Arabic Word Generation and Modelling for Spell Checking

Khaled Shaalan▫,Younes Samih‡, Mohammed Attia†▫,
Pavel Pecina◊, and Josef van Genabith†
▫The British University in Dubai, UAE
khaled.shaalan@buid.ac.ae
‡Heinrich-Heine-Universität, Germany
samih@phil.uni-duesseldorf.de
◊ Institute of Formal and Applied Linguistics,
Faculty of Mathematics and Physics,
Charles University in Prague, Czech Republic
pecina@ufal.mff.cuni.cz
†School of Computing, Dublin City University, Ireland
{mattia,josef}@computing.dcu.ie

Abstract

Arabic is a language known for its rich and complex morphology. Although many research projects have focused on the problem of Arabic morphological analysis using different techniques and approaches, very few have addressed the issue of generation of fully inflected words for the purpose of text authoring. Available open-source spell checking resources for Arabic are too small and inadequate. Ayaspell, for example, the official resource used with OpenOffice applications, contains only 300,000 fully inflected words. We try to bridge this critical gap by creating an adequate, open-source and large-coverage word list for Arabic containing 9,000,000 fully inflected surface words. Furthermore, from a large list of valid forms and invalid forms we create a character-based tri-gram language model to approximate knowledge about permissible character clusters in Arabic, creating a novel method for detecting spelling errors. Testing of this language model gives a precision of 98.2% at a recall of 100%. We take our research a step further by creating a context-independent spelling correction tool using a finite-state automaton that measures the edit distance between input words and candidate corrections, the Noisy Channel Model, and knowledge-based rules. Our system performs significantly better than Hunspell in choosing the best solution, but it is still below the MS Spell Checker.

Keywords: Arabic, spelling error detection and correction, finite state morphological generation, character-based language model, Arabic open-source spell checker

1. Introduction

With the advent of the era of free and open-source software, the need for language resources has become even stronger. Very few researchers and developers have tried to develop a free alternative to the proprietary Microsoft Arabic spell checker. One example is the Arabic Spell[1] open-source project (designed for Aspell), which relies on the Buckwalter morphological analyser and generates about 900,000 inflected words. Another example is the Ayaspell[2] word list, which is the official resource used in OpenOffice applications. Developers of this word list created their own morphological generator, and their word list contains about 300,000 inflected words. In this paper we use the term 'word' to designate fully inflected surface word forms, while the term 'lemma' is used to indicate the uninflected base form of the word without affixes or clitics.

We create a very large word list for Arabic using AraComLex[3], an open-source finite-state large-scale morphological transducer (as presented in Section 2), which generates about 13 million words, of which 9 million are found to be valid forms. For the sake of comparison, we also use a word list created from a corpus of 1,000,000,000 words. We automatically match all the word lists against the Microsoft Spell Checker (in the Office 2010 suite) to determine how many words are accepted and how many are rejected. We note her that at this stage, the acceptance and rejection decisions by MS Spell Checker are taken *per se* without any further checking. The results are shown in Table 1.

	No. of Words	MS Accepted	MS Rejected
AraComLex[4]	12,951,042	8,783,858	4,167,186
Arabic-Spell for Aspell (using Buckwalter)	938,977	673,875	265,103
1 billion-word corpus (Gigaword[5] and Al-Jazeera)	2,662,780	1,202,481	1460,447
Ayaspell for Hunspell	292,464	230,506	61,958
Total*	15,147,199	**9,306,138**	5,841,061

Table 1: Arabic word lists matched against Microsoft Spell Checker
* Totals will not add due to the removal of duplicates

[1] http://sourceforge.net/projects/arabic-spell/files/arabic-spell
[2] http://ayaspell.sourceforge.net/
[3] http://aracomlex.sourceforge.net/
[4] http://arabic-wordlist.sourceforge.net/
[5] http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC2009T3

Proclitics		Prefix	Stem	Suffix	Enclitic
Conjunction/ question article	Complementizer	Tense/mood – number/gender	Verb	Tense/mood – number/gender	Object pronoun
Conjunctions و wa ‘and’ or ف fa ‘then’	ل li ‘to’	Imperfective tense (5)	Stem	Imperfective tense (10)	First person (2)
Question word أ ’a ‘is it true that’	س sa ‘will’	Perfective tense (1)		Perfective tense (12)	Second person (5)
	ل la ‘then’	Imperative (2)		Imperative (5)	Third person (5)

Table 2. Possible concatenations in Arabic verbs

By comparing our list to other languages, we find that for English there are, among other word lists, AGID[6], which contains 281,921 words, and SCOWL [7], containing 708,125; for French, there is a word list that contains 338,989 words. The largest word list we find on the web is a Polish word list for Aspell containing 3,024,852[8]. This makes our word list one of the largest for any human language so far. Finnish and Turkish are agglutinative languages with rich morphology that can lead to an exposition in the number of words, similar to Arabic, but word lists for these two languages are not available to us yet. We take the combined and filtered list of 9,306,138 words as our spell checking word list and name it “AraComLex Extended”.

A complete and conclusive word list of Arabic is almost impossible to obtain due to the high morphological productivity of the language and the many constraints which even experienced linguists might not be able to explicitly formulate. Therefore, we use a character-based tri-gram language model trained on the total MS accepted list in Table 1 to detect permitted vs. disallowed clusters of characters. The result of our experiment (in Section 3.2) gives a precision of 97% at a recall of 100% for detecting valid and invalid word forms.

Having detected possible errors, the next step is to provide an ordered list of candidate corrections. We deploy the full list of correct Arabic words in a finite-state automaton to propose candidate corrections for misspelled words within a specified edit distance from the correct words. In order to rank the candidate corrections, a word-based language model (trained on Gigaword corpus data) is used to assign scores to each correction. We also use language specific knowledge to help choose the best correction in a given context.

The remainder of this paper is structured as follows. Section 2 illustrates the complexity of Arabic morphology and show how the word list is created from the AraComLex finite-state morphological generator. Section 3 shows how we use language modelling to predict valid words versus invalid words. Section 4 shows how finite-state edit distance (modified with knowledge-based rules) is used to provide spelling corrections for errors. Finally Section 5 gives the conclusion.

2. Finite-State Word Generation

Arabic morphology is well-known for being rich and complex (Beesley and Karttunen, 2003; Watson, 2002; Kiraz, 2001; Hajič et al., 2005). Arabic morphology has a multi-tiered structure and applies non-concatenative morphotactics. Words in Arabic are originally formed through the amalgamation of roots and patterns, as shown in Table 3. A root is a sequence of three consonants and the pattern is a template of vowels with slots into which the consonants of the root are inserted. This process of insertion is called interdigitation (Beesley and Karttunen, 2003). The resulting lemmas then pass through a series of affixations (to express morpho-syntactic features) and clitic attachments (as conjunctions and prepositions, for example, are mostly joined to adjacent words in writing) until they finally appear as surface forms.

Root	درس drs			
POS	V	V	N	N
Pattern	$R_1aR_2aR_3a$	$R_1aR_2R_2aR_3a$	$R_1AR_2iR_3$	$muR_1aR_2{\sim}iR_3$
lemma	darasa ‘study’	darrasa ‘teach’	dAris ‘student’	mudar∼is ‘teacher’

Table 3. Root and Pattern Interdigitation

Due to the richness and complexity of Arabic morphology, there is no corpus, no matter how large, that contains all possible word forms. Given a word in Arabic, one can change its form by adding or removing yet another prefix, suffix, proclitic or enclitic. This is why a morphological generator is essential in creating an adequate list of possible words.

As an example, possible concatenations and inflections in Arabic verbs are shown in Table 2. All affixes and clitics are optional, and they can be connected together in a series of concatenations. The maximum number of concatenations (of bound morphemes representing affixes and clitics) with Arabic verb lemmas is five. Unconstrained concatenations for verbs can produce as many as 33,696 forms for a single verb lemma. In real

6 http://sourceforge.net/projects/wordlist/files/AGID/Rev%204/agid-4.zip/download
7 http://sourceforge.net/projects/wordlist/files/SCOWL/Rev%207.1/scowl-7.1.zip/download
8 http://www.winedt.org/Dict/

constrained examples, some verbs, such as شكر *šakara* 'to thank', generate 2,552 valid forms. This considerable amount of form variation is a good indication of the productive power of Arabic morphology.

Attia et al. (2011) build a large-scale open-source finite-state morphological transducer for Arabic, AraComLex, that contains 30,587 lemmas. There are a number of advantages of this finite-state technology that makes it especially attractive in dealing with human language morphologies (Wintner, 2008). Among these advantages are the ability to handle concatenative and non-concatenative morphotactics, compactness, high speed, and efficiency.

	No. of Words	Coverage	Accuracy	f-measure
AraComLex	12,951,042	0.9697	0.9803	0.9899
Arabic-Spell for Aspell	938,977	0.6553	0.6586	0.7917
1 billion-word corpus	2,662,780	0.9982	0.9845	0.9921
Ayaspell for Hunspell	292,464	0.5529	0.5681	0.7190
AraComLex Extended	9,306,138	0.9840	0.9998	0.9999

Table 4: Comparison of coverage, accuracy and f-measure of AraComLex Extended against other resources

Another main advantage of finite-state morphology is that it is bidirectional, which makes it suitable for both analysis and generation. We use this generation feature to produce all the possible words within this morphology. AraComLex generates 12,951,042 words. When automatically validated using MS Spell Checker 4,167,186 forms are rejected, leaving 8,783,858 as accepted forms, that is 68% of the data. At the current stage, we just accept MS decisions without questions, yet more validation is needed to check how accurate the MS Spell Checker is in accepting and rejecting forms. The word list that we have created will be available as an open-source resource under GPLv3 license.

In order to test the the various components in this research project, we create a test set of 400,000 words collected from documents from Arabic news websites, and we make sure that these documents are not included in the Gigaword corpus. Words in the test set are automatically marked by MS Spell Checker as either 'correct' or 'incorrect'. We test the coverage, accuracy, and f-measure of AraComLex Extended against this test set and compare it to the other available resources, as shown in Table 4. AraComLex Extended has the best scores in accuracy and f-measure, and second best in coverage after the corpus.

3. Error Detection

We use two methods, the direct method, that is matching against a word list, and a language modelling method in case such a word list is not available.

3.1 Direct Detection

The direct way for detecting spelling errors is to match words in an input text against a list of correct words. Such a word list in Arabic can run into several millions as shown earlier. This is why it is more efficient to use finite state automata to store words in a more compact manner. Input string can then be composed against the valid word list paths and spelling errors will merely be the difference between the two word lists (Hassan et al., 2008).

3.2 Detection through language modelling

Language modelling has been used frequently for the purpose of spelling correction (Magdy and Darwish, 2006; Brill, Eric and Moore, 2000; Choudhury et al. 2007). However, here we build a language model in order to help in the validation and classification of Arabic words either in the existing word list or in new forms that can be encountered at future stages. Arabic is challenging for language modelling due to the high graphemic similarity of Arabic words. This is shown by Ben Othmane Zribi et al. (2003) who conducted an experiment using four editing operations (addition, substitution, deletion and interchanging two adjacent letters), calculating the number of correct forms among the number of automatically built forms (or lexically neighbouring words) resulting from these editing operations. They found that the average number of neighbouring forms for Arabic is 26.5, which is significantly higher than that for French, 3.5 and English, 3.0.

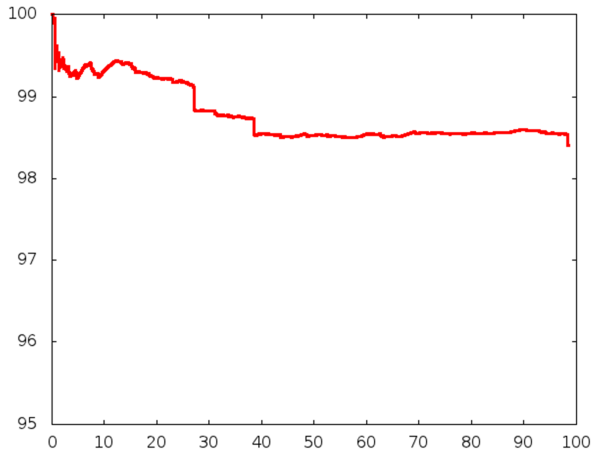

Figure 1: New Results of the LM classifier identifying valid and invalid Arabic word forms

We build a character-based tri-gram language model using SRILM[9] in order to classify generated words as valid and invalid. We split each word into characters, and create two language models: one for the total list of words accepted as valid (9,306,138 words), and one for the total list rejected as invalid (5,841,061 words) through MS Spell Checker filtering as shown in Table 1 above. The maximum word length attested in the data is found to be

[9] http://www.speech.sri.com/projects/srilm/

19 characters. We test the model against our test set of 400,000 words. Figure 1 shows a precision-recall curve of the classifier. The curve represents precision and recall scores of the detection of spelling errors based on the difference between the perplexity obtained by the accept model and the perplexity of the reject model. The downward movement of the curve indicates that the model is working quite reasonably giving a precision of 98.2% at a recall of 100%. The model also achieves a precision of around 98,8% at a recall of 40%. We can identify 30% of all errors with a precision of 99%, i.e. with 1% false alarms only.

4. Spelling Correction

Having detected an error in spelling, the next step is to propose a list of corrections. We deploy finite-state automata to propose candidate corrections within a specified edit distance measured by Levenshtein Distance (Levenshtein, 1966) from the misspelled word (Oflazer, 1996). Our implementation covers the edit distance primitive operations of substitution, deletion and insertion, in addition to the non-primitive operation of transposition (ab → ba). After choosing candidate corrections, we use the Noisy Channel Model (Section 4.2.2) trained on our corpus of one-billion words, and knowledge-based rules to assign scores to the candidate corrections and choose the the best correction independent of the context. In Section 4.2, we explain the ranking procedures and compare the results against the Microsoft Spell Checker in Office 2010 and Hunsell-ar (using Ayaspell) used in OpenOffice.

4.1 Related work

Hassan et al. (2008) applied an edit distance spelling correction approach to an Arabic dictionary of 526,492 entries and tested it on words of lengths from 3 to 13 characters. Here we apply a similar approach to the total word list of 9.3 million words, and we also apply it to all allowable word lengths from 2 to 19 characters. We complement our approach by using knowledge-based rules focused on the most common types of errors. For example, Arabic writers are frequently confused about the placement of *hamzah*, which can have 8 different forms (أ – إ – ا – ؤ – آ – ئ – ى – ـئـ - ء). *Alif maqsoura* and *yaa* (ى - ي) are often confused at the end of words, as one of them has two dots beneath and the other does not; the same with *taa marboutah* and *haa* (ة - ـه), where one of them has two dots above and the other does not.

Hassan et al. (2008) reported 89% auto-correction (or first order choice) accuracy, but they conducted the test on only 556 words. Moreover, they did not indicate the source of these errors and whether they are actual errors extracted from real texts or whether they were artificially introduced. In this paper we test our first order ranking (having the best correction first, also called 'auto-correction) approaches on 1,799 naturally-occurring (i.e., not machine-generated) spelling errors that are obtained from a news corpus of 400,000 words especially extracted for testing.

We tried using Foma (Hulden, 2009) to generate the finite-state automata, but although it worked nicely with smaller datasets, it failed to read the full list of 9.3 million words apparently due to its huge size. The Xerox XFST successfully built a finite-state network for the whole Arabic word list, and is used in the experiments reported below. The spelling-correction tool can be accessed on this temporary link[10].

4.2 Correction Procedure and Evaluation

Ranking is an important feature of a spelling checker. An optimized ranking algorithm would yield a reduced number of suggestions. Here we experiment with two methods of ranking spellchecking candidates. Our results (Section 4.2.3) show that augmenting edit distance and the Noisy Channel Model with knowledge-based rules improved the accuracy of ranking by 6.8% absolute.

Our first approach consists of assigning weights based on the edit distance between the misspelled word and a set of correction candidates. We train the Noisy Channel probability model on our one-billion word corpus (Gigaword and Al-Jazeera). This model is used to rescore candidates so that two candidates might receive different weights for the same number of edits. Our second approach differs from the first only in the edit distance scoring mechanism. Unlike the first edit distance, where costs assignment is based on arbitrary letter change, we augment the edit distance scoring mechanism with rules following the error patterns in the Arabic language (Shaalan et al. 2003), so that letters belonging to the same groups will be assigned lower edit costs.

Here we present a step-by-step explanation of the procedure we followed in spelling error correction.

1. All the misspelled words are extracted from a test corpus of 400,000 words. The number of misspelled tokens automatically identified by the MS Spell Checker is 6,279.
2. After removing duplicates from the 6,279 misspelled tokens, the number dropped to 3,012 types.
3. The 3,012 types were manually reviewed to create a gold standard correction for each misspelled word. Sometimes, the word was not actually misspelled, but was not known to the spell checker. The count of real spelling errors (true negatives) that have manual corrections is 1,799 types. There are 1,213 incorrectly detected by MS Spell checker as errors (false negatives). Manual analysis of this list of false negatives is reported in Table 5.
    a. We evaluate MS first order choice of correction candidates for the 1,799 spelling errors against our manually

[10] http://user.phil-fak.uni-duesseldorf.de/~samih/arabic.php

annotated gold standard, resulting in 80.54% accuracy.

b. We also evaluate Hunspell-Ayaspell first order choice for the 1,799 spelling errors, resulting in 45.64% accuracy.

4. We generate a group of candidates for each of spelling the errors (1,799) using edit distance 1 and 2.
5. For the misspelled words with candidates, we want to rank the candidates so that the best correction goes high up in the list. For this purpose we use two approaches: the first uses the Noisy Channel Model alone, and the second uses the Noisy Channel Model with knowledge-based language-specific heuristics.

Type	Count	Examples
Normal Words	493	الأفارقة African الصنعة craftsmanship الطمأنة comforting
Proper Nouns	364	بايدن Biden ماكين McCain مالكولم Malcolm
Newly coined	222	الرقمنة digitalizing الدمقرطة democratization المشخصنة personalized
Newly borrowed	38	لوبيات lobbies الإسلاموفوبيا Islamophobia تابوهات taboos
Colloquial	39	اتنسفت destroyed حواديت tales ماحدش nobody
Unknown	34	صوملته القرسطوية ومدينية
Classical	23	المأزور sinful المأزوم critical المضمخ soaked

Table 5: Analysis of false negatives in the test corpus.

Figure 2 shows a diagram of the procedure for ranking spelling error candidates. The finite state transducer takes input and matches it against our list of 9.3 million words. If a word is not found, it is considered as containing a spelling error and is passed to the 'suggestion list generator' which uses the Edit Distance algorithm to generate all words within distance 1 and 2 from the input word. This is then passed to the candidate list scorer which uses the Noisy Channel Model (trained on the corpus) and Edit Distance augmented with language-specific rules. We post-process the output before displaying suggestions. This process is explained in more detail in the following sections.

4.2.1 Candidate Generation

We integrated a candidate generator in our system in the form of an edit distance finite-state transducer. It generates all possible words that are within edit distance 1and 2 from the entries included in the word list (Oflazer, 1996; Hulden, 2009). It is basically a character-based generator as it replaces each character with all possible characters in the alphabet as well as deleting, inserting, and transposing neighbouring characters. This is a brute-force process that ends up with a huge list of candidates that need to be reduced and scored. To filter out unnecessary words, only words that are found in the word list (by the transducer) are passed to the scorer. This is done by composing two automata $lex\ .o.\ ed$ where lex stands for the lexical items in the word list and ed is the words generated by the edit distance generator.

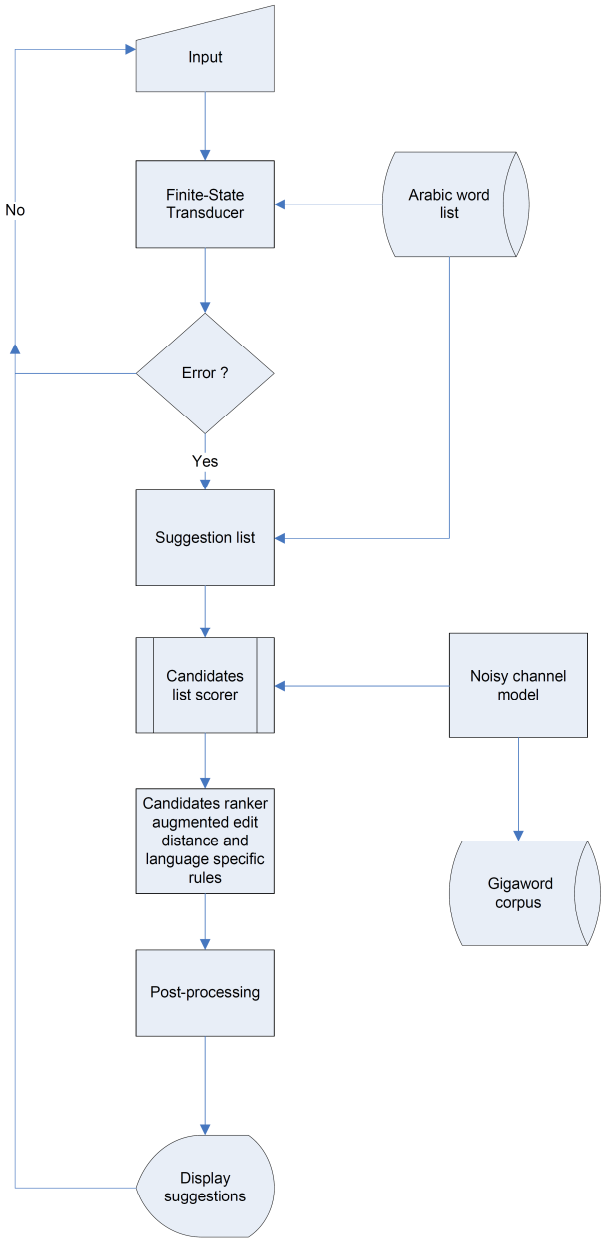

Figure 2. Flow chart of spelling error correction

4.2.2 Scoring the Candidates

Our scoring mechanism is based on the Noisy Channel

Model (Kernigan, 1990). The posterior probability of each plausible candidate is evaluated according to Bayes' formula which is equivalent to:

$$argmax_c p(w|c) = p(w|c)p(c)/p(w)$$

where *p(c)* is the language model or a prior probability of the correction *c; p(w|c)* is the error model or the probability that the word *w* was misspelled when the user meant *c*, *p(w)* is a normalizing constant, $argmax_c$ is the scoring mechanism that computes all plausible values of the correction *c* and maximizes its probability given the original word *w.* These probabilities are trained on our one-billion word corpus.

4.2.3 Final Ranking

First Approach. At this stage, every word is reordered by the scores previously computed by the Noisy Channel Model and a normalized minimum edit-distance so that corrections that exceed a pre-defined threshold are discarded. In this way, our program yields a reduced number of corrections.

Technically speaking, the problem of threshold definition can be empirically solved if it is understood in terms of allowable errors (Davis and de Salles, 2007). Ideally, the maximum number of allowable errors *k* for a string *S* is defined as shown in the following equation:

$$k = (1.0 - \delta) \cdot |S|$$

A Frequency distribution analysis of word lengths in the corpus is used to give an approximate calculation of the individual word matching threshold. The separation of 8 million words from the corpus has revealed that 80% of words have between 4 and 9 characters. As a result, defining a threshold of $\delta=0.87$ means to allow for a maximum of one error in a 3 to 9 letters word, and 2 errors for a 10 to 15 letter word.

	First	Top 5	Top 10
Approach 1: Edit distance 1 & Noisy Channel	68.2%	79.3%	85.1%
Approach 2: Edit distance 1 & Noisy Channel & heuristics	71.3%	82.1%	92.4%

Table 6: Comparing Approaches 1 and 2

Second Approach. In the second approach, we adopt a slightly different but well established method to improve the ranking results. The ranking mechanism we have used so far is based on the Noisy Channel Model and a simple minimum edit where the cost assignment is based on arbitrary letter change. We primed the edit distance scoring mechanism with different rules following the error patterns for Arabic as defined by Shaalan et al. (2003) so that substituting letters belonging to the same groups are evaluated at reduced edit costs (Mitton, 1996).

{ا, أ, إ, آ}, {ب,ت,ث,ن,ي}, {ج,ح,خ}, {د,ذ}, {ر,ز}, {س,ش}, {ص,ض}, {ط,ظ}, {ع,غ}, {ف,ق}, {ه,ة}, {و,ؤ}, {ي,ى}

For example, substituting letters within the same group only costs 0.5 instead of 1.0 . This extension to the basic edit distance improved accuracy of the ranking algorithm by 3.1% (Table 6). The ranking results of Approaches 1 and 2 are reported in Table 6 showing accuracy for having the gold correction in the first, top 5 and top 10 candidates.

Approaches 1 and 2 have been concerned with the re-ranking of candidates. We supplement Approach 2 with a post-processing step that simplistically handles the set of unknown words (227 words) for which no candidates have been generated at all. Inspecting the list of unknown words, we found that 63% of them are two or more words that are joined together, such as التطورالعلمي "scientific development", عبدالدايم "Abdul-Dayem", ولاتريد "and does not want", and مايحدث "what happens". We deal with those words through regular expression replace rules which separates 7 words and particles that are commonly found in the joint word forms. These words are:

{ما ,وما ,لا ,ولا ,أبو ,يا ,عبد}

This simple process corrects 102 words from a total of 105 words affected. This step yields significant improvement on the system's accuracy raising it up to 75% (Table 7). In future research we need to develop a more structure strategy to deal with unknown words, maybe through language modelling or machine learning.

We compare our first order selection (or auto-correction) results against both the Microsoft Spell Checker in Office 2010 and the OpenOffice Hunspell-ar using the Ayaspell Arabic spell checker. Our test results show a significant improvement over Hunspell, yet our results are still below the accuracy of MS Spell Checker, as shown in Table 7.

Spell Checker	First order ranking
MS Spell Checker	80.54%
Hunspell using Ayaspell	45.64%
Approach 1: Edit distance & Noisy Channel	68.20%
Approach 2: Adding heuristics to Edit distance	71.3%
Approach 2 with post-processing	75%

Table 7: First order ranking for 1,799 misspelled words

5. Conclusion

Arabic is a challenging language due its rich and complex morphology. Our research contributes to the open-source community by creating a large and adequate word list for Arabic to be integrated in text authoring tools. We use a tri-gram language model of the allowable vs. unallowable sequences of Arabic characters, which can help in the

validation of existing word lists and making decision on new unseen words. We also create a hybrid spelling correction methodology that significantly outperforms Hunspell in first order ranking of candidates.

Acknowledgements

This research is funded by the Irish Research Council for Science Engineering and Technology (IRCSET), the UAE National Research Foundation (NRF) (Grant No. 0514/2011), the Czech Science Foundation (grant no. P103/12/G084), and the Science Foundation Ireland (Grant No. 07/CE/I1142) as part of the Centre for Next Generation Localisation (www.cngl.ie) at Dublin City University.

References

Attia, Mohammed, Pavel Pecina, Lamia Tounsi, Antonio Toral, Josef van Genabith. (2011). An Open-Source Finite State Morphological Transducer for Modern Standard Arabic. International Workshop on Finite State Methods and Natural Language Processing (FSMNLP). Blois, France.

Beesley, K. R., and Karttunen, L. (2003). *Finite State Morphology*: CSLI studies in computational linguistics. Stanford, Calif.: CSLI.

Ben Othmane Zribi C. and Ben Ahmed M., Efficient Automatic Correction of Misspelled Arabic Words Based on Contextual Information, *Lecture Notes in Computer Science*, (Springer, 2003), Vol. 2773, pp.770–777.

Brill, Eric and Robert C. Moore. 2000. An improved error model for noisy channel spelling correction. ACL '00 Proceedings of the 38th Annual Meeting on Association for Computational Linguistics.

Choudhury, Monojit, Rahul Saraf, Vijit Jain, Animesh Mukherjee, Sudeshna Sarkar and Anupam Basu. 2007. Investigation and modeling of the structure of texting language. *International Journal on Document Analysis and Recognition*. Volume 10, Numbers 3-4, 157-174, DOI: 10.1007/s10032-007-0054-0

Davis, Clodoveu A., Emerson de Salles. (2007). Approximate String Matching for Geographic Names and Personal Names. In Proceedings of GeoInfo'2007. pp.49-60.

Hajič, J., Smrž, O., Buckwalter, T., and Jin, H. (2005). Feature-Based Tagger of Approximations of Functional Arabic Morphology. In: The 4th Workshop on Treebanks and Linguistic Theories (TLT 2005), Barcelona, Spain.

Hassan, Ahmed, Sara Noeman and Hany Hassan. (2008). Language Independent Text Correction using Finite State Automata. IJCNLP. Hyderabad, India

Hulden, Mans. (2009). Fast Approximate String Matching with Finite Automata. Proceedings of SEPLN.

Hulden, Mans. (2009). Foma: a Finite-state compiler and library. EACL '09 Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics. Association for Computational Linguistics Stroudsburg, PA, USA

Kernigan, M., Church, K., Gale W. (1990). A Spelling Correction Program Based on a Noisy Channel Model. AT & T Laboratories, 600 Mountain Ave., Murray Hill, NJ.

Kiraz, G. A. (2001). *Computational Nonlinear Morphology: With Emphasis on Semitic Languages*. Cambridge University Press.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. In: Soviet Physics Doklady, pp. 707-710.

Magdy, Walid and Kareem Darwish. (2006). Arabic OCR error correction using character segment correction, language modeling, and shallow morphology. EMNLP '06 Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing.

Mitton, Roger (1996). *English spelling and the computer.* Harlow, Essex: Longman Group

Oflazer, K. (1996) Error-tolerant finite-state recognition with applications to morphological analysis and spelling correction. Computational Linguistics 22(1): 73-90

Watson, J. (2002). *The Phonology and Morphology of Arabic*, New York: Oxford University Press.

Wintner, Shuly. (2008). Strengths and weaknesses of finite-state technology: a case study in morphological grammar development Natural Language Engineering 14(4):457-469, October 2008.