

# Classification and Postprocessing of Documents Using an Error-correcting Parser

H. Bunke, R. Liviero

Institut für Informatik und angewandte Mathematik, University of Bern  
Neubrückestrasse 10, CH-3012 Bern, Switzerland  
bunke@iam.unibe.ch

## Abstract

*In this paper an error-correcting parsing algorithm and its application to a postprocessing task in the context of automatic check processing is described. The proposed method has shown very good results in terms of recognition accuracy and execution speed on both real and synthetic data.*

## 1 Introduction

The recognition of machine printed characters has been intensively studied during the past years and significant progress has been made [1]. For example, there exist commercial OCR systems that achieve a correct recognition rate of over 99% today [2]. But depending on the particular application, such a high recognition rate may be still insufficient. In order to further improve recognition accuracy, contextual postprocessing is often very useful. Different contextual postprocessing methods have been proposed in the literature. A recent survey has been given in [3]. For earlier overviews see [4, 5].

In the present paper we propose the application of finite state automata and error-correcting parsing to solve a particular postprocessing problem occurring in the context of automatic check reading. The proposed method is not only an aid to recover from OCR errors but also to classify a document, i.e. a check, based on its contents in the presence of OCR errors. The present paper is a shortened version of [6].

## 2 Theoretical Foundations

In this section we give a brief review of error-correcting parsing, which serves as theoretical foundation of the method described in Section 3. The algorithm presented in this section is a restricted version of the parser introduced in [7]. A similar algorithm have been described in [8].

We consider a finite alphabet  $X = \{x_1, \dots, x_n\}$  of symbols. The set of all words over  $X$ , including the empty word  $\epsilon$ , is denoted by  $X^*$ . A (deterministic) finite state automaton (fsa) over  $X$  is a 5-tuple  $A = (Q, X, \delta, q_0, F)$  where  $Q$  is the finite set of states,  $q_0 \in Q$  is the initial state,  $F \subset Q$  is the set of final states, and  $\delta : Q \times X \rightarrow Q$  is the transition function. The transition function can be extended to  $\delta : Q \times X^* \rightarrow Q$  by defining  $\delta(q, \epsilon) = q$  for any  $q \in F$ , and  $\delta(q, xa) = \delta(\delta(q, x), a)$  for any  $x \in X^*$  and  $a \in X$ . The language  $L(A)$  accepted by a fsa  $A$  is defined by  $L(A) = \{x \mid x \in X^* \wedge \delta(q_0, x) \in F\}$ . This means that  $L(A)$  consists of all words  $x$  over  $X$  for which there exists a sequence of state transitions, defined by  $\delta$ , from  $q_0$  to a final state. It is well known that the class of languages accepted by fsa's is identical to the class of languages of Chomsky-type 3. These languages are also called *regular*.

The errors produced by OCR devices can be classified into three types, namely deletion, insertion, and substitution of a symbol. These three types of errors are also called edit operations. In order to model the fact that certain errors are more likely to occur than others, each edit operation  $s$  gets assigned a cost  $c(s)$ , which is a non-negative integer. Given a sequence  $S = s_1, s_2, \dots, s_n$  of  $n$  edit operations, its cost is defined as  $c(S) = \sum_{i=1}^n c(s_i)$ . Now for any two words  $x$  and  $y$  over an alphabet  $X$ , the string distance  $d(x, y)$  is defined as the minimum cost taken over all sequences of edit operations that transform  $x$  into  $y$ . Formally,  $d(x, y) = \min\{c(S) \mid S \text{ is a sequence of edit operations transforming } x \text{ into } y\}$ . An algorithm for the computation of  $d(x, y)$  have been described in [9].

The task of a parser is to decide, for a given word  $x$  and language  $L$ , if  $x \in L$ . The fsa  $A$  of any language  $L = L(A)$  can be used as a parser in a straightforward way by starting with the initial state and traversing the states of  $A$  according to  $\delta$  and the actual input word  $x$ . After reading  $x$  we are in some state  $\delta(q_0, x)$ .

### algorithm error-correcting parser

**input:** a fsa  $A = (Q, X, \delta, q_0, F)$  and input  $x = x_1 \dots x_n$   
**output:**  $d = \min\{d(x, z) | z \in L(A)\}$   
**method:**  
*/\*initialization\*/*  
 $L(0) := \{(q_0, 0)\};$   
**for**  $i = 1$  **to**  $n$  **do**  $L(i) = \emptyset;$   
*/\*main loop\*/*  
**for**  $i = 1$  **to**  $n$  **do** {  
  **repeat** {  
    **for all**  $(q, c) \in L(i)$  **do** {  
       $\text{add}[(q, c + \text{ins}), L(i + 1)];$  */\*insertion\*/*  
      **for all**  $q' \in \text{next}(q)$  **do** {  
         $\text{add}[(q', c + \text{del}), L(i)];$  */\*deletion\*/*  
        **if**  $\delta(q, x_{i+1}) = q'$  **then**  $\text{add}[(q', c), L(i + 1)]$   
        */\*match\*/*  
        **else**  $\text{add}[(q', c + \text{sub}), L(i + 1)]$   
        */\*substitution\*/*  
      **until** no more elements can be added to  $L(i)$   
       $d := \min_c \{(q, c) | (q, c) \in L(i), q \in F\}$   
      **end** error-correcting parser  
    }
  }
}

Figure 1: The error-correcting parsing algorithm

Now if  $\delta(q_0, x) \in F$  then we conclude  $x \in L(A)$ , otherwise  $x \notin L(A)$ . In error-correcting parsing, we are given a fsa  $A$  and a string  $x$  that does not necessarily belong to  $L(A)$ . If  $x \in L(A)$  then the error correcting parser is supposed to report this fact. Otherwise, if  $x \notin L(A)$ , the error-correcting parser has to find string  $y$  such that  $d(x, y) = \min\{d(x, z) | z \in L(A)\}$  and outputs  $d(x, y)$ . In other words, it has to find that element of  $L(A)$  that has the smallest edit distance to the input  $x$ .

The pseudo code of an error-correcting parsing algorithm for Chomsky-type 3 languages is given in Fig. 1. This algorithm constructs a list  $L(i)$  for each input symbol  $x_i$ . Additionally, there is an initial list  $L(0)$ . Each list contains a number of elements of the form  $(q, c)$  where  $q \in Q$  and  $c$  is the cost of a sequence of edit operations. More precisely, if  $(q, c) \in L(i)$  then there exists a string  $y$  such that  $d(x_1 \dots x_i, y) = c$  and  $\delta(q_0, y) = q$ . In other words, if  $L(i)$  contains  $(q, c)$  then we know that after reading  $x_1 \dots x_i$  state  $q$  can be reached if we apply a sequence of edit operations with cost  $c$  to  $x_1 \dots x_i$ . Furthermore, we know that there is no other such sequence of edit operations with a smaller cost. This property implies that after  $L(n)$  has been constructed, the cost  $c$  of the element  $(q, c) = \min_{c'} \{(q', c') | q' \in F\}$  is our desired output.

In the formulation of the algorithm given in Fig. 1, we assume constant cost *ins*, *del* and *subst* for any insertion, deletion, and substitution, respectively. These costs are global variables to the algorithm. However, the algorithm can be easily extended to the case where each insertion, deletion, and substitution may have its individual cost.

The algorithm uses two functions. The function  $\text{next}(q)$  returns the set of successor states of  $q \in Q$  under any symbol. More precisely,  $\text{next}(q) = \{q' | \delta(q, a) = q', a \in X\}$ . Let  $L(i)$  be a list and  $(q, c)$  a list element. Then  $\text{add}((q, c), L(i))$  constructs a new list  $L'(i)$  as specified below

$$L'(i) = \begin{cases} L(i) \cup (q, c) & \text{if case 1} \\ (L(i) - (q, c')) \cup (q, c) & \text{if case 2} \\ L(i) & \text{otherwise} \end{cases}$$

Case 1 means that  $L(i)$  contains no list element  $(q, c')$  for any  $c'$ ; case 2 means that  $L(i)$  contains a list element  $(q, c')$  with  $c < c'$ . Obviously,  $\text{add}((q, c), L(i))$  keeps track of the minimum cost necessary to reach a certain state after reading  $x_1 \dots x_i$ . It can be easily concluded that the time and space complexity of the error-correcting parsing algorithm are  $O(n \cdot m)$ , where  $n$  is the length of the input word and  $m$  denotes the number of states of the fsa.

It is easy to augment the algorithm shown in Fig. 1 by pointers that allow to extract the word  $y$  in the language that has the minimum edit distance to the input, i.e. the word  $y$  with  $d(x, y) = \min\{d(x, z) | z \in L(A)\}$ . The pointers just indicate for each list element  $(q, c)$  from which other element it has been generated by means of which edit operation.

### 3 Problem Description and Proposed Solution

The application area considered in this paper is the automatic reading of checks. Over thirty different types of checks, each having an individual layout format, are commonly used for money transfer in Switzerland. An example is shown in Fig. 2. Although a large number of such checks are submitted daily, their processing at banks and post offices is only partly automated. That is, only the so-called coding line on a check is read by machine. The coding line is in the lower right part of a check. Its location is predefined and is the same for all different types, i.e. layout formats, of checks. For a graphical illustration see Fig. 2.

The coding line of each check follows a predefined format. This format, however, depends on the particular type of check. The definition of the format of the

position	meaning	possible value
1-2	check subcategorie	one out of {01, 03, 11}
3-12	amount	any sequence of digits
13	parity digit 1	parity check for position 1-12
14	delimiter	>
15-40	reference number	any sequence
41	parity digit 2	parity check for positions 15-40
42-43	delimiter	+ <i>space</i>
44-51	customer identification	any sequence of digits
52	parity digit 3	parity check for positions 44-51
53	delimiter	>

Table 1: Format definition of coding line on the check in Fig. 1.

position	meaning	possible value
1-2	check subcategorie	one out of {46, 47, 56, 57}
3	parity digit 1	parity check for positoin 1-2
4	delimiter	>
5-24	reference number	any sequence of digits
25-30	deadline	date in format YYMMDD
31	parity digit 2	parity check for positions 5-30
32-33	delimiter	+ <i>space</i>
34-41	customer identification	any sequence of digits
42	parity digit 3	parity check for positions 34-41
43	delimiter	>

Table 2: Format definition of coding line of another type of check.

coding line of the check in Fig. 2 is given in Table 1. The format definition of another type of check is given in Table 2. The ultimate goal of automatically reading the coding line on a check is not only to correctly recognize the sequence of characters on the coding line, but also to infer the meaning of each character. That is, one wants to assign an interpretation to each character in the sense of the definition shown in Table 1 or 2. For example, when processing the check in Fig. 2, we intend to derive a result similar to Table 3.

Apparently, if the type of a check were known, the inference of the meaning of each character on the coding line would be more or less trivial because the format of the coding line for a given type of check is precisely defined. In reality, however, the type of a check is not known as only the coding line on a check - and nothing else - is captured by the scanning device. Therefore, in order to infer the meaning of each character on the coding line, we first have to determine the type of the actual check using only the sequence of characters on the coding line. Solving this task is not trivial as at least two subproblems are encountered. First, the formats of the coding lines of different types of checks may be similar to each other, and secondly, there may be OCR errors resulting in the insertion, deletion, or substitution of characters on the coding line.

In our system, we have a module that digitizes the coding line on a check, extracts the individual characters, and feeds them into an OCR program. The

type of information	value
check subcategorie	01
amount	187.50 Fr.
reference number	20011282367002209310248139
customer identification	01000064

Table 3: Result of automatic processing of Fig. 1.

postprocessing module compares the sequence of characters output by the OCR module to the format definitions of the coding lines and determines the type of check that fits best. This process yields the meaning of each character on the coding line (as shown in Table 3) as a by-product. Our actual comparison procedure is an error-correcting parser that is controlled by a regular grammar, which describes the coding line formats of all different types of checks. In the present paper, we concentrate on the postprocessing module.

The legal symbols occurring on the coding line are from the alphabet  $X = \{0, 1, \dots, 9, <, >, +, \textit{space}\}$ . The coding line of each type of check consists of a sequence of logical units, where a logical unit is one of the following (see also Table 1 and 2):(1) a sequence of fixed length  $l \geq 1$  of arbitrary symbols from a subset of  $X$ ; (2) one out of a finite number of constant sequences of symbols; (3) a range of integer values; (4) a date; (5) a parity digit. Obviously each of these logical units can be represented by a fsa in a straightforward way. Consequently, any coding line can be represented by concatenation a number of fsa's, each defining one of the types (1) to (5) from the list above.

The fsa's that represent the coding lines of the different types of checks in our system have been generated from their definitions. Given these fsa's and the sequence of symbols output by the OCR-module, the error-correcting parser described in Section 2 can be applied. It determines the most similar type of check for a given input coding line based on the minimum edit distance. Thus the actual check can be classified into one of the types defined a priori. Evaluating the pointers set by the algorithm, the meaning of the characters on the actual coding line can be determined (see Table 3).

## 4 Experimental Results

The error-correcting parser described in Section 2 has been implemented in C under MS-DOS and UNIX and runs on both personal computers and workstations. As the printing quality of the characters on the coding lines of the checks under consideration is generally quite good, the error rate of the OCR-module can be expected fairly low. Consequently, we have

	T=0	T=1	T=2
C	99.27	99.67	100.00
R	0.73	0.33	0.00
E	0.00	0.00	0.00
L	100.00	100.00	100.00

Table 4: Result of the first experiment ( $R$  = rejection rate,  $E$  = error rate,  $L$  = reliability)

defined an error threshold  $T$  for our error-correcting parser. As soon as the cost  $c$  of a pair  $(q, c)$  in any of the lists  $L(i)$  exceeds this threshold, i.e.  $c > T$ , the item  $(q, c)$  is not included in  $L(i)$ . Practically, this prevents any item which will not contribute to the final solution from being considered, and thus speeds up the algorithm. Theoretically, it reduces the time complexity of the parser from  $O(n \cdot m)$  to  $O(n \cdot T)$  (see Section 2). The concrete value of  $T$  has been varied in our experiments as will be described below.

A number of experiments were done aiming at the classification of a check into its type based on the output of the OCR-module. The 14 most frequent check types, i.e. 14 different fsa's, were used in these experiments. As OCR-module, a commercial product was used. Particularly, we were interested in error rate and reliability depending on the error threshold  $T$ . Let  $N = N_1 + N_2 + N_3$  where  $N$  denotes the total number of checks, and  $N_1, N_2, N_3$  are the number of rejected, correctly, and incorrectly classified checks, respectively. From these numbers, we define the rejection rate  $R = N_1/N$ , the correct recognition rate  $C = N_2/N$ , the substitution error rate  $E = N_3/N$ , and the reliability rate  $L = N_2/(N - N_1)$ . We will say that the word  $x$  has the distance  $i \geq 1$  to the language  $L(A)$ ,  $d(x, L(A)) = i$ , where  $A$  is a fsa, if (1)  $x \notin L(A)$ , (2) there exists  $y \in L(A)$  with  $d(x, y) = i$ , and (3) there is no  $z \in L(A)$  with  $d(x, z) < d(x, y)$ . If  $x \in L(A)$  then  $x$  has distance zero to  $L(A)$ .

In our first experiment, we used 2'455 coding lines that came from real checks. The result of this experiment is shown in Table 4. There were 99,27% of all checks correctly classified with  $T = 0$ . This means that one or more OCR error occurred in 0,73% of all checks such that a word  $x \in L(A)$  was transformed into another word  $x' \notin L(A)$ . As  $E = 0$ , no word  $x \in L(A)$  was transformed into another word  $x' \in L(A')$ ,  $A' \neq A$ . With  $T = 1$ , all distorted words  $x$  with distance 1 to  $L(A)$  have been correctly classified. The remaining words  $x'$  were rejected because  $d(x', L(A)) > 1$  for any fsa  $A$ . Finally, setting  $T = 2$ , all words were correctly classified. For  $T = 2$ , the execution speed is over 100 documents per second on a

pc.

It can be concluded from the first experiment that the error-correcting parser proposed in this paper is a very suitable tool for the classification of checks in a real world scenario. In order to reveal the limitations of the method, we did another experiment with more difficult data, that were artificially generated. The results of this experiment are reported in [6].

## 5 Discussion and Conclusions

A postprocessing module for automatic check processing was proposed in this paper. It is based on an error correcting parser for regular languages. The method has been tested on a large number of real and synthesized data, and has shown very good performance, in terms of classification and error-correcting accuracy, and computational efficiency. In an experiment with over 2'000 real checks, a correct classification rate of 100% has been achieved with an appropriate error threshold  $T = 2$ .

One additional strength of the method is that it can be easily adapted to new types of coding lines. Earlier (commercial) postprocessing modules were mainly "handcrafted", i.e. heuristically designed<sup>1</sup>. A serious drawback of this approach is that the whole postprocessing module has to be redesigned from scratch if a new type of check is to be taken into account, or an old one is redefined. By contrast, in the present system, all format definitions can be kept in a database and automatically converted into their corresponding fsa<sup>2</sup>. Thus, any updates or modifications of the coding line format definitions can be handled by our system at almost zero cost.

A theoretical alternative to the method proposed in this paper is not to represent a coding line by means of a fsa, but by the finite set of all its possible instances, i.e. words, and to use an algorithm for string edit distance computation [9] instead of the error-correcting parser. As the number of different coding lines is finite for any type of check, this method is equivalent to the one proposed in this paper from the theoretical point of view. In practice, however, it can be expected much slower because of the large number of different prototype strings that are to be tested.


Finally, we would like to mention that the parser described in Section 2 is not restricted to the application described in Section 3. It is rather a general tool that may have applications in many other OCR contextual postprocessing tasks.

<sup>1</sup>According to various personal communications.

<sup>2</sup>This feature is included in our present implementation.

## References

- [1] Pavlidis, T. and Mori, S. (eds.): Optical Character Recognition, Special Issue of Proceedings of the IEEE, Vol. 80, No. 7, July 1992, 1027-1209
- [2] Rice, S.V., Kanai, J. and Norther, T.A.: An evaluation of OCR accuracy, in UNLV Inform. Sci. Research Inst., Annual Reptot, 1993, 9-39
- [3] Kukich, K.: Techniques for automatically correcting words in text, ACM Comp. Surveys, Vol. 24, No. 4, 1992, 377-439
- [4] Elliman, D.G. and Lancaster, I.T.: A review of segmentation and contextual analysis techniques for text recognition, Pattern Recognition, Vol. 23, No. 3/4, 1990, 337-346
- [5] Srihari, S.N.(ed.): Computer Text Recognition and Error Correction, Tutorial, IEEE Computer Society Press, Silver Spring, MD, 1985
- [6] Bunke, H., Liviero, R.: An error-correcting parser for the postprocessing of documents, Technical Report IAM-095-xxx, Department of Comp. Science, Univ. of Bern, 1995
- [7] Aho, A.V. and Peterson, T.G.: A minimum distance error-correcting parser for context-free languages, SIAM J. Computing 1, 1972, 305-312
- [8] Wagner, A.: Order-n correction for regular languages, CACM, Vol. 17, No. 5, 1974, 265-268
- [9] Wagner, R.A. and Fischer, M.J.: String-to-string correction problem, Journal of the ACM, Vol. 21, No. 1, 1974, 168-173

Empfangsschein / Récépissé / Ricevuta	Einzahlung Giro PTT	Versement Virement PTT	Versamento Girata PTT				
Einzahlung für / Versement pour / Versamento per <b>Ferrmeldedirektion (FD)</b> Direction des télécommunications (DT) Direzione delle telecomunicazioni (DT) Tel.113 <b>3030 BERN</b>  Konto / Compte / Conto <b>01-64-6</b>  Fr. <table border="1"><tr><td>187</td><td>50</td></tr></table> Einbezahlt von / Versé par / Versato da  <b>20 01128 23670</b> <b>02209 31024 81391</b> <b>Thien Ha Minh</b> <b>Zähringerstr. 14</b> <b>3012 Bern</b>	187	50	Einzahlung für / Versement pour / Versamento per <b>Ferrmeldedirektion (FD)</b> Direction des télécommunications (DT) Direzione delle telecomunicazioni (DT) Tel.113 <b>3030 BERN</b>  Konto / Compte / Conto <b>01-64-6</b>  Fr. <table border="1"><tr><td>187</td><td>50</td></tr></table>	187	50	Bitte keine Mitteilungen anbringen Pas de communications s.v.p. Non aggiungete comunicazioni p.f.  Kontroll-Nr. N° de contrôle N° di controllo <b>031 24 81 39</b> Bei Rückfragen bitte diese Kontroll-Nr. angeben. Dans la correspondance, prière d'indiquer ce n° de contrôle. Nella corrispondenza vogliete indicare questo n° di controllo.  Giro aus Konto Virement du compte Girata dal conto .....  Referenz-Nr./N° de référence/N° di riferimento <b>20 01128 23670 02209 31024 81391</b>  Einbezahlt von / Versé par / Versato da  <b>Thien Ha Minh</b> <b>Ingenieur</b> <b>Zähringerstr. 14</b> <b>3012 Bern</b>	
187	50						
187	50						

010000187503>200112823670022093102481391+ 010000646>

PT 083.43 (1/20 64) 552 ERZ

Die Annahmestelle  
L'office de dépôt  
L'ufficio d'accettazione

Figure 2: Example of a check