, Dr. Ajoy K. Datta, Ph. D.
Graduate Faculty Representitive, Dr. Emma Regentova, Ph. D.