A Weighted Finite-State Framework for Correcting Errors in Natural Scene OCR

Richard Beaufort
Multitel Research Center
Belgium
richard.beaufort@multitel.be

Céline Mancas-Thillou
Faculté Polytechnique de Mons
Belgium
celine.thillou@fpms.ac.be

Abstract

With the increasing market of cheap cameras, natural scene text has to be handled in an efficient way. Some works deal with text detection in the image while more recent ones point out the challenge of text extraction and recognition. We propose here an OCR correction system to handle traditional issues of recognizer errors but also the ones due to natural scene images, i.e. *cut characters, artistic display, uncomplete sentences (present in advertisements) and out-of-vocabulary (OOV) words such as acronyms and so on. The main algorithm bases on Finite-State Machines (FSMs) to deal with learned OCR confusions, capital/accented letters and lexicon look-up. Moreover, as OCR is not considered as a black box, several outputs are taken into account to intermingle recognition and correction steps. Based on a public database of natural scene words, detailed results are also presented along with future works.*

1. Introduction

With the emerging explosion of cameras' sales, from low-priced to high professional quality, taking a picture is easy and common. It opens a wide range of applications, especially for automatic processing in order to bring written information to visually impaired people, for example. Camera-based images lead to several degradations, with uneven lighting, sensor noise, various blurs, varying colors and contain a large diversity of text fonts and sizes, with sometimes very artistic display. Character recognition of these natural scenes is much more difficult than scanner-based one and moreover follows the arduous steps of text detection and text extraction. Hence, several additional degradations may be listed such as cut or touching characters due to low-resolution or out-of-field images, inaccurate word segmentation or absence of punctuation due to the low quality of images or by the text itself (for instance, the one printed for advertisements). These errors may be present in traditional OCR but in a more rare way and dealing with all these degradations is a new challenge of natural scene understanding. Correction of character recognition is hence an obvious and mandatory step to substantially increase recognition rates and make applications possible.

To reach this satisfactory goal, correction using FSMs is presented in the following sections. FSMs are widely used to model systems in diverse areas, such as natural language processing or data compression (for instance, a lexicon of several thousands of forms, compiled as an FSM, allows instantaneous accesses while it takes less than 1 MB of memory (RAM)). As they accept the class of regular languages and can be expressed using regular expressions, FSMs have been shown to be a very versatile, descriptive framework: they lead to a compact representation of rewrite rules, used among others for describing linguistic phenomena. FSMs are one of the most widely used models in computer programming in general; they have even been adopted as a part of the well-known Unified Modeling Language (UML).

In order to highlight and compare results presented in this paper, we will present recent works in terms of OCR correction in Section 2 and will give an overview of FSMs in Section 3. We then describe in Section 4 our FSM-based method for correcting non-word errors in natural scene OCR, and present in Section 5 the performance of our algorithm tested on the public database of ICDAR 2003 [1]. Finally in Section 6, we will conclude this paper and browse future works in the field of real-word error correction or OOV words by using additional FSMs representing, among others, syntactic rules.

2. State-of-the-Art of Natural Scene OCR Correction

A natural scene OCR is error-prone in a larger percentage than conventional OCR and a post-processing correction solution is necessary. Main ways of correcting pattern

recognition errors are either multiplication of classifiers to statistically decrease errors by adding information from different computations, such as Lopresti and Zhou proposed for WWW images [9], or by exploiting linguistic information. Commercial OCRs use mainly a lexicon to validate in-dictionary words, and ask the user for feedback on more erroneous words. For embedded natural scene Chinese sign recognition [17], user feedback is also required through dedicated interfaces for OOV words, meaning that no correction is performed. User intervention may be awkward for industrial purposes or meaningless for applications dedicated to blind people.

There are essentially two types of word errors: non-word ones leading to invalid words and real-word errors which are different interpretations from the printed word. Most works deal with non-word errors, handled in this paper. More details for real-word errors will be discussed in Section 6 and may be found in the survey of Kukich [7].

For isolated word correction, lexicons are often used for their error rates decrease. On the contrary, for lightweight methods, systems use probabilistic techniques and n-gram analysis, classically solved through Hidden Markov Models (HMM) or dynamic programming, first used by Neuhoff [14] in text correction. Borovikov et al. [3] have built a HMM-based correction using several post-OCR filters. OCR errors were modeled in terms of a two-layer stochastic process to deal with known and observed characters. Mancas-Thillou et al. [16] have exploited the 3 best OCR outputs to feed a trigram of letters solved using dynamic programming. Due to the absence of a lexicon, real words may be corrected into OOV words and it is very difficult to correct words with a large number of errors.

In contrast, some methods use lexicon look-up. Bunke [4] built an FSM to find out the most similar strings from a vocabulary using minimum edit distance techniques. Nevertheless, a high spatial cost was mentioned as the correction was not driven by errors but by similarity only. Jones et al. [5] described an OCR post-processing system which uses individual steps for correction: rewrite rules, correction of word split errors and use of word bigram probabilities. The three phases interact with others to guide the search but decision has to be taken at each step. Kolak et al. [6] use at run-time a single transducer that takes a sequence of OCR characters as input, and returns a lattice of all possible sequences of real words as output, along with their weights. This transducer is the result of the off-line composition of several transducers trained separately on the same corpus. The main idea of this system is to split each in-dictionary word into its two most probable subsequences of characters (*e.g.*, "example" $\Rightarrow$ "ex | ample" and "exam | ple"), and to propose, from the training corpus, a list of observed corrupted sequences (*e.g.*, "exam" $\Rightarrow$ "exain", "cxam", etc.). A first drawback of this system is this OCR confusion model, which is context-dependent. The second drawback is surely the off-line composition of a single transducer, as no simplification between the different steps is still possible at run-time.

All aforementioned methods consider OCR as a "black box" and start correction independently of OCR results. In contrast, we shall propose an efficient non-word error correction which bases on a finite-state framework for combining the 3 best OCR outputs with trained confusions and linguistic information, among which fast lexicon look-up. This system, designed for use on handheld devices, is lightweight at run-time thanks to simplifications that are performed, without loss of information, after each composition with one of our models.

3. Overview of Finite State Machines

Due to the brevity of this overview, we urge the reader who is not familiar with FSMs to consult the state-of-the-art literature [11, 12, 13, 15]. FSMs, which include finite-state automata (FSAs), finite-state transducers (FSTs) and their weighted counterparts (WFSAs and WFSTs), can be seen as defining both a class of graphs and a class of languages.

On the first interpretation, an FSM is simply an oriented graph with labels on each arc. Transitions of FSAs are labeled with symbols from a single alphabet Σ, while transitions of FSTs are labeled with both input and output symbols belonging to two different alphabets Σ_1 and Σ_2. Weighted machines put weights on transitions in addition to the symbols.

On the second interpretation, FSMs define the class of regular languages. In this definition, an FSA is an *acceptor*: it represents the set of strings over Σ for which there is a path from the initial state to a final state of the graph. In contrast, an FST translates strings of a first language over Σ_1 into strings of a second language over Σ_2; hence, it defines *relations* between languages. In weighted machines, weights, which encode probabilities or distances, are accumulated along paths to compute either the overall weight of a string (in WFSAs), or the overall weight of mapping an input string to an output string (in WFSTs). WFSMs are thus a natural choice for solving the *n*-best-strings problem.

A few fundamental theoretical properties make FSMs very flexible, powerful and efficient. Among them, the composition ($\circ$), a generalization of automaton intersection: from an FST T_1 working on Σ_1 and Σ_2, and an FST T_2 working on Σ_2 and Σ_3, the composition computes their intersection on Σ_2 and builds the FST T_3 working on Σ_1 and Σ_3. Hence, this operation makes it possible to combine different levels of representation for building complex relations from simpler ones. Note that composition can apply to an automaton, in so far as it is considered as the *identity transducer*, *i.e* a transducer in which each symbol is mapped

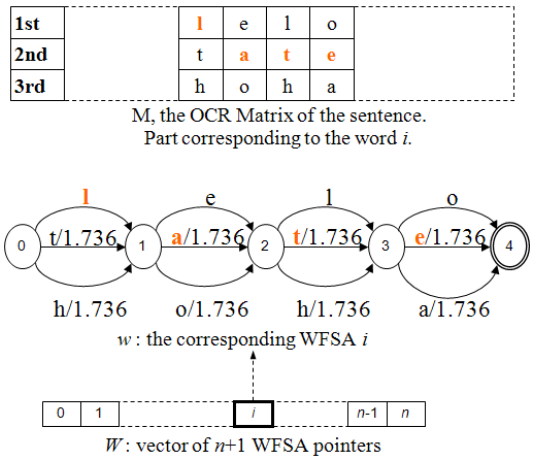

Figure 1. From M, the OCR matrix, to W, the vector of WFSA pointers.

onto itself. Our algorithm directly relies on this operation.

4. FSM-based OCR Correction

Overview. The pseudo-code is shown in Algorithm 1. Our algorithm works on a matrix M containing the 3 best OCR outputs for each letter of the sentence to be recognized. Given a vector S of n space locations, the algorithm starts splitting M into $n+1$ words. As shown in Figure 1, for each word, a dynamic WFSA w is built and stored in the vector W, a vector of WFSA pointers (line 1).

By integrating the 3 best outputs for each letter, w proposes in a simple way all possible successions of letters. In w, each OCR output receives a weight. Our hypothesis is that the best output, which gets often more than 90% of the OCR confidence, has to be privileged, while second and third outputs, that the OCR does not sometimes distinguish enough, should receive the same weight. Hence, we experimentally award a weight 3 times larger for the first output ($-\log_2(1)$) than the weight for both others ($-\log_2(0.33)$). Note that weights are expressed with logarithms in order to add weights instead of multiply them to compute probabilities with several transitions.

The algorithm then iterates on the machines stored in W (lines 3–15). A given WFSA $W[i]$ is combined, by composition, with m (W)FSTs, which represent the different models ($\mathcal{M}$) we take into account (lines 6–8). If the result ρ of these compositions is not empty, the WFSA $\mathrm{B}[i]$ is created (line 10): it contains the best path of ρ, *i.e* the closest form of word i that belongs to our lexicon, given the modifications our intermediate models authorize.

However, if either $\mathrm{B}[i]$ does not exist or its weight is upper than a given threshold, $\mathrm{B}[i]$ is replaced by the best path directly taken from $W[i]$ (line 13). It enables to handle very efficiently OOV words. Experimentally, the threshold is defined as:

$$\mathrm{Threshold}(W[i]) = \mathrm{L}(w_i) \cdot -\log_2(0.25) \qquad (1)$$

where $\mathrm{L}(w_i)$ is the length of word i.

The algorithm ends building, from B, the string corresponding to the complete sentence (line 16).

Algorithm 1 FSM correction

```
1:  W ← FSMVecCreate(M, S, n)
2:  B ← new FSM[n + 1] /* Initialization */
3:  for i = 0 to n do
4:      B[i] ← ∅
5:      ρ ← W[i]
6:      for j = 0 to m and ρ ≠ ∅ do
7:          ρ ← ρ ∘ M[j]
8:      end for
9:      if ρ ≠ ∅ then
10:         B[i] ← FSMGetBestPath(ρ)
11:     end if
12:     if B[i] = ∅ or Weight(B[i]) > Threshold(W[i])
        then
13:         B[i] ← FSMGetBestPath(W[i])
14:     end if
15: end for
16: return StringCreate(B)
```

About the models $\mathcal{M}$. Three models are currently used, ranked as follows: a confusion list, an alphabetic mapping and finally the lexicon itself.

- The confusion list contains weighted confusions like 'i' for 'l' or 'rn' for 'm'. The list itself and the weight associated to each confusion have been learned from the errors of our home-made OCR, launched on the ICDAR 2003 [1] corpus. From this training, the probability of 'i' for 'l' is computed as $\mathrm{P}(l|i)$:

$$\mathrm{P}(l|i) = \frac{\#(i \longmapsto l)}{\#(i)} \qquad (2)$$

  where $(i \longmapsto l)$ means "i for l". The list, from which errors that occurred only once have been removed, contains 231 confusions and has been compiled as a WFST that allows only 1 confusion every 3 letters. This restriction is added to reduce the number of possible answers but it is not strict as recognition rate is larger than 80%. This machine takes 2.2MB[1].

[1] A compiled FSM is a binary dump of the memory. Hence, the size of the result is the same on the HDD and in the RAM.

- An alphabetic mapping is needed because the OCR works on alphanumeric symbols[2], while the lexicon works on the extended ASCII set. The alphabetic mapping allows 2 kinds of conversions: from lower to upper case, and from non-accentuated to accentuated characters (a → [Aäâáàã], n → [Nñ]). Compiled as an FST, the mapping takes 1.5KB.
- The last of these models is and *must be* the lexicon itself, since it directly depends on the targeted language. For French, the lexicon contains 330,000 flexed forms and takes 1.3 MB once compiled. For English, less inflective than French, the lexicon only contains 75,000 forms, but takes 2.4 MB once compiled. Up to now, our lexicons are simple FSAs, but could be WFSAs by allowing for word frequencies, not yet estimated on wide enough corpora.

Why using FSMs? FSMs present great advantages. Among them, flexibility, modularity, and time and space efficiency.

- **Flexibility.** FSMs allow one to represent any kind of knowledge, from the most linguistic to the most mathematical, as far as this knowledge can be expressed as a regular language [15]. It is the case of the models we designed. Flexibility enables also to circumvent frequent errors in natural scene recognition such as the non-detection or recognition of accents.
- **Modularity.** By using composition at run-time instead of compiling a monolithic model once and for all (algorithm 1, lines 6-8), we can easily remove or add a model wherever in the composition process, and directly observe the impact of this modification on the results. For instance, we could add a machine modeling the insertion of spaces in a word just before composing ρ with the lexicon. Modularity is also true for the global process: instead of building a string of letter at the end (line 16), we could concatenate all WFSAs of W, and compose this machine with a language model (either lexical or syntactic), also represented as a WFSA.
- **Time and space efficiency.** Finite-state machines can be turned into canonical forms that allow for optimal time and space efficiency, and entail the decidability of a wide range of questions. At run-time, this is possible by simplifying intermediate results using operations like projection [11], epsilon-free [12] and determinization [13].

As our algorithm aims at computing a measure of similarity between strings (the OCR output against words of the lexicon), the most important feature of this finite-state framework is its ability of easily modeling complex Levenshtein distances. As well known, Levenshtein distance is a measure of the similarity between two strings: the distance is the number of deletions, insertions, or substitutions required to transform a source string into a target string [8]. However, the standard algorithm only allows for one-to-one substitutions, like 'i' for 'l', while *n*-to-*m* substitutions, like 'rn' for 'm' or 'h' for 'li', are simply handled as sequences of one-to-one substitution surrounded by one or more insertions/deletions.

Our intermediate models, the substitution list and the alphanumeric mapping, have been compiled using a homemade compiler able to convert weighted rewrite rules into FSMs [2]. This compiler allows in a simple way the description of weighted *n*-to-*m* substitutions. Hence, our complete algorithm may be seen as a complex Levenshtein distance which easily allows for *n*-to-*m* substitutions, and takes their learned weights into account.

5. Tests and efficiency

First, we used the English ICDAR 2003 corpus [1] in order to train a static confusion list, by comparing the corpus labeling to the decisions of our OCR.

We then made two tests. The first one was on a homemade French corpus of more than 400 natural scene words. With a simple trigram-based correction, the recognition rate was 86.5%. With our FSM-based correction, the recognition rate increased to 94.7% [10]. This good result on a French corpus shows that the confusion list is not specific to the corpus it has been trained on, but is rather a correct estimation of the confusions of the OCR.

The second test was on the ICDAR 2003 database itself, which includes 2268 natural scene words. It compares our system to a commercial OCR[3]. After a convenient segmentation for natural scenes [10], images were processed by the commercial OCR, with an error rate of 29.1%. The test is focused on these errors only. Table 2 compares the 3 first suggestions made by the commercial OCR with the automatic corrections performed by our FSM-based model, provided with either the best or the 3 best OCR outputs. Our model clearly outperforms the commercial system and the use of the 3 best OCR outputs highly improves our own results. The only drawback in using the 3 best outputs is the growing amount of wrong corrections on OOV: it increases from 0.6% to 8.6%. This means that adding more information with the 3 best outputs leads to better correction and global recognition, but also adds confusion in a minor way.

Our corrector efficiently deals with *n*-to-*m* substitutions. The word 'Office' (Figure 3, left) was recognized as 'Ohice'

[2] 36 symbols: [a-z] and [0-9].

[3] ABBYY FineReader 8.0 Professional Edition Try&Buy.

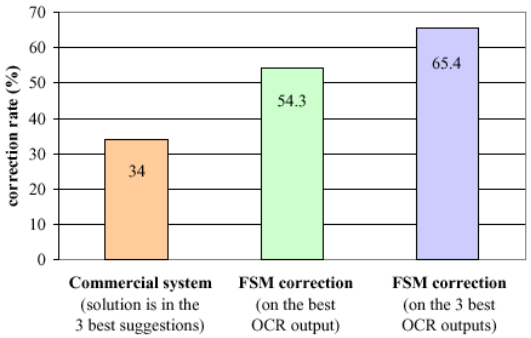

Figure 2. Tests on the ICDAR 2003 database

and the commercial OCR could not propose the proper word. Suggestions it does are based on the Levenshtein distance between the recognized word and in-dictionary words. On the contrary, our corrector finds the proper word. The word 'form' (Figure 3, right) is a typical case of cut characters in natural scenes. Our OCR recognized it as 'fnrm' but proposed an 'o' as second best output for the second letter which enables our corrector to find the word properly and to rank 'form' before 'farm'.

The efficiency of our FSM-based model has been tested on a classical PDA (PocketPC 2003, 520 MHz): it corrects a word of 13-character length or less in 0.2s.

6. Conclusion and Future Works

An end-to-end FSM-based correction with no intermediate decision between each step has been proposed to correct natural scene text. Based on finite state machines to highlight flexibility, modularity in a lightweight way, linguistic information was intermingled with several best outputs of a recognizer. This has proven efficiency in a public database to handle usual cases of correction, i.e insertion, deletion and substitution. Moreover, the presence of OOV words in natural scenes is quite usual and our corrector deals quite well with this difficulty. Nevertheless, results may be still improved by using an additional machine based on morphemes instead of words to permit the correction of OOV words. For example 'Constantine' is an OOV word present in this database and was recognized as 'Comstantine'. With a morpheme-based machine, 'Coms' would be easily turned into 'Cons', even if it leads to an OOV word. In addition, a syntactic machine may follow our main chain to correct real words with other in-dictionary words but also to discriminate good answers after correction. The example could be for the word 'form' expressed in Section 5 where the correction was good in this case but could be 'farm' in another context. Hence using syntax may improve results in a higher level of linguistic information. Actually, these two

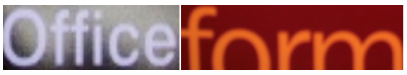

Figure 3. Two interesting images that our corrector handles properly.

latest machines are part of our future works.

References

[1] *Robust reading competition*, 2003. `http://algoval.essex.ac.uk/icdar`.

[2] R. Beaufort. *FSM Library: description de l'API*, 2006. Multitel tech. report.

[3] E. Borovikov, I. Zavorin, and M. Turner. A filter based post-ocr accuracy boost system. In *1st ACM Workshop on Hardcopy Document Processing*, pages 23–28, 2004.

[4] H. Bunke. Fast approximate matching of words against a dictionary. *Computing*, 55(1):75–89, 1995.

[5] M. Jones, G. Story, and B. Ballard. Integrating multiple knowledge sources in a bayesian OCR post-processor. In *Int. Conf. on Document Analysis and Recognition*, volume 1, pages 925–933, 1991.

[6] O. Kolak, W. Byrne, and P. Resnik. A generative probabilistic OCR model for NLP applications. In *Conf. of NAACL-Human Language Technology*, volume 1, pages 55–62, 2003.

[7] K. Kukich. Techniques for automatically correcting words in text. *ACM Computing Surveys*, 24(4):377–439, 1992.

[8] V. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics*, 10:707–710, 1966.

[9] D. Lopresti and J. Zhou. Using consensus sequence voting to correct OCR errors. *Computer Vision and Image Understanding*, 67(1):39–47, 1997.

[10] C. Mancas-Thillou. *Natural Scene Text Understanding*. PhD thesis, Faculté Polytechnique de Mons, Belgium, 2006.

[11] M. Mohri, F. Pereira, and M. Riley. The design principles of a weighted finite-state transducer library. *Theoretical Computer Science*, 231(1):17–32, 2000.

[12] M. Mohri, F. Pereira, and M. Riley. Generic ϵ-removal algorithm for weighted automata. *Lecture Notes in Computer Science*, 2088:230–242, 2001.

[13] M. Mohri and M. Riley. Weighted determinization and minimization for large vocabulary speech recognition. In *Eurospeech'97*, pages 131–134, 1997.

[14] D. Neuhoff. The Viterbi algorithm as an aid in text recognition. *IEEE Trans. Information Theory*, 21:222–226, 1975.

[15] E. Roche and Y. Schabes, editors. *Finite-state language processing*. MIT Press, Cambridge, Massachusetts, 1997.

[16] C. Thillou, S. Ferreira, and B. Gosselin. An embedded application for degraded text recognition. *Eurasip Jour. on Applied Signal Processing*, 13:2127–2135, 2005.

[17] J. Zhang, X. Chen, A. Hanneman, J. Yang, and A. Waibel. A robust approach for recognition of text embedded in natural scenes. In *Int. Conf. on Pattern Recognition*, 2002.