Algorithms for certain classes of Tamil Spelling correction

Authors: Muthiah Annamalai*, T. Shrinivasan
Corresponding author: ezhillang@gmail.com

1. Introduction

Tamil language has an agglutinative, diglossic, alpha-syllabary structure which provides a significant combinatorial explosion of morphological forms all of which are effectively used in Tamil prose, poetry from antiquity to the modern age in an unbroken chain of continuity.

However, for the language understanding, spelling correction purposes some of these present challenges as out-of-dictionary words. In this paper the authors propose algorithmic techniques to handle specific problems of conjoined-words (out-of-dictionary) e.g. செஞ்சுகளைநீதியானை = செஞ்சு + களைநீதியானை when parts are alone present in word-list in efficient way. Morphological structure of Tamil makes it necessary to depend on synthesis-analysis approach and dictionary lists will never be sufficient to truly capture the language.

1.1 State of the art

Many popular spell-checking applications have advanced state of the art in Tamil spelling correction but their methods are sometimes opaque. The work of Rajaraman(Vaani) [14] and Dr. Vasu Renganathan [13] remain popular in the interwebs, while Google has used hunspell/aspell variants with affix dictionary of Thamizha collaborative [15] to create spelling support of Tamil text in Google Docs product - without contributing back much. Whereas, the input method editor for Apple iOS and such from Murasu Anjal team is proprietary [16]. Our own work of solthiruthi is nascent and performs somewhat slowly due to overzealous searches, and not ready for production; however, that does not mean inferiority in class of corrections it is capable of - some of which is shared in this paper.

1.2 Overview

Authors propose some algorithms to identify and correct conjoined words; algorithms for Mayangoli letter transposition/substitution error correction; algorithms are proposed for correcting typographical errors originating from keyboard layouts. Authors also propose using machine-learning / deep-neural network based techniques to (a) identify a Tamil word sequence as valid word or mis- spelled word. Tamil misspelled letters can be collected using
crown-sourced format and used as training data for a Tamil based Deep Neural Network (DNN) for the purpose of classifying correctly spelled and misspelled words. Further, Tamil verb declension problem can also perhaps be resolved using the big-data AI/ML approaches.

We also present algorithmic techniques to optimize a spell-checker implementation, to detect the presence of Tamil unicode character in unicode-point quickly, and also algorithm to detect foreign-language words in Tami and substitute with equivalent (when available, or acceptable Tamil form). While some of these techniques are common, this work shows the algorithms in Tamil language context, and their novelities are elucidated in this work.

2 Edit Distance Search Algorithm

Basic spell checker algorithm for Tamil is laid out in [1] by team of Prof. Geetha, Dhanabalan and co-workers which includes verb declension, affix removal, morpheme extraction and then applying corrections to root word and then synthesizing it.

2.1 Norving Algorithm

A simpler algorithm for correcting Tamil words is to directly apply the Edit distance algorithm - popularly referred to as the Norving-algorithm [2,4]. This algorithm essentially computes for each letter in a word the possibility that the letter at the position in word could be,

1. Deleted,
2. Substituted - with alternate letter
3. Inserted - with alternate letter

and enumerates through all the alternate forms of input word at edit distance of 2

Clearly the presence of 247 unique Tamil letters (323 including Grantha letters) including the alpha-syllabary forms explodes the search space for a N-letter word can go as, \((247)^N\) or \((323)^N\) as you may choose to implement the algorithm for an edit-distance of upto N letters in word. Clearly not all letters in a word will be mis-spelled.

Generally the algorithm is limited to searching 2 or 3 edit distance. For an N letter word, we have then \(nC_2 \times 323 \times 323\) options or \(nC_3 \times 323 \times 323\) options respectively for the chosen edit-distance. Judiciously implementing this algorithm is key to having a data-driven spell-checker perhaps by tree-pruning techniques [11].

The implementation in Open-Tamil solthiruthi module is the following which generates suggestions which are filtered against a dictionary (typically represented as Trie data structure [8]. This edit-distance search occurs in spelling correction in Mayangoli correction, and Typographical error correction as well as described in the sections below.

```python
def norvig_suggestor(word: list, alphabets=tamil_letters, nedit=1, limit=float("inf")):
    # recursive method for edit distance > 1
    if nedit > 1:
        result = []
        for nAlternate in norvig_suggestor(word, alphabets, nedit-1, limit-len(result)):
            if len(result) > limit:
```
3. Mayangoli letter transposition/substitution error correction

As per the above algorithm, we correct only words not in dictionary. This is non-word error correcting algorithm - i.e. it corrects only words not in dictionary. This algorithm won’t correct word-sense disambiguation type errors, homonyms etc. - i.e. the in-dictionary word error which occurs out of place in text. Algorithm will iterate through the words in text and generate alternates using best effort for the wrong word and send them to user (replacing per user choice).

2.2 Driver Algorithm for Spell Checker

As per the above algorithm, we correct only words not in dictionary. This is non-word error correcting algorithm - i.e. it corrects only words not in dictionary. This algorithm won’t correct word-sense disambiguation type errors, homonyms etc. - i.e. the in-dictionary word error which occurs out of place in text. Algorithm will iterate through the words in text and generate alternates using best effort for the wrong word and send them to user (replacing per user choice).
class Mayangoli:
    varisai = [[u"", u"", u""], [u"", u"", u""], [u"", u"", u""], [u"", u"", u""], [u"", u"", u""], [u"", u"", u""]]

@staticmethod
def run(word, letters):
    obj = Mayangoli(word, letters)
    obj.find_letter_positions()
    if len(obj.matches_and_positions) == 0:
        return []
    obj.find_correspondents()
    obj.generate_word_alternates()
    return obj.alternates

def find_letter_positions(self):
    for idx, letter in enumerate(self.letters):
        p = tamil.utf8.splitMeiUyir(letter)
        if len(p) == 1:
            continue
        mei, uyir = p
        for r in range(0, len(Mayangoli.varisai)):
            for c in range(0, len(Mayangoli.varisai[r])):
                if mei == Mayangoli.varisai[r][c]:
                    self.matches_and_positions.append((idx, r, c))
    return len(self.matches_and_positions) > 0
def find_correspondents(self):
    for pos, r, c in self.matches_and_positions:

```bash
~/devel/open-tamil$ ./spell.sh -i
>> காணலாம் "எம்பாடதமானதலைஏற்கனெடுப்புல்லாம் என்பதை உளவைறைத்தழந்து மாறகைளளாக்கியுடன் கறந்து பலலாம்; பலாம் மாகைளாக்கின் பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகவைந்து பிரித்தலாகavar

```
src_letter = self.letters[pos]
_._src_uyir = tamil.utf8.splitMeiUyir(src_letter)
alt_letters = list()
for alternate_mei in Mayangoli.varisai[r]:
    alt_letters.append( tamil.utf8.joinMeiUyir(alternate_mei,src_uyir) )
self.pos_classes.append(alt_letters)
return True

def _generate_combinations(self):
    return itertools.product("self.pos_classes")
def generate_word_alternates(self):
    for position_sub in self._generate_combinations():
        alt_letters = copy.copy(self.letters)
        if _DEBUG: pprint.pprint(position_sub)
        idx =0
        for pos,r,c in self.matches_and_positions:
            alt_letters[pos] = position_sub[idx]
        idx += 1
        word_alt = u"".join(alt_letters)
        self.alternates.append(word_alt)
        return True

4. Algorithm for conjoined word recognition

4.1. Algorithm for conjoined word recognition

4.1.1. Algorithm for conjoined word recognition

4.1.2. Algorithm for conjoined word recognition

4.2. Algorithm for conjoined word recognition

4.2.1. Algorithm for conjoined word recognition

class OttruSplit:
    """ அைரையுடன் = கூறு + காகம் """
    def run(self,lexicon):
        self.generate_splits()
        return self.filter(lexicon)
    def generate_splits(self):
        """
        நேற்றியுள்ளது =
        [[['உ', 'காகம்'],
          ['அங்காய்', 'காகம்']]
         [[['அர்னத்', 'காகம்'],
          ['அப்பத்', 'பாது']]
        """
        L = len(self.letters)-1
        for idx,letter in enumerate(self.letters):
            if not( letter in tamil.utf8.grantha_uyirmei_letters):
                continue
            muthal = idx == 0 and u"" or u"".join(self.letters[0:idx])
```
infitt-2019, chennai, india.
6 / 10

meethi = idx == 1 and u'' or u''.join(self.letters[dx+1])
mei,uyir = tamil.utf8.splitMeiUyir(letter)
muthal = muthal + mei
meethi = uyir + meethi
self.results.append([muthal,meethi])
return len(self.results) > 0

def filter(self,lexicon):
    self.results = list(filter(lambda x: all(map(lexicon.isWord,x)),self.results))
    return self.results
```

5. Typographical error correction in Tamil

அலசக அவண்டி தமிழ்99 வைணாம் அவண்டி கிளைச் செய்யும் வழியில் முதல் அணுவு செய்யும் செயலாக்கத்தில் அகத்தைத் தமிழ்மொழி அறிக்கையில் வரவு செய்துள்ளன. தமிழ் மொழியில் செய்யும் வழியில் கொண்டான். தமிழ் மொழியில் செய்யும் வழியில் செய்துள்ளார் [10].

Figure: (left) confusion matrix for Tamil-99 keyboard, (right) Keyboard layout - Tamil-99.
# explore all edit distances - i.e. len(word_in) or only upto value in ed.
# we can restrict the edit distance search to any value from [1-N]
def oridam_generate_patterns(word_in,cm,ed=1,level=0,pos=0,candidates=None):
    """ ed = 1 by default, pos - internal variable for algorithm'' ""
    alternates = cm.get(word_in[pos],[])
    if not candidates:
        candidates = []
    assert ed <= len(word_in), 'edit distance has to be comparable to word size [ins/del not explored]'  
    if (pos > len(word_in)) or ed == 0:
        return candidates
    pfx = ''
    sfx = ''
    curr_candidates = []
    for p in range(0,pos):
        pfx = pfx + word_in[p]
    for p in range(pos+1,len(word_in)):
        sfx = sfx + word_in[p]
    for alt in alternates:
        word_alt = pfx + alt + sfx
        if not (word_alt in candidates):
            candidates.append( word_alt )
            curr_candidates.append( word_alt )
    for n_pos in range(pos,len(word_in)):
        # already what we have ' candidates ' of this round are edit-distance 1
        for word in curr_candidates:
            oridam_generate_patterns(word,cm,ed=1,level+1,n_pos,candidates)
    if level == 0:
        #candidates.append(word_in)
        for n_pos in range(pos,len(word_in)):
            oridam_generate_patterns(word_in,cm,ed, level+1,n_pos,candidates)
    return candidates

def corrections(word_in,dictionary,keyboard_cm,ed=2):
    """
    @input: word_in - input word
dictionary - dictionary/lexicon
keyboard_cm - confusion matrix for keyboard in question
    """
    assert isinstance(dictionary,Dictionary)
6. AI/ML approaches to Tamil spelling correction

Wide-spread success of various tasks like image recognition [6] which surpasses human level cognition in this task, and significant improvements in speech synthesis and speech recognition have demonstrated A.I. and Machine Learning methods suitable for these tasks, Sequence to sequence models including Recurrent Neural Networks (RNN), Long Short-term Memory (LSTM), and Word2Vec type representations of Neural Networks have enabled high success rate in translation, concordance tasks [7].

6.1 Anomaly Detection to Reduce Processing

Currently spelling correction is required in less than 25% of user input which remains perhaps requiring deep analysis and complex algorithms. To improve speed of spelling correction we can also take the approach of training a AI/ML based system to declare a word or text as valid Tamil word - this can (in principle) automatically resolve 75% of input text and free up the time required for the complex analysis. Such heuristics can allow using anomaly detection algorithms for proper identification of the errors in the text and filter the computational load on the spell checking program.

Further, Tamil misspelled letters can be collected using crowd-sourced format and used as training data for a Tamil based Deep Neural Network (DNN) for the purpose of classifying correctly spelled and misspelled words. Further, Tamil verb declension problem (see: Rajam Krishnan) can also perhaps be resolved using the big-data AI/ML approaches.

6.2 Word Sense Disambiguation

1) அேப்பசிவ. 2) அேப்பசவ

Both the sentences contain legal words from a Tamil Lexicon but only one makes sense [12]. The simple minded non-word error correcting spell-checker will not be able to tell them apart. The simple way to tell them apart is to use a concordance database for Tamil and find the words co-occurring successors/predecessors and offer alternate. Yet another way is to find the word-level bi-gram, tri-gram probability distribution from a language model for Tamil and use it to identify சவ is not a highly probable successor to அேப்பப thereby determining such an instance could trigger the suitable search and replacement.

6.3 Algorithm for Foreign-Language Word Substitution

In a previous research article we proposed a fully feed-forward ANN which was capable of identifying non-tamil words in text as well as English text (originally reported in [17]).
We propose a simple algorithm which can use the above AI classifier and extract this word and look up equivalent Tamil word for English or other language text using a parallel dictionary. Updating the word for the tense and any morphological processing we can replace the anglicized word in Tamil script with an equivalent Tamil word.

6.4 Computational Complexity

We try to answer the question of complexity of Tamil language spell-checker algorithm in various contexts. It seems the correction of Tamil words in text will be somewhere between exponential in number of letters of word - clearly the substitution cases are shown to be exponentially complex in number of letters.

The complexity of affix removal is polynomial in the number of affixes.

7 Optimizing a Spell Checker Implementation

We can propose techniques for how to improve speed of spell checker.

7.1 Algorithm for fast Unicode letter detection

Text containing a mix of Tamil and English (or other language scripts) can be quickly eliminated by using the check if a character falls into the Tamil Unicode block in basic block range. This check eliminates a character in analyzed text from further complex processing. In Python3, the following function performs the Unicode Tamil letter/character detection.

\[
is\textunderscore tamil\textunderscore unicode\textunderscore predicate = \lambda x: x >= \text{chr}(2946) \text{ and } x <= \text{chr}(3066)\]

7.2 Performance Engineering

7.2.1 Caching results

A given text for spelling correction can be grouped into a series of words, excluding stop words. Now we can make spell checker save suggestion list for each mis-spelled word (non-word) and re-use the suggestions list from cache the second and later times when word is mis-spelled identically in the document. The same mis-spelling can, however, have different correct words in the document at each site depending on context and only be replaced up to first order approximation in “white-washed” fashion. Caching has a potential to restore performance of spell checker.

7.2.2 Multi-Threading/Multi-Processing

The non-word errors suggestion generation and morphological processing can be carried out in parallel in the whole document, perhaps by a producer-consumer model serviced by a dozen(s) of worker threads which all generate suggestions for non-word errors including caching as mentioned above. Such an architecture of spelling checker could improve speed.
7.2.3 Redis / distributed DB

The memory performance of the spelling checker can be offloaded to a remote machine by requesting a distributed data-based to hold the dictionary or trie form of the word-list/Lexicon. This may be suitable for production environments.

8 Conclusion

In this paper we have attempted to make a summary of various known algorithms for specific classes of Tamil spelling errors. We believe this collection of suggestions to improve future spelling checkers. We also note do not cover many important techniques like affix removal and other such techniques of key importance in rule-based spell checkers.

References

1. “Tamil spell checker,” T. Dhanabalan, R Parthasarathi – Sixth Tamil Internet 2003-இல் இலங்கா பள்ளிவாசல் பொறியியல் கொள்கிறது, சங்கவாரம்.
2. J. L. Peterson, "Computer programs for detecting and correcting spelling errors," CACM (1980).
3. செயற்கையியல் – செயற்கையியல் 1, https://ezhillang.blog/2019/02/08/செயற்கையியல்-செயற்கையியல்-1
4. மாண்டும் விளக்கங்கள் விளக்கங்கள் https://ezhillang.blog/2017/11/04/மாண்டு-விளக்கங்கள்-மாண்டுவிளக்கங்கள்/
5. Peter Norvig, "How to write a spelling corrector," https://norvig.com/spell-correct.html
6. L. Fei-Fei, ImageNet: crowdsourcing, benchmarking & other cool things, CMU VASC Seminar, March, 2010.
7. Ian Goodfellow and Yoshua Bengio and Aaron Courville, 'Deep Learning,' MIT Press (2016).
8. Trie Data Structure, https://xlinux.nist.gov/dads/HTML/trie.html and https://en.wikipedia.org/wiki/Trie (accessed Aug 2019).
9. Mayangoli Sorkal List from Valaitamil.com http://www.valaitamil.com/list-of-mayankoli-sorkal_15177.html (accessed Jan 2019)
10. Typographical error correction for Tamil-99 keyboard https://ezhillang.blog/2019/02/18/செயற்கையியல்-செயற்கையியல்-3/ (2019).
11. Peter Norvig, ‘Artificial Intelligence - a modern approach,’ 3rd ed. (2009).
12. செயற்கையியல் செயற்கையியல் செயற்கையியல், https://ezhillang.blog/2018/04/07/செயற்கையியல்-செயற்கையியல்/
13. Tamil Spell Checker by Dr. Vasu Rengananthan, http://thetamillanguage.com/spellcheck/
14. Vaani Tamil Spell Checker by Rajaraman, http://thetamillanguage.com/spellcheck/
15. Hunspell Affix dictionary by V. Ilanchezhian, Mugunth, et-al Thamizha Collaborative, https://github.com/thamizha/thamizha-solthiruthi
16. Muthu Nedumaran and Murasu Anjal-team, iOS/Mac OS (High Sierra/Mojave) support for Tamil input editing via Tamil 99 (Anjal) keyboard, and Tamil phonetic keyboard (accessed 2019).
17. Syed Abuthahir et-al, “Growth and evolution of open-tamil,” INFITT, Coimbatore, India (2018).