LOGIC COMPRESSION OF DICTIONARIES FOR MULTILINGUAL SPELLING CHECKERS

Boubaker MEDDEB HAMROUNI

GETA, IMAG-campus (UJF & CNRS)
BP 53, F-38041 Grenoble Cedex 09, FRANCE
Boubaker.Meddeb-Hamrouni@imag.fr

&

WinSoft SA.
34, Bd. de l'Esplanade
F-38000 Grenoble, FRANCE

ABSTRACT

To provide practical spelling checkers on micro-computers, good compression algorithms are essential. Current techniques used to compress lexicons for indo-European languages provide efficient spelling checker. Applying the same methods to languages which have a different morphological system (Arabic, Turkish,...) gives insufficient results. To get better results, we apply other "logical" compression mechanisms based on the structure of the language itself. Experiments with multilingual dictionaries show a significant reduction rate attributable to our logic compression alone and even better results when using our method in conjunction with existing methods.

KEY WORDS: Spelling checkers, Multilinguism, Compression, Dictionary, Finite-state machines.

INTRODUCTION

Since the first work in 1957 by Glantz [6], a great deal of theorizing and research has taken place on the subject of spelling verification and correction. Many commercial products (word processors, desktop presentation,...) include efficient spelling checkers on micro-computers. The classical methods, used are generally based on a morphological analyzer. This is sufficient to provide a robust monolingual spelling checker, but using morphological analyzers can become unrealistic when we want to develop an universal solution. In fact, the analyzers built for each language use various linguistic models and engines, and it is impossible to convert a morphological analyzer from one formalism to another. Furthermore, using these classical methods would lead to combining into the host application as many of grammars and parsers as languages, which would increase the code size and the maintenance problem of rules and data. The method presented in this paper is based on building a dictionary of all surface forms for each language, which is sufficient for spelling checkers applications. The dictionary built with the existing generators can be easily updated manually but may be huge, especially for some agglutinative language (Arabic, Turkish,...). A compression process on the multilingual dictionaries is necessary to obtain a reduced size. The existing compression methods generally used are physical and provide good results for indo-European languages. Applying the same techniques to other languages (Arabic, Turkish,...) shows their limits. For this reason we introduce a new kind of compression techniques that we called "logic compression". This new technique requires a primitive morphological knowledge during the compression process and requires less storage space than previous methods. It also has the advantage of being an universal method applicable to all languages.

Section 1 contains an overview of existing methods for building spell checkers and the limits of such system when we take into account new constraints such as multilingualism. Section 2 outlines the first two steps of our work: we adapt an existing method to Arabic, then make a first extension by introducing a new kind of compression called "logic compression". Section 3 introduces in detail the logic compression with its application to other languages, and shows the improvements obtained when using logic compression in conjunction with existing methods. Section 4 outlines the architecture of our multilingual spelling checker system and some future projects.

I. OVERVIEW OF EXISTING METHODS

I.1. Grammar-based approach

These methods were used in the beginning on early computers when storage space was expensive. It consists in building a small lexicon containing roots and affixes, a grammar of rules that express the morphographemic alternations, and an engine that uses the grammar and the lexicon to see if an input word belongs to the language or not. If the process of recognition fails, some operations (substitution, insertion,...) are performed on the misspelled word to provide a list of candidate words that helps the user to select the correct form.

Even though, it is a great accomplishment to design a powerful engine [3] [8] and to express rules in a pseudo natural way [9] even for different languages [1] [2] [11], these systems present some limits:

- Multilinguism: This methods does not support all languages. To offer a multilingual solution for n languages you have to store n grammars and n lexicons, and generally n different engines into the host application.
- Cost of retrieval: For some languages, the retrieval of words may be long. For instance, a vocalized Arabic spell checker must accept non-vocalized or partially vocalized words which require more time to be accepted than fully vocalized words.
- Cost of guessing alternatives for a misspelled word: To guess a correct word when a misspelled word is found, we have to modify the misspelled word by all possible operations (substitution, insertion, suppression,...) for 1 or 2 characters and then try to check them. This matter can take a lot of time before displaying the correct forms for end-users.
- Maintaining the grammars and data: The grammars and lexicon require continuous updating. You need to find a multilingual computational linguist who knows the linguistic theory and the formalism to easily update data and rules [8].
- Ergonomic features: In some languages, end users want to have some options that let them choose how the spell checker will accept words. In Arabic, for example, different regions have slightly different orthographical conventions.

I.2. Lexical-based approach:

Lexical-based approach appear after the first methods described above, when storage space become less expensive. The first step is to build complete list of surface forms belonging to the language using morphological generators, SLLP (Specialized Languages for Linguistic Programs), etc. and then compresses the large word-dictionary. They are generally used for office applications such as word processors, desktop presentation, etc. Their main advantage is that they cover a complete language since all the forms can be found in the initial list. Also, they allow efficient retrieval and guessing of misspelled words [4]. However, some limits exist in such systems:

- Multilinguism: The compression process give a good ratio for languages with a weak inflexion factor (English,...) where the compression mechanism give up to 150 KB of storage from around 3 MB of a full list [4]. The compression technologies are still powerful for languages with a medium inflexion factor (Russian,...). For example, a list of all surface Russian words of between 10 and 15 MB of size can be reduced to 700 KB [4]. For languages with a high inflexion factor (Arabic, Finnish, Hungarian,...), it won't be easy to find compression technologies that give practical results [4]. For instance, a full list of completed vocalized words in Arabic has 300 MB in size and the current compression methods are impractical.

- No morphological knowledge : These methods are neutral with respect to the text language, the efficiency of compression techniques may be improved by using specific properties of the language [4].

II. A FIRST APPROACH: ADAPTING AN EXISTING METHOD FOR ARABIC

II.1. Using an existing method

As a first step, we take an efficient method used to compress dictionaries for European (English, French,...) spelling checkers [4] and try to apply it to Arabic. The first step of our work consists in building a full list of surface forms using a morphological generator [5] and completed by all irregular forms and existing corpus. The final large word dictionary which covers non-vocalized Arabic has a size of 75 MB. The compression process yields 18 MB in a compressed format. For an idea of the compression process readers can refer to [10]. Table 1 gives some results of the compression process for a few European languages to see the efficiency of the method and its inadequacy for the Arabic language.

	word forms	size uncompressed	size compressed
Danish	448.000	5689 KB	725 KB
German	403.000	5297 KB	866 KB
Arabic	**7 millions**	**75 MB**	**18 MB**
English	88.000	841 KB	224 KB

Table 1

The result for Arabic is impractical for small computers. We must then find other techniques that produce a smaller dictionary or extend this method; to get an exploitable solution.

II.2. Extension of the method:

The initial idea is applied to the morphological system of Arabic. While most of the fully inflected forms words in Arabic are built by adding to a stem prefixes and suffixes we propose replacing some words with only one form beginning by a special code that represents a family of prefixes and finishing by another special code which represents a family of suffixes. For this purpose, we wrote a program in MPW-C that processes a full list of inflected forms and using an existing decomposition of affixes into sub-sets already established, give the reduced lexicon where many forms are replaced by only one representation (PS_i_stem_SS_j) where PS_i (with respect to SS_j) is the set i (with respect to j) of prefixes (with respect to suffixes). Note that the reduced lexicon represents faithfully the initial list without any silence (missing words) or noise (incorrect words). Only compressed words are replaced, and the rest remain in the reduced list. The figure 1 gives an example of words, an example of a decompositions and the obtained result.

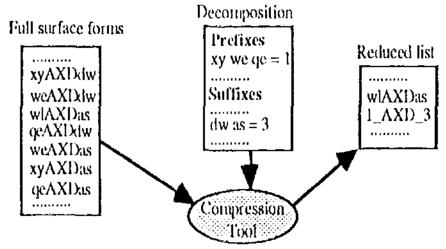

Fig. 1: Example of the compression process

The next crucial problem to resolve is to find the best decomposition that provide the best reduced lexicon. The method must be automatic. It must process the large word-dictionary, and regarding an initial list of prefixes and suffixes, must give as output the best decomposition and the optimal reduced dictionary. But, before studying the implementation of such an algorithm, we began, to see how much space we could gain by this technique starting from a manual decomposition.

• Manual method: Starting from a different full lists for each category of words (transitive verbs, nouns,...), we choose different decompositions and processed the full list with the compression tool. The best decomposition kept for each category was the decomposition which eliminated the maximum forms. This method gave many candidate decompositions depending on the grammatical category of the word. To choose the best global one we took into account the frequency of dictionary entries. This method was tested on different Arabic word lists and some results are described here. Readers can refer to [10] or [11] for more information. To see some decomposition, consider the following sets:

E_1 = {wa, fa}, / و ف / E_2 = {la, sa}, / ل س /
E_3 = {na, aa}, / ن أ / ......
F_1 = {tom, touma, ta, tona}, / تن.../
F_2 = {ya, aân, yina, ouna}, /ان ان ين /......
F_6 = {ha, haâ, ya, ka, kom, kouma, kona, hom, houma, hona, naâ}, F_7 = F_6 \ {ya, naâ} + {ni},
F_9 ={wa}, ...

E_i (with respect to F_j) is a set of prefixes (with respect to suffixes). We note the quantity $E_i.E_j$ (with respect to $F_i.F_j$) all strings built by a concatenation of each element of E_i (with respect to F_i) with each element of E_j (with respect to F_j).

Example of 3 class (from 6) of the prefix class:

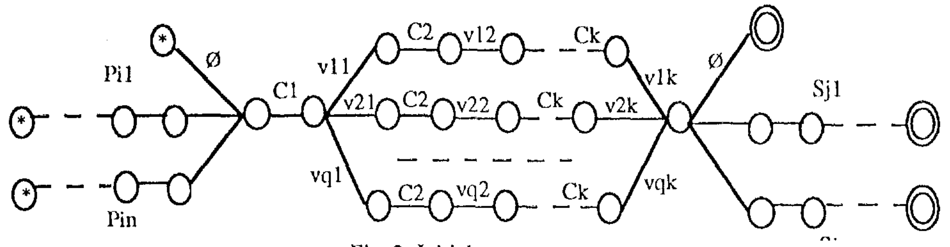

Fig. 2: Initial automaton

$P_1 = E_1$. $P_2 = E_4$.
$P_3 = E_3 + E_2 .E_3 + E_1.E_2 .E_3$
Example of 4 class (from 13) of the suffix class:
$S_1 = F_1$. $S_2 = F_2$. $S_7 = F_7$. $S_8 = F_9.F_7$.

• First results: case of Arabic: With all the classes already found for Arabic (6 classes of prefixes, 13 classes of suffixes; each class containing an average of 8 affixes), we processed a collection of non-vocalized Arabic dictionaries (17 MB), the result gave a reduction lexicon of 254 KB. Used this in combination with the compression process described in § I.2, the final result is 121 KB. Note also that part of this work was implemented in a commercial multilingual word processor (WinText©) to offer Arabic spell checking.

III LOGIC COMPRESSION:

III.1. Theoretical aspects:

Let V be a finite set and V^* the set of words built on V including null strings noted Ø.
$W \in V^*$. $W = W_1W_2...W_n$. $W_i \in V$.
$i \in [1..n]$. Let $V^+ = V^* - \{Ø\}$.
Let Y be a sub-set of V that contain vowels.

1. Prefix(W). $\forall W \in V^+$.
We call order i prefix the quantity:
$P_i = W_1W_2...W_i$. $(1 \leq i \leq n\text{-}1)$.

2. Suffix(W). $\forall W \in V^+$.
We call order j suffix the quantity:
$S_j = W_jW_{j+1}...W_n$. $(1 \leq j \leq n)$.

3. VocPat(W) $\forall W \in V^+$.
We call vocalic pattern of W the set:
$Vy = \{W_i, W_j,...W_k\}$. $W_i \in Y$.
$card(Vy) \leq length(W)$

4. Root(W). $\forall W \in V^+$.
We call root the quantity:
$R = W_p...W_q$. $(1 \leq p < q \leq n)$,
$card(R) \leq q\text{-}p+1$.

5. P_i: Prefixes class. $P_i = \{Ø, P_{i1}, P_{i2},...P_{ik}\}$.
P_{ij} is a prefix. $1 \leq j \leq k$
$Card(P_i) = k + 1$. if $k \geq 1$.
$= 1$. if $P_i = \{Ø\}$.

6. S_j: Suffixes class. $S_j = \{Ø, S_{j1}, S_{j2},...S_{jk}\}$.
S_{ji} is a suffix. $1 \leq i \leq k$
$Card(S_j) = k + 1$. if $k \geq 1$.
$= 1$. if $S_j = \{Ø\}$.

7. V_k: Vowel class.
$V_k = \{Ø, Vy_{k1}, Vy_{k2},...Vy_{kk}\}$
Vy_{ki} is a vocalic pattern. $1 \leq i \leq k$
$Card(V_k) = k + 1$. if $k \geq 1$.
$= 1$. if $V_k = \{Ø\}$.

III.2. Logic Compression: What is it ?

Let's take the following automata that represent some surface vocalized words (fig 2)

P_{ij} is a prefix. $1 \leq j \leq n$.
S_{ji} is a suffix. $1 \leq i \leq n$.
C_i are the consonants of the vocabulary.
$1 \leq i \leq k$.
v_{ij} is the vowel attached to the consonant C_j.
$1 \leq i \leq q$ and $1 \leq j \leq k$.
Ø is the null string.

This automata recognizes all words beginning from an initial state (marked by *) and finishing in a final state (marked by a double circle)
The number of arcs of such an automata is:

$$\sum_{k=1}^{n} \text{length}(Pik) + \sum_{k=1}^{n} \text{length}(Sjk) + 2q(k\text{-}1)$$

If we consider, for example, that affixes have a single character, the number of arcs is equal to $2(n+1) + 2q(k\text{-}1)$.

The logic compression consist in supplying the class of prefixes, suffixes and vowels and replaces each set by only one arc that represent a family of prefixes, suffixes or vowels.

Starting from the following sets already established:
$P_i = \{Ø, P_{i1}, P_{i2},...P_{in}\}$ a class of prefixes stored as x.
$S_j = \{Ø, S_{j1}, S_{j2},...S_{jn}\}$ a class of suffixes stored as y.
$V_k = \{\{v_{11},...v_{1k}\}, \{v_{21},...v_{2k}\},...\{v_{q1},...v_{qk}\}\}$ a class of vocalic pattern stored as z.

The logic compression reduces the initial automaton to this new one:

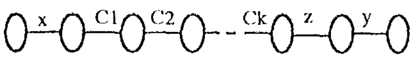

Fig. 3: Reduced automata

The number of arcs kept in the automata is equal to 3 + k. The set V_k contains a sub-set of k vowels which must be applied to the last k characters.

III.3. Experiments:

The logic compression with only an affix decomposition, built by the manual method explained above, has been tested on various list of words that represent collections of multilingual dictionaries (a list of inflected forms). Three languages are tested: non-vocalized Arabic which has a great inflexion factor, French which has a

	Arabic	French	Russian
Size of uncompressed list (MB)	17	2.636	1
Ratio from a complete dictionary	33	80	16
Number of inflected forms	1.980.280	247.406	75.234
Class decomposition (Prefixes)	6	0	3
(suffixes)	13	84	23
1 - Physical compression	5 660	892.646	348.636
2 - Morpho-physical comp.	4 221	311.593	109.418
3 - FSM compression	88	201.216	48.78
4 - Logic compression	253.686	480.770	163.202
4 + 1	145.086	207.376	56.784
4 + 2	121.500	**104.665**	37.74
4 + 3	**57.214**	150.321	**36.71**

Table 2

weak inflexion factor, Russian which has a medium inflexion factor. Experiments are done in two ways. First by using our logic compression alone and, then, in conjunction with other methods by supplying the reduced lexicon (list of compressed words in text format) obtained with our method as input to existing methods. The three other methods tested are the following:

• Physical compression: Using a commercial physical process (Stuffit).

• Morpho-physical compression: This method was used to compress dictionaries used to build a spell checker [4]. It combines morphological proprieties by taking into account the suffixes of the language, but without any link between them. It also contains some physical features [7].

• FSM (Finite-State Machine) Compression: Using the Lexc (Finite State Lexicon Compiler) which allows the conversion of a list of surface forms into a transducer which is then minimized [8].
Results are described in table 2.

III.4. Interpretations:

The most interesting thing observed on this table is the improvement obtained when we combine our method with a previous one. These results show that the existing methods are not optimal and can be improved by our logical compression in its first step. These important results in storage space should not hide others aspects of spell checker systems (retrieval and guessing). It would be interesting if the results given in the table were followed by other results showing improvements in the retrieval and guessing of words.

IV. A PROPOSED ARCHITECTURE OF A UNIVERSAL SPELLING CHECKER:

Figure 3 shows the architecture of our proposed universal spelling checker. Our method is inspired from previous methods (§ I.2), but presents some new original aspects that allow it to be considered a truly multilingual solution. In summary, our system has the following features:

• Multilinguism: this method will insure the multilingual constraint. By using different tools, specific to each language, to create a list of all surface forms.

• Storage space: by introducing the logic compression into the compression process, we will be able to get a reduced lexicon for whatever language we have to use. One task that still remains is to improve the logic compression by making the task of finding the best decomposition more automatic. This problem is combinatorial; we must discover how to apply the optimization algorithms (genetic algorithm, stochastic algorithm,...) in each case to find an optimal reduced lexicon starting from the large word-dictionary and primitive morphological knowledge (list of affixes and vowels).

• Retrieval/guessing: even though we haven't any concrete results now, the first experiments show that the process of checking words in an FSM formalism is faster than other existing methods. Furthermore, we are exploring paths to introduce functions (similarity key,...) into the final obtained lexicon to make a rapid guessing of replacements for misspelled words.

CONCLUSION

Our approach to spell checking differs from previous methods by taking into account a new parameter which is

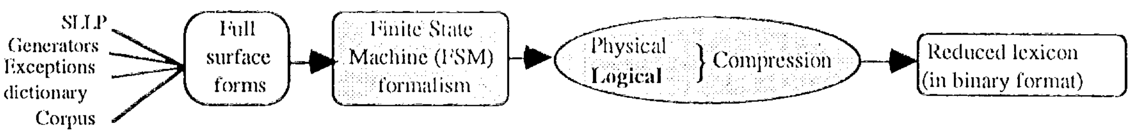

Fig. 3: Universal spelling checker

the multilinguism. The system proposed tries to give solutions for the three main problems: Multilinguism, detection/guessing and storage size.

The first results, although using a manual method to find the decomposition in this first step, show that the previous methods to store dictionaries are not optimal and can be improved by exploring other techniques from the language itself. Another interesting experiment is to find an original optimization algorithm to find the optimal reduced lexicon that represents faithfully the initial list without any silence (missing words) or noise (incorrect words). Yet another project is to build a more robust method for the two other problems (detection and guessing) from the reduced lexicon.

ACKNOWLEDGMENTS

The author would like to thank Prof. Christian BOITET for his constant support and encouragement. I am also very grateful to Mr. Kenneth BEESLEY (Rank Xerox, Grenoble) for his fruitful discussions and Mr. Lauri KARTTUNEN (Rank Xerox, Grenoble) for his help to realize some experiments.

REFERENCES

[1] **Beesley K. R., Bukwalter T., (1989)** *Two-level, Finite-State Analysis of Arabic Morphology.* Proceedings of the Seminar on Bilingual Computing in Arabic and English, 6-7 Sept. 1989. Cambridge, England: The Literary and Linguistic Computing Center & The Center for Middle Eastern Studies.

[2] **Beesley K. R., (1990)** *Finite-state description of Arabic Morphology,* in the Proceeding of the Second Cambridge Conference on Bilingual Computing in Arabic and English, Cambridge, England, 6-7 September 1989. No pagination.

[3] **Ben Hamadou A., (1986)** *A Compression technique for Arabic Dictionaries: The affix Analysis,* in the Proceeding of COLING-86, Bonn 1986, pp. 286-289.

[4] **Circle Noetic Services (1989)** *Passwd,* Reference Manual, MIT Branch Office, Boston, pp. 1-6.

[5] **Circle Noetic Services (1989)** *Conjugate tool,* Reference Manual, MIT Branch Office, Boston, pp. 1-5.

[6] **Glantz H., (1957)** *On the recognition of information with a digital computer,* J. ACM, Vol. 4, No. 2, 178-188.

[7] **Huffman D. A., (1951)** *A method for the construction of minimum redundancy codes,* Proc. IRE 40 (1951), 1098-1101.

[8] **Karttunen L. (1993),** *Finite-State Lexicon Compiler,* Xerox Palo Alto Research Center, April 1993, 1-35.

[9] **Koskeniemmi K., (1983)** *Two level Morphology,* Publication no. 11, Department of General Linguistics, University of Helsinki, pp. 18.

[10] **Meddeb H.B., (1993)** *Intégration d'une composante morphologique pour la compression d'un dictionnaire arabe,* in Proc. Langue Arabe et Technologies Informatiques Avancées, Casablanca, pp. 14.

[11] **Meddeb H.B., (1994)** *Logic Compression of Multilingual dictionaries,* in Proc. of ICEMCO-94, International Conference and Exhibition on Multi-lingual Computing, University of Cambridge, Center of Middle Eastern Studies, London, April-1994, pp. 14.

[12] **Oflazer K, Solak A, (1992)** *Parsing agglutinative word structures and its application to spelling checking for Turkish,* Proc. of COLING-92, Nantes, Aug. 23-28, Vol. 1, pp. 39-45.