An experimental Tagalog Finite State Automata spellchecker with Levenshtein edit-distance feature

Joseph Marvin R. Imperial
College of Computing and Information Technologies
National University
Manila, Philippines
jrimperial@national-u.edu.ph

Czeritonnie Gail V. Ya-On, Jennifer C. Ureta
College of Computer Studies
De La Salle University
Manila, Philippines
czeritonnie_ya-on@dlsu.edu.ph,
jennifer_ureta@dlsu.edu.ph

***Abstract*— In this paper, we present an experimental development of a spell checker for the Tagalog language using a set of word list with 300 random root words and three inflected forms as training data and a two-layered architecture of combined Deterministic Finite Automaton (DFA) with Levenshtein edit-distance. A DFA is used to process strings to identify if it belongs to a certain language via the binary result of accept or reject. The Levenshtein edit-distance of two strings is the number (k) of deletions, alterations, insertions between two sequences of characters. From the sample trained wordlist, results show that a value of 1 for the edit-distance (k) can be effective in spelling Tagalog sentences. Any value greater than 1 can cause suggestion of words even if the spelling of words is correct due to selective and prominent usage of certain characters in the Tagalog language like a, n, g, t, s, l.**

Keywords— Deterministic Finite State automata, spell checker, Tagalog, Levenshtein edit-distance

I. Introduction

A spell checker is a tool used to identify a misspelled word based on a selected training set. A generated list of spelling suggestions is created upon detection of a misspelled word. Using Levenshtein edit-distance, the word with the minimum number of character pair transpositions is selected. The Tagalog language can have several possible inflections and morphological rules. It is not efficient to examine all the words in Tagalog to model training. Thus, encoding all the rules in the automaton is not practical. Rules are created by using training set to train the automaton.

The finite state machine (FSM) created is the input Filipino alphabet. Each character changes the state of the model. If the final character put the model is accepted by the final state, then the word belongs to the language. A Deterministic Finite Automaton

In this study, a Levenshtein automaton model, a DFA with Levenshtein-distance feature, was trained with a Tagalog wordlist consisting of 300 words and three of its inflected forms. Once trained, varying values of k-edit distance has been explored to identify the effectiveness of employing such model to a morphologically rich language.

II. Review of Related Literature

Filipino is a morphologically rich language whose word combination uses different verb affix such as prefix, infix, suffix, and syllable or word duplication. [1] To search a Filipino word in a dictionary, the keyword should consider the tense associated with the action word. The past (*pangnangdaan*), present (*pangasalukuyan*), and future (*panghinaharap*) tense form are treated different entries in a dictionary system. Say, the word "*punta*" which means "*go to*" is a different entry from the word "*pumunta*" meaning "*went to or move forward*". It can be space-wasting to create an individual entry for each inflection. Moreover, there is difficulty in maintaining such a system.

The existing Filipino spellcheck solution applied the entire spelling rule and guidelines, namely, Komisyon sa Wikang Filipino 2001 Revision of the Alphabet and Guidelines in Spelling Filipino Language (or KWF), and the Gabay sa Editing Wikang Filipino (or GABAY) rulebooks into the system. The spellchecker carried out its tasks through manual-formulated and learned rules. [6]

Development of spell checker using finite state machines was also explored and applied to Malayalam, a morphological-rich Dravidian language spoken in the Indian state of Kerala [7]. A finite state transition model was generated using a predefined training set of Malayalam root words and their inflected forms. The FSM was trained is such a way that it can recognize the root words and its inflections and if correctly spelled, the string word is accepted by the final state of the FSM.

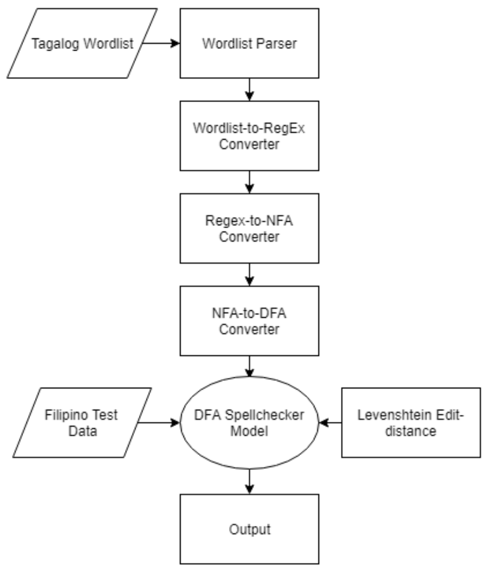


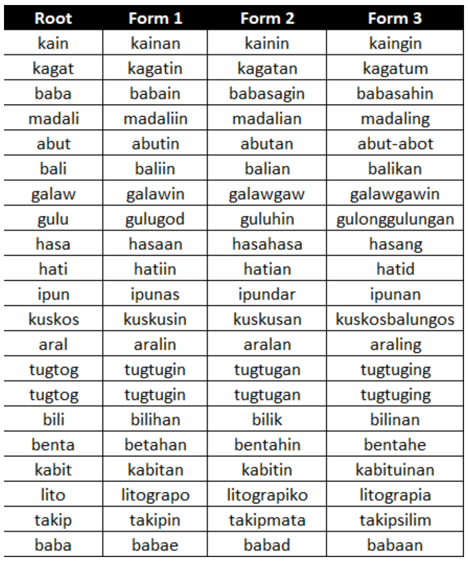
Fig 1. Methodology for Spellchecking using DFA with Levenshtein edit-distance.

III. Methodology

The research follows a step-by-step methodology as shown in Figure 1.

A. Word List Data

For the training data, a sample Tagalog wordlist was used containing 300 Tagalog root words with its inflected forms. Inflected forms are stemmed from the root words added with suffixes connoting various factors in the Tagalog language like repetition, action, and time. For the training set of wordlists, three inflected forms are used. However, additional forms can be added. Figure 2 shows a sample of Tagalog words list used for training. All root words have corresponding three or more inflected forms.



Root	Form 1	Form 2	Form 3
kain	kainan	kainin	kaingin
kagat	kagatin	kagatan	kagatum
baba	babain	babasagin	babasahin
madali	madaliin	madalian	madaling
abut	abutin	abutan	abut-abot
bali	baliin	balian	balikan
galaw	galawin	galawgaw	galawgawin
gulu	gulugod	guluhin	gulonggulungan
hasa	hasaan	hasahasa	hasang
hati	hatiin	hatian	hatid
ipun	ipunas	ipundar	ipunan
kuskos	kuskusin	kuskusan	kuskosbalungos
aral	aralin	aralan	araling
tugtog	tugtugin	tugtugan	tugtuging
tugtog	tugtugin	tugtugan	tugtuging
bili	bilihan	bilik	bilinan
benta	betahan	bentahin	bentahe
kabit	kabitan	kabitin	kabituinan
lito	litograpo	litograpiko	litograpia
takip	takipin	takipmata	takipsilim
baba	babae	babad	babaan

Fig 2. The sample Tagalog wordlist training data

B. Parsing Wordlist to RegEx

To make the wordlist understandable by a Deterministic Finite Automata, it is essential to convert it to a regular expression. Regular Expressions, or RegEx, describe regular languages. It can be used to determine if a given string or sequences of characters are accepted by a language [2]. Each set of the root word and inflected forms have their own RegEx group to be combined as one for a single pass to the RegEx-to-NFA converter.

C. RegEx to Nondeterministic Finite Automaton (εNFA)

The first phase in the conversion process is transforming the input RegEx string to an NFA with epsilon transitions. Using Thompson's Construction algorithm [3], each expression will be split into subexpressions. These subexpressions can be categorized into one of the following rules in Thompson's construction algorithm based on its operator. The algorithm for converting the RegEx to ε-NFA is described below.

For every character in the RegEx string, the converter identifies if it is a member of the predefined alphabet or not, or if it is one of the operators and closures accepted. Three separate Python lists were created functioning as stacks to keep track of the conversion. One is for storing the operators, the second contains the Thompson construction for every alphabet in the automata, and the last records all the alphabet in the RegEx.

For every operator (union or concatenation) or in the stack, two objects from the automata stack will be popped to be processed.

For every closure property, the last object in the automata stack will be popped and passed to one of the three available Thompson Construction function for conversion. Once processed, the converted forms will be appended back to the automata stack

D. Nondeterministic Finite Automata with Epsilon Transitions (ε-NFA) to Deterministic Finite Automata (DFA)

After processing every alphabet, operators, and closure from the RegEx string to an ε-NFA, it will now be converted into a Deterministic Finite Automaton (DFA).

The algorithm implemented by the proponents for the said conversion is as follows:

1. Iterate through every character of the language (previously saved from ε-NFA).
2. The ε-closure for the beginning states of NFA are taken as beginning states of DFA.
3. The states that can be traversed from the present to each symbol (union of transition value and their closures for each states of NFA present in the current state of DFA and including itself) are searched. If any new state is found, it is considered as current state.
4. The search is continued until there is no new state present in DFA transition table.
5. The states of DFA which contains final states of NFA are marked as final states of DFA.

The final output of the DFA will be responsible for first-tier spell checking of the sample Tagalog test sentences to identify it the words are recognized by the language.

E. The Levenshtein Edit-Distance Feature

The Levenshtein edit-distance is a metric for identifying the number of changes via character edits of insertions, deletions, and alterations of one source string from the other. [4] An example of Levenshtein edit-distance calculation is shown below.

1. *kainan* → *kainin* (edit-distance of 1 by substituting 'a' to 'i'
2. *bago* → *bagay* (edit-distance of 2 by substituting 'o' to 'a' and appending 'y')
3. *gamitin* → *gamit* (edit-distance of 2 by deleting the last two characters of the first string)

For the implementation of the Tagalog DFA spellchecker with Levenshtein automaton, the applicable edit-distance value was experimented using the values 1 and 2.

F. *Suggesting Possible Spellings*

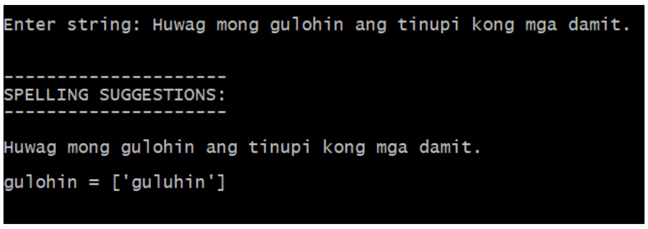
After the first tier and second tier spellchecking of the DFA and Levenshtein edit-distance feature, possible suggested spellings are printed out as an additional feature. Since the developed spell checker program recognizes misspelled words from the trained wordlist, it can also suggest words based on the wordlist based on the number of edit-distance. For Tagalog, an edit-distance of 2 and 1 was explored.

IV. Results and Discussion

The collected Tagalog wordlist containing 300 sets each with a single root word and three inflected forms were trained using the specified methodology. The total number of states generated for the DFA using the various converters was 226, with 79 final states. The DFA performs the first-tier spell checking by indicating if each word is accepted by the language. The second-tier spell checking is performed by the Levenshtein edit-distance feature of the DFA. Figures 3, 4, 5, and 6 below shows the checking and suggestive capabilities of the developed program using DFA and Levenshtein edit-distance using sample sentences.

Test Sentence A = "*Huwag mong gulohin and tinupi kong mga damit.*" (*Don't mess with my folded clothes*)

Test Sentence B = "*Pwede bang abotan mo ako ng tubig?*" (*Can you pass me some water?*)



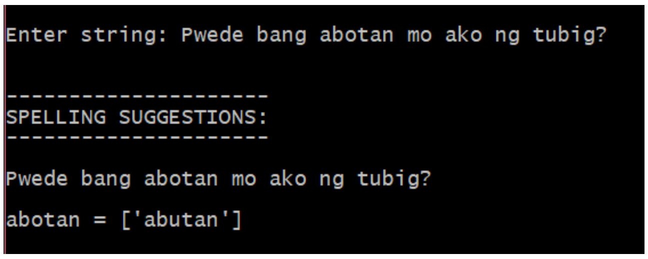
```
Enter string: Huwag mong gulohin ang tinupi kong mga damit.

---------------------
SPELLING SUGGESTIONS:
---------------------

Huwag mong gulohin ang tinupi kong mga damit.

gulohin = ['guluhin']
```

Fig 3. Spell checking Test Sentence A with edit-distance of 1



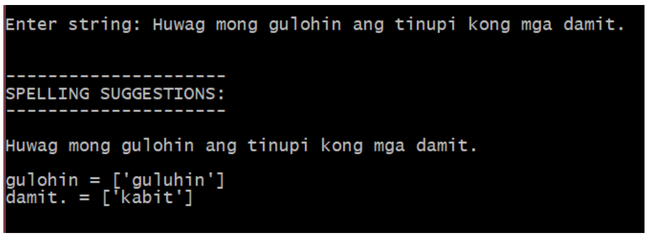
```
Enter string: Pwede bang abotan mo ako ng tubig?

---------------------
SPELLING SUGGESTIONS:
---------------------

Pwede bang abotan mo ako ng tubig?

abotan = ['abutan']
```

Fig 4. Spell checking Test Sentence B with edit-distance of 1



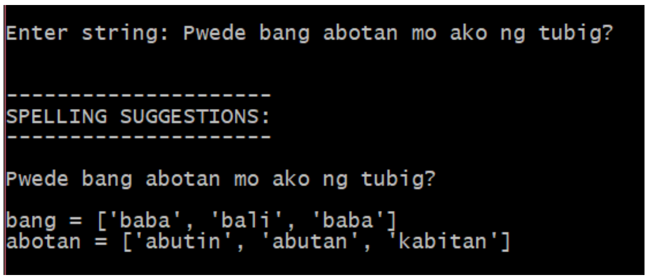
```
Enter string: Huwag mong gulohin ang tinupi kong mga damit.

---------------------
SPELLING SUGGESTIONS:
---------------------

Huwag mong gulohin ang tinupi kong mga damit.

gulohin = ['guluhin']
damit. = ['kabit']
```

Fig 5. Spell checking a Test Sentence A with edit-distance of 2



```
Enter string: Pwede bang abotan mo ako ng tubig?

---------------------
SPELLING SUGGESTIONS:
---------------------

Pwede bang abotan mo ako ng tubig?

bang = ['baba', 'bali', 'baba']
abotan = ['abutin', 'abutan', 'kabitan']
```

Fig 6. Spell checking a Test Sentence B with edit-distance of 2

From the results of spellchecking of sample sentences, the number of spelling suggestions may increase with the number of Levenshtein edit-distance used. With this, even words of correct spelling may be highlighted as incorrect and other words of unrelated meaning can be suggested. In view with this, the proponents inferred that the main cause of this is how alphabet is used in the Tagalog language. For the Tagalog language, the top 10 characters widely used are shown below. The list of most frequently occurring characters in the Tagalog language from a 2,151,963-character (368,905 words) document was curated by Stefan Trost of Word Creator [5].

TABLE I. Most frequently used characters in the Tagalog language

Character	Frequency
A	24.25%
N	11.77%
G	8.51%
I	7.89%
S	5.6%
T	4.87%
M	4.27%
O	4.19%
L	3.77%
K	3.61%

V. Results and Discussion

The collected Tagalog wordlist containing 300 sets each with a single root word and three inflected forms were trained using the specified methodology. The total number of states generated for the DFA using the various converters was 226, with 79 final states. The DFA performs the first-tier spell checking by indicating if each word is accepted by the language. The second-tier spell checking is performed by the Levenshtein edit-distance feature of the DFA. Figures 3, 4, 5, and 6 below shows the checking and suggestive capabilities of the developed program using DFA and Levenshtein edit-distance using sample sentences.

References

[1] Roxas, R.R, & Mula, G.T, “A morphological analyzer for Filipino verbs,” 22nd Pacific Asia Conference on Language, Information and Computation, 2008.

[2] Regular expression. (2018, December 11). Retrieved from https://en.wikipedia.org/wiki/Regular_expression

[3] Anon. 2017. Theory of Computation | Minimization of DFA. (May 2017). Retrieved October 23, 2018 from https://www.geeksforgeeks.org/program-implement-nfa-epsilon-move-dfa-conversion/

[4] Levenshtein, Vladimir I. (February 1966). "Binary codes capable of correcting deletions, insertions, and reversals". Soviet Physics Doklady. 10 (8): 707–710. Bibcode:1966SPhD...10..707L.

[5] Trost, S. (n.d.). WordCreator. Retrieved from https://www.sttmedia.com/characterfrequency-filipino

[6] C. Cheng, C. P. Alberto, I. A. Chan, and V. J. Querol, “SpellCheF: Spelling Checker and Corrector for Filipino,” Journal of Research in Science, Computing and Engineering, vol. 4, no. 3, 2008.

[7] N. Manohar, P. T. Lekshmipriya, V. Jayan, and V. K. Bhadran, “Spellchecker for Malayalam using finite state transition models,” 2015 IEEE Recent Advances in Intelligent Computational Systems (RAICS), 2015.