Natural Language Names Recognition Method – Fault Tolerant to the Most Common Mistypings (*)

Dimo T. Dimov

Institute of Information Technologies (IIT)
Bulgarian Academy of Sciences (BAS)
1113 Sofia, Acad. G. Bontchev Str., Block 29-A
dtdim@bgcict.acad.bg

Abstract: A heuristic method is proposed in the paper – for a fault-tolerant key-access to database records of natural (and/or artificial) language texts using an extra-dictionary of keys. The "approximate-string-matching" problem known from the practice of information retrieval systems can be resolved by the method, namely the problem for obtaining the proper key being a character string of substantial length that the most common typist errors should to be expected (i.e. character substitution, omission, insertion, neighbouring reversal, etc.). A new organization of dictionary search has been proposed through a direct construction of hypotheses for words and texts under recognition, based on a preliminary knowledge about statistics of the most probable input errors. The "similarity of strings" relation is build following the principle of "clustering without metrics" through a heuristic model of probability kind over the essential sets of strings only, in this way improving the effectiveness of the method in comparison with known methods based on metrics (most frequently versions of the "Damerau-Lewenstein" metric). Finally, a comparative analysis of results is briefly reported for two experiment systems: (i) a geographic information system (about 3500 objects for search) and (ii) an English-Bulgarian computer dictionary (of about 22000 words and phrases).

Key-words: Natural language text processing, Approximate string matching, Most probable typist errors correction, Database access, Recognition/classification improvement by post-processing using a dictionary.

1 Introduction

The paper considers recognition of character strings and, in particular, recognition of natural or artificial language texts with a view to finding ways to correct the most probable mistypings, thus building a fault-tolerant database access method. Using a key-based access to database records, the key being a character string of substantial length, the method is tolerant to most common typist errors. To this end, it offers an efficient solution to the problem of **approximate string matching** [1, 2, 8] known from the theory and practice of information retrieval systems [11]. This approach can be also applied to the task of improving plausibility in text (or graphics) recognition, consequently, in document analysis systems [3, 9, 12, 14].

Searching data by key is a basic operation in Data-Base Management Systems (**DBMS**). Data are accessed not only by a so-called primary key but also by a number of auxiliary keys (primary or secondary ones). A wide range of access methods is developed, e.g. index, index-sequential, direct (by Hash-functions), associative etc., which perform a logical-level linear ordering of the database records [10, 11].

(*) This research has been worked out in the frames of following projects:

- "Handwritten Text Recognition Using a Dictionary of Patterns", supported by BAS (IIT No. 010011).
- "Written and Printed Text Recognition: Methods and Information Technology for Cyrilics", supported by the National Science Fund, Ministry of EST, Bulgaria, contract No. I-524/95 (IIT No. 020025).

The problem is not encountered until the key length remains relatively small, i.e. with little or no extra information (ordinal number, product code etc.). But when it exceeds 5÷6 characters (as in names, text descriptors etc.) the problem arises of possible errors in the data input via keyboard. Against the natural information surplus in such cases being so great, the human operator would scarcely be able to detect his typing errors. Studies cited in [8] on a large number of text databases show that these errors are stable in statistical distribution. Among them, most common are the so-called "typist errors" (or mistypings, or misspellings), namely – typing the wrong character ("substitution"), omitting a character, inserting extra character, exchanging two neighbouring characters (neighbouring reversal) etc., in descendant order by their frequency. The total frequency of these errors (about 10%) is over ten times more than the frequency of all the rest of mistypings [8]. On the other hand, it is not clear when the error is entered – at the moment of query or even before, during the database loading with text data. Anyway, as the mistyped text will hardly match any database entry, the system will report a search failure [11].

Two main approaches to this problem are known – (i) equipping the DBMS with an automatic error correction utility, and (ii) allowing the human operator to make a decision interactively. Possible "collisions" during error correction can be resolved by the operator, using the latter approach, although currently it is considered inappropriate [8, 11]. The former approach is preferable, but to achieve an acceptable level of performance it usually needs a large dictionary. **HMM** (Hidden Markov Models) is an effective probabilistic technique to treat approximate matching here, but only for a relatively small set of words and/or phrases [7]. Use of **metrics** over the set of words, is more global approach, but it leads to high complexity of time and space [2, 4, 8, 14]. A heuristic method reducing this complexity has been proposed in this paper.

2 Description of the Proposed Method

The texts for recognition are considered natural-language **names** (also short phrases or isolated words), i.e. except mistypings there may occur other kinds of errors, viz. arbitrary reduction at the end of words, arbitrary permutation of words within the name, omission of whole words, etc. So, the texts are regarded subsets of the set of words they all consist of.

The set of texts subject to recognition is denoted by $\mathcal{T}$, the set of their words – by W, and the set of word variants at the input – by V, thus:

$$T \in \mathcal{T} \equiv \{T: T \subseteq V \cup W\} \quad , \quad V \cup W \neq \varnothing \quad . \tag{1}$$

It is presupposed that the texts under consideration, $T \in \mathcal{T}$, are segmented beforehand into words they consist of (using a procedure denoted below by $\mathcal{W}$). Similarity of texts T, $T \subset V \cup W$ is supposed to be implied by similarity of their words, $W_i \in V \cup W$, i.e. $T \equiv \{W_1, W_2, \ldots W_i, \ldots W_n\}$. For a given text T, among the texts contained in a **dictionary** (denoted herein by $\mathcal{D}$), the closest one is searched. More precisely, for the words of T, the corresponding closest words from the dictionary $\mathcal{D}$ are searched.

Two basic approaches can be proposed to the task for dictionary structure searching for texts/words in it: (j) using an appropriate metric over the set $V \cup W$ and searching for the closest text in the whole set or in large its part, and (jj) by direct construction of important and small subsets (clusters) of $V \cup W$ and then searching for the closest text only in them.

The former approach is widely popular, e.g., discriminant analysis methods whose efficiency is due to the possibility of defining analytically the cluster bounds (by a metric) in the set of recognized objects. In our case of text-string objects, the metrics most often used are variants of the Damerau-Lewenstein's (DL) metric [8], known also as Wagner-Fisher's one [1, 2], as "generalized editing metric" [12], etc. Still, these metrics do not enable an efficient analytical description of cluster bounds. Their computation is based on dynamical programming [2, 8], and a linear time complexity is achieved still recently, using a finite automata technique requiring exponential complexity of space [1]. Besides, a sequential search over the whole dictionary is necessary to find the string closest to a given one [1]. For the average number ***TRecs*** of accesses to the file containing $V \cup W$ it can be noted:

$$\mathit{TRecs} \cong \tfrac{1}{2} |V \cup W| \quad . \tag{2}$$

The latter approach of significantly reduction of ***TRecs*** is considered below.

2.1 Preliminary Considerations

Each word $W \in V \cup W$ is mapped to a *modelling set* $G(W)$ of derivative words called *lexemes*, produced by the original word solely by the delete-a-character operation using a *generating procedure* G. Words $W \in G(W)$ are treated as *trivial lexemes*, i.e. experienced no change.

The set L of considered lexemes is defined as union of the modelling sets for all words in $V \cup W$. The lexemes of modelling sets for all recognizable words $W \in W$ are generated off-line and stored in a dictionary D, while the similar for the input words $V \in V$ is being done on-line. The intersection $G(C(T)) \cap G(W)$ of the modelling sets $G(C(T))$ of the words $V \in C(T)$ from the input text T with the modelling sets $G(W)$ of the words $W \in W$ from the dictionary, serves as a basis for defining the efficient **text closeness** criterion. Each element of this intersection represents a successful operation of the kind of **searching by key of a record** (for the current lexeme) **in a file** (i.e. the dictionary D). As a comparison to (2), we have:

$$TRecs \cong \sum_{V \in C(T_{input})} G(V) \quad << |V \cup W| \quad , \tag{3}$$

which explains the high performance of implementations of the proposed method.

The assignment of the model of input errors to the intersection $G(C(T)) \cap G(W)$ which represents them is made through a *goal function* F over it, due to following considerations:

- The fact of coincidence of an input lexeme with some dictionary entry, models certain kind of error whose type and position in the word can be determined (cf. Table 2) .
- The lack of coincidence, i.e. absence of an input lexeme in the dictionary D, means that either the lexeme is rather "distant" from the words in D (in the sense of a DL-metric), i.e. it would model a rare error, or that the lexeme corresponds to a word unknown for D.
- Each fact of coincidence, called a *hit* (of searching lexeme in D), contributes to the construction of the probabilistic estimation of the input word closeness to a limited set of dictionary lexemes for a hypothesis of the word, and respectively for its text.
- Totalled up, *hit-contributions* in favour of a hypothesis defines its F-score. The hypothesis with the highest F-score is considered the most correct spelling of the input text. Two remarks should be also added. If the highest score turns out to be too low, then the input text can be considered unknown to the dictionary, as it relies on adequacy in the input errors model. Besides, the presence of a large number of hypotheses with scores much close to the highest score shows a recognition-time **collision**, i.e. the input text words are not informative enough with respect to the dictionary data.

A simplified version of the proposed method (up to word level only) is considered in [3] as a possible application of the method to hand-written text recognition.

2.2 Word-Level Model of Input Errors

Words $W \in V \cup W$ are considered character strings of length k, $k=K(W)$, $k_{min} \le k \le k_{max}$, over a finite alphabet A. The bounds, k_{min} and k_{max}, $0 < k_{min} \le k_{max}$, are global parameters for $V \cup W$. The most probable errors in a word are assumed independent events, of l different kinds, enumerated in decreasing order of their probabilities considered a priori known:

$$P_i < P_0 \ , \quad |\ 1 - P_0 - \sum_{i=1}^{l} P_i\ | \approx 0 \ ; \quad 0 < P_i < 1 \ , \quad i = 0, 1, 2, \ldots l \ . \tag{4}$$

In the considered case of most probable typist errors, namely – a character substitution (P_1), omission (P_2), insertion (P_3), neighbouring reversal (P_4), etc. ($P_5 \div P_8$), according to the statistical study cited in [8], and as illustrated in Table 1 (and Figure 1), we can state:

$$l=8 \ , \quad P_0 \approx 89\% \ , \quad P_1 + P_2 + P_3 + P_4 \approx 10\% \ , \quad P_5 + P_6 + P_7 + P_8 \approx 1\% \tag{5}$$

2.3 Basic Procedures of the Method

Procedure C : (*Extraction of the input text words*). The input text is being split into words simply recognizing word separators (terminals not belonging A). Each word is fitted into the maximum length k_{max}, and the rest of characters (if any) are being cut. The limitation of k_{max} is put after a statistical estimation over the initial set of words for recognition. Each word may undergo additional "routine operations", as: ignoring words of length less than

$k_{\min}$, uppercase/lowercase conversions, Roman/Cyrillics character conversions, etc., and details here depend mostly on the application. Formally, we have:

$$C : T \to V \quad , \quad T \in \mathbf{T} \quad , \quad V \in C(T) \subset V \subseteq C(\mathbf{T}) \quad . \tag{6}$$

Procedure $\mathcal{G}$: (*Generating a modelling set of lexemes for a given word*). For each word W, $W \in \mathbf{V} \cup \mathbf{W}$ (at the input: $V \in \mathbf{V}$, and/or in the dictionary: $W \in \mathbf{W}$) , we define a **modelling set** $G(W)$ of derivative words (lexemes) as follows:

$$G(W) \equiv \{ L \mid L \Leftarrow W \} \quad , \quad W \in \mathbf{V} \cup \mathbf{W} \quad ; \tag{7}$$

where the operator $\Leftarrow$: $L \Leftarrow W$ means a production of lexeme L from a word W deleting characters in W. As the number E of deletions as their positions in W are defining parameters for every lexeme $L \in G(W)$. The parameter E which is essential for the considerations, will be called *weight* by analogy with the notion of "error weight" from the error correcting codes theory [13]. The limiting inequality $0 \le E \le k$, $k=K(W)$ is evident, but for practical reasons it will be strengthened to the form:

$$0 \le E \le E_{\max} < k \quad , \tag{8}$$

where $E_{\max} \in \{0,1,2\}$ is defined at the time of dictionary initialization, and k, $k=K(W)$ is the length of the word $W \in \mathbf{V} \cup \mathbf{W}$. The union of the modelling sets for all the considered words W, $W \in \mathbf{V} \cup \mathbf{W}$ determines the set $\mathbf{L}$ of considered lexemes, which must enter the dictionary:

$$\mathbf{L} \equiv \bigcup_{W \in \mathbf{V} \cup \mathbf{W}} G(W) \supseteq \bigcup_{W \in \mathbf{V} \equiv C(\mathbf{T})} G(W) \quad , \tag{9}$$

And, for the *generating procedure* $\mathcal{G}$, we have formally:

$$G : W \to G(W) \quad , \quad W \in \mathbf{V} \cup \mathbf{W} \quad , \quad G(W) \subseteq \mathbf{L} \quad ; \tag{10}$$

where $G(W)$ is the corresponding modelling set for the word W. Possible modifications of $\mathcal{G}$ with respect to the kind of the limitation (8) are considered below.

2.4 Types of Representing Models and their Generating Procedures

The error representation by the method largely depends on the choosen error model (4). Specific generating procedure $\mathcal{G}$ has to be defined respectively for the modelling sets $G(W)$, $W \in \mathbf{V} \cup \mathbf{W}$. We shall consider 4 different representation models of practical interest:

- Model_0 *(l=0)* := <represents the trivial lexeme that coincides with the word itself>.
- Model_A *(l=3)* := <Model_0>&<substitution, omission and insertion of a character>,

i.e. _A comprises the word itself and all the lexemes, $K(W)$ in number, of length $K(W)-1$, that can be generated from the word after deleting of a character.

- Model_B *(l=4)* := <Model_A> & <a neighbouring reversal of characters>, i.e. the set union of Model_A with all the lexemes, $K(W)-1$ in number, that can be additionally generated from the word after deleting of two neighbouring characters.
- Model_(*(l=8)* := <Model_B> & <all the rest of lexemes which can be generated deleting two characters in the word >.

Models of higher weight ($E > 2$) are not considered for practical reasons. The situation (5) corresponds to a Model_((cf. Table 1).

2.5 The Dictionary Structure

The dictionary $\mathbf{D}$ contains uniform records, one or more for each lexeme $L \in G(\mathbf{T})$, $T \in \mathbf{T}$, generated for the word-set extracted from the set of texts under recognition. Records are of fixed length, and their logical format, depending on the representation model, is as follows:

(11)

LEXKEY	N_L	***NP***	PTD_1, E_1, STD_1		PTD_i, E_i, STD_i		$PTD_{NP}, E_{NP}, STD_{NP}$	*free*

where:

LEXKEY: – primary key (to access the lexeme L), made up by concatenation of the lexeme and the ordinal number of the record for it;

N_L : – number of records for the lexeme;

NP: – number of texts generating the lexeme, i.e. reflected in this record for it;

[$\boldsymbol{PTD_i}$, $\boldsymbol{E_i}$, $\boldsymbol{STD_i}$] : – a compound field (containing data about texts T_β) where:

PTD_i : – a pointer to the generating text T_D (of the processed text file);

E_i : – the error weight (8), i.e. the number of characters deleted during the lexeme generation (respecting the model) from corresponding word of T_D ;

STD_i : – the actual length of the text T_D taken as total of lengths of its words;

i : – the ordinal number of sub-field in the compound field, $1 \le i \le NP$.

The number N_L of records per lexeme is not principally limited. For a better dictionary integrity, the following restriction is put – no record of the lexeme but the last one can be partially filled. For it, $1 \le NP \le NP_{\max}$. Otherwise, $NP = NP_{\max}$. And, $NP_{\max}$ is a constructive dictionary parameter which defines the maximum number of texts (sub-fields containing pointers to texts) that can be referred to in each record.

The questions about optimal choice of $NP_{\max}$, about multiple records for a lexeme (in overflow cases), etc., affects mostly the dictionary physical structure. Roughly, the best $NP_{\max}$ can be determined beforehand statistically, as a compromise between the records filling ratio and the average lexeme access-time. Direct (associative) access is recommended, but index access is also possible alternative for a basic access to the dictionary [10].

2.6 Modelling the Most Probable Errors

The proposed modelling of the most probable input errors for the 4 above described models (_Ø, _A , _B and _C) , listed in a descending order (4) of probabilities P_i , $i=0 \div 8$, is shown on Table 1 . Errors in a word W (of length $k=K(W)$) are represented solely by the operation of "deletion of E characters", stratified into two stages:

- on-line (algorithmically by the procedure G), where $E=E_G$, $0 \le E_G \le E_{\max}$, and
- off-line (structurally in the dictionary D), where $E=E_D$, $0 \le E_D \le E_{\max}$; $E_{\max} \in \{0,1,2\}$.

Lexemes $L \in G(V)$ generated algorithmically from the currently extracted word $V \in \Gamma(T)$ of the input text T, are searched for in the dictionary D. A **hit** would model a possible error of type (E_G, E_D) somewhere in the word V , where E_G is the number of deleted characters in V, and E_D whose meaning is the same as E_G but for the lexeme of hit, is taken from D . It is the error type that is processed directly in the four models in use, rather than the error position (in the input word V) that is obtained here as a result. Every hit gives an **additive contribution** $\omega(E_G, E_D)$ to the probability evaluation of modelled errors. Lack of hit is treated as contributing nothing. Each of the 4 models is an extension of the previous ones (cf. Table 1), and is specified by:

- The set of the most probable errors it represents;
- The number of dictionary accesses per word – it is determined by the number $|G(V)|$,

Model	Maximal accesses number $\lvert G(V) \rvert$	Errors for $V \in \Gamma$: Kind E_G E_D	Most probable input error description	Other errors	# i	Total hits number (vs. the model) $\alpha(V)$: C_	B_	A_	Ø_
Ø_	*1*	0 0	No errors	-	0	*1+k+ k(k-1)/2*	*2k*	*1+k*	*1*
A_	*1+k*	1 1	Substitution <1S>	<1O + 1I>	1	*k*	*3*	*1*	
		0 1	Omission <1O>	-	2	*k+1*	*3*	*1*	
		1 0	Insertion <1I>	-	3	*k*	*3*	*1*	
B_	*2k*	2 2	Neighbour reversal <1N>	<2S> <1S+1O+1I>	4	*2k-1*	*3*		
* C_	*1+k + k(k-1)/2*	2 2	<2S>	<2O+2I>	* 4				
		2 2	<1S + 1O>	<2O+1I>	5				
		2 2	<1S + 1O>	<2I + 1O>	6				
		2 2	<2O>	-	7				
		2 2	<2I>	-	8				

(*) In Model_B and _C , the errors of type <1N> are much more probable than errors of type <2S>. In this case they are merely formally separated in the table to show differences in the access number.

Table 1: Errors representation models of the method

of algorithmically generated lexemes per input word $V \in \mathbb{V}$, i.e. ~k for models _A and _B, and ~k^2 for model _C, where $k=K(V)$ is the length of V;

- The number $\alpha(V)$ of hits in $\mathbb{D}$ per word. Normally, $\alpha(V) \le |\mathbb{G}(V)|$, but in case of close words in $\mathbb{D}$, often $\alpha(V) \ge |\mathbb{G}(V)|$;
- The volume of $\mathbb{D}$ – it is proportional to the average value of $|\mathbb{G}(W)|$, $W \in \mathbb{W}$, and hence to the number $|\mathbb{W}|$ of represented words.

2.7 An Illustrative Example

The set $\mathbb{G}(W)$ of modelling lexemes which are written in $\mathbb{D}$ for each of the models mentioned, is illustrated on Table 2 by means of the word $W \equiv$ <**think**>. The hits in the dictionary are indicated for the same word $V \equiv W$ and for 5 more illustrative versions of it. The formulae are shown for hit number $\alpha(V)$ separately for each error type.

Hits' kind in $\mathbb{D}$			Modeling sets $\mathbb{G}(V)$ of the input words examples (V) ...					
Modeling set $\mathbb{G}(W)$			ø_	A_			B_	C_
W = "think"	$L \in \mathbb{L}$	$K(L)$	think	th*a*nk	thik	thi*i* nk*	*ht*ink	t*ni*hk*
ø_	think	5	(0,0)	-	-	(1,0) AA	-	-
A_	_hink	4	(1,1)	-	-	(2,1) CC	(1,1)	-
	t_ink	4	(1,1)	-	-	(2,1) CB	(1,1)	-
	th_nk	4	(1,1)	(1,1)	-	(2,1) B	-	-
	thi_k	4	(1,1)	-	(0,1)	(2,1) BC	-	-
	thin_	4	(1,1)	-	-	(2,1) CC	-	-
B_	__ink	3	(2,2)	-	-	-	(2,2)	-
	t__nk	3	(2,2)	(2,2)	-	-	(2,2) C	(2,1) *
	th__k	3	(2,2)	(2,2)	(1,2)	-	-	(2,1) *
	thi__	3	(2,2)	-	(1,2)	-	-	-
C_	_h_nk	3	(2,2)	(2,2)	-	-	(2,2)	
	_hi_k	3	(2,2)	-	(1,2)	-	(2,2)	-
	hin	3	(2,2)	-	-	-	(2,2)	-
	t_i_k	3	(2,2)	-	(1,2)	-	(2,2)	(2,2)
	t_in_	3	(2,2)	-	-	-	(2,2)	-
	th_n_	3	(2,2)	(2,2)	-	-	-	-
Err.	Input kind		no err.	<1S>	<1O>	<1I>	<1N>	<2S>
	(E_G+o,E_D+o)		(0,0)	(1,1)	(0,1)	(1,0)	(1,1)	(2,2)
Length $k=K(V)$			5	5	4	6	5	5
Number of hits by their kind	Model	$\omega(E,E)$						
	ø_ (E_G+o,E_D+o)		*1*					
		$\alpha(W)$	*1*					
	A_ (E_G+o,E_D+o)		*1*	*1*	*1*	*1* (2)		
	A_ (E_G+1,E_D+1)		*k*	-	-	*0*		
		$\alpha(W)$	*k+1*	*1*	*1*	*1*		
	B_ (E_G+o,E_D+o)		*1*	*1*	*1*	*1* (2)	*2*	
	B_ (E_G+1,E_D+1)		*k*	*2*	*2*	*2* (3)	*1*	
	B_ (E_G+2,E_D+2)		*k-1*	-	-	*0*	-	
		$\alpha(W)$	*2k*	*3*	*3*	*3*	*3*	
	C_ (E_G+o,E_D+o)		*1*	*1*	*1*	*1* (2)	*2*	*1* (3)
	C_ (E_G+1,E_D+1)		*k*	*k-1*	*k*	*k-1* (2k-3)	*2k-3*	-
	C_ (E_G+2,E_D+2)		*k(k-1)/2*	-	-	*0*	-	-
		$\alpha(W)$	*1+k+k(k-1)/2*	*k*	*k+1*	*k*	*2k-1*	*1*

(*) An input error specificity is chosen in the case.

Table 2: How the models work on the example of the word "**think**"

2.8 Selected Model Setting-Up

The setting-up can be performed by ad hoc calculation of the contributions $\omega(E_G, E_D)$, $E_G \in \{0,1,2\}$, $E_D \in \{0,1,2\}$, with respect to the given distribution (P_i, $i=0 \div l$) of input errors (4) expected for an average length k of the words $W \in \mathbb{W}$.

For example, according to Table 2, we obtain the following linear equations system for Model_(, ($l=8$, $E_{\max}=2$):

$$\left\|\begin{array}{lll} \omega(0,0) + k\,\omega(1,1) + \frac{1}{2}k(k-1)\,\omega(2,2) = P_0 & , & \text{<no err.>} \\ \omega(1,1) + (k-1)\,\omega(2,2) = P_1 & , & \text{<1S>} \\ \omega(0,1) + k\,\omega(1,2) = P_2 & , & \text{<1O>} \\ \omega(1,0) + (k-1)\,\omega(2,1) = P_3 & , & \text{<1I>} \\ 2\,\omega(1,1) + (2k-3)\,\omega(2,2) = P_4 & , & \text{<1N>} \\ \omega(2,2) = P_{4*} & , & \text{<2S>*} \\ \omega(1,2) = P_5, \quad \omega(2,1) = P_6, \quad \omega(0,2) = P_7, \quad \omega(2,0) = P_8 & ; & \text{<other err.>} \end{array}\right. \qquad (12)$$

where $k = K(W)$, $k_{\min} \le k \le k_{\max}$. After transformations, we obtain for the contributions ω:

$$\left\|\begin{array}{ll} \omega(0,0) = P_0 - (2P_1 - \frac{1}{2}P_4)k + (P_1 - \frac{1}{2}P_4)\,k^2 & , \\ \omega(1,1) = (3P_1 - P_4) - (2P_1 - P_4)\,k & , \\ \omega(0,1) = P_2 - P_5\,k & , \\ \omega(1,0) = (P_3 + P_6) - P_6\,k & , \\ \omega(2,2) = P_{4*} = 2P_1 - P_4 & , \\ \omega(1,2) = P_5, \quad \omega(2,1) = P_6, \quad \omega(0,2) = P_7, \quad \omega(2,0) = P_8 & ; \end{array}\right. \qquad (13)$$

where $k = K(W)$, $k_{\min} \le k \le k_{\max}$, and P_{4*} is a working parameter (cf. Table 1).

Evidently by (13), the contributions essentially depend on k. It can be shown that (13) is also correct in statistical, i.e. for an average length k_{mean}, over the whole set $\mathbb{W}$ of words in $\mathbb{D}$. As for an initial setting-up as for on-line tunings of the chosen model, the contribution mean values can be evaluated by (13) but through the substitutions:

$$k := k_{\text{mean}} = \sum_{k=k_{\min}}^{k_{\max}} k\,P_W(k) \quad , \qquad k^2 := (k^2)_{\text{mea}} = \sum_{k=k_{\min}}^{k_{\max}} k^2\,P_W(k) \quad ; \qquad (14)$$

where $P_W(k)$, $k_{\min} \le k \le k_{\max}$ is the word distribution in $\mathbb{D}$, a priori estimated. In a similar way (cf. Tables 1 and 2), the contributions ω can be as evaluated with respect to an other choice of model (_Ø , _A and _B).

2.9 Closeness Goal Function and Decision Rule for Recognition

For each model of dictionary, an input error is considered *recognizable* if the result of dictionary search is a hit. The goal function $F: \mathbb{T} \to [0,1)$ is defined as probabilistic estimation of closeness to the input text T_{inp} for all dictionary texts $T_D \in \mathbb{T}_T \subset \mathbb{T}$, for which at least one recognizable error is hit, and consequently a non-zero contribution ω is added to $F(T_D)$. For the rest of dictionary texts $T_D \in \mathbb{T} \setminus \mathbb{T}_T$, $F(T_D)$ is assigned to zero. Formally:

$$\left\|\begin{array}{ll} F(T_D) = e_1\, I_T(T_{\text{inp}}, T_D) \sum_{\substack{V \in \mathbb{C}(T_{\text{inp}}) \\ k = K(V)}} P_W(k) \sum_{\substack{L_G \in \mathbb{G}(V) \\ L_G \equiv L_D}} \omega(E_G, E_D)\, I_L(L_G, L_D) \quad , & T_D \in \mathbb{T}_T \quad ; \\ F(T_D) \equiv 0 \quad , & T_D \in \mathbb{T} \setminus \mathbb{T}_T \quad ; \end{array}\right. \qquad (15)$$

where:

$\mathcal{T}_T$, $\mathcal{T}_T \subset \mathcal{T}$ – the subset of texts T_D (represented in the dictionary D) which are candidates to be closest to the input text T_{inp} ;

$\omega(E_G, E_D)$ – the contribution of lexeme $L_G \in \mathcal{G}(V)$ generated for the current word $V \in \mathcal{V}(T)$ of the input text T_{inp} . In a case of hit, the value $\omega(E_G, E_D)$, $E_G, E_D \in \{0,1,2\}$, is determined by (13), where:

E_G is the number of deletions during the lexeme L_G generation (available by G),

E_D – the same for lexeme L_D, and available by the dictionary record (11) for L_D ;

$P_W(k)$ – the probability of word $W \equiv V$, a priori or currently estimated, cf. (14) ;

e_1 – a normalizing factor over the interval [0,1) of goal function values.

$I_T(T_{\text{inp}}, T_D)$ and $I_L(L_G, L_D)$ are heuristic factors normalizing the goal function value by the correspondent informational content of texts and their lexemes. More details about evaluation of the normalizing factors can be found in [4].

Obviously, the set $\mathcal{T}_T$ of candidates T_D to be the closest to the input text T_{inp} is non-empty if the errors in the input words $V \in \mathcal{V}$ are among those provided for in the dictionary (for the selected model). Besides, texts T_D, $T_D \in \mathcal{T}_T \subset \mathcal{T}$, can be ordered decreasingly by their F-scores. So, a decision rule for recognition of T_{inp} can be defined as the maximum criterion:

$$T_{D\,\text{closest}} = \text{Arg}\left(\max_{T_D \in \mathcal{T}_T} (F(T_D)) \right) \quad , \tag{16}$$

where $T_{D\,\text{closest}}$ is the text closest to T_{inp} among the texts T_D represented in the dictionary. And, the power of $\mathcal{T}_T$ is not significant as compared to the whole set $\mathcal{T}$ of dictionary texts:

$$|\mathcal{T}_T| \;\le\; \sum_{V \in \mathcal{C}(T)} |\mathcal{G}(V)| \sum_{\substack{L_G \in \mathcal{G}(V) \\ L_G \equiv L_D}} M(L_D) \;\ll\; |\mathcal{T}| \quad , \tag{17}$$

what accounts for high performance of the method, both in searching lexemes $L \in \mathcal{G}(\mathcal{V}(T_{\text{inp}}))$ of the input text T_{inp} , and in sorting the set $\mathcal{T}_T$ in order to find the text $T_{D\text{closest}}$.

4 Experiments

The proposed method has been implemented in a software package named *Fault-Tolerant Text Key-Access Method* (**FTKAM**). The system is autonomous and flexible to various applications. The main part of FTKAM is realised as a software extension over the basic data-access method particularly used by the DBMS of interest. Up to now, the FTKAM system is developed to serve two particular databases, namely: (i) a Geographical Information System (**GIS**) , and (ii) an English-Bulgarian computer Dictionary (**EBD**).

4.1 About Experiments themselves

The above-mentioned GIS, entitled *Street network for the capital city of Sofia*, is described in a more detail in [5]. The texts under recognition are the names of the principal streets of the city – 3278 names containing a total of 6492 words (whereof 1169 entirely digital). The length of each name (text key) is up to 30 characters (1÷5 words). The employed model is Model_B. The dictionary structure parameter $\boldsymbol{NP}_{\max}$ is set as a compromise to $\boldsymbol{NP}_{\max}$=10, after a preliminary statistical evaluation. The maximum length of words is between $k_{\min}$= 3 and $k_{\max}$= 9 , for which the number of lexemes generated and stored in the dictionary is 28461, allocated in 29370 records of type (11), while the total power of the modelling sets is 52294 (i.e. the number of pointers to the 3278 recognizable names).

The EBD system was especially developed to verify the efficiency of the proposed method over a text file by times larger than one of GIS case. The texts under recognition here are 22009 English words (and short phrases) representing the contents of a typical English-Bulgarian dictionary like [6].

4.2 A Comparative Analysis of Results

Equal computer and software environment has been supplied for both the experiments to perform a comparative analysis of results: an IBM/486/ DX4/100MHz computer with a HDD 540MB having an average seek time t_{disk} ≈12ms ; the FTKAM software differs only in the user-interactive features; Model_B for the dictionary with NP_{max}=10; a consecutive organization (no fragmentation) for the text and dictionary files.

More important results are shown on Fig. 1. With a view of the minimum of dictionary volume, the choice of the constructive parameter NP_{max} , cf. (11), would yield NP_{max}=2 . The result would be the same if the choice were made by minimum of the relative prise of using the method ***Price***(NP_{max}), while by the minimum of the estimated access-time per name ***t_NR***(NP_{max}), we would get $NP_{max} \rightarrow m_{max}$ but for unacceptably high values for ***Price***. The choice NP_{max}=10 resulted in ***Price***(10) ≈ 20:1 , i.e. in a relatively low dictionary volume usage factor ***Mju***(NP_{max}) ≈ 32,3% ; but, on the other hand, it allowed for an acceptable average time ***t_N***$(NP_{max}=10)$ per name access. The experimentally measured mean access-time ***t_N*** per name are: ***t_N***(10) ≈113 ms, ***t_W***(10) ≈70 ms , ***t_L***(10) ≈2,3 ms (for GIS) ;

and ***t_N***(10) ≈171 ms, ***t_W***(10) ≈163 ms, ***t_L***(10) ≈2,0 ms (for EBD).

The last can be counted as affordable from an ergonomic viewpoint (viz. to the requirements for an user-friendly information-retrieval system [11]). More details about the comparative analysis can be found in [4].

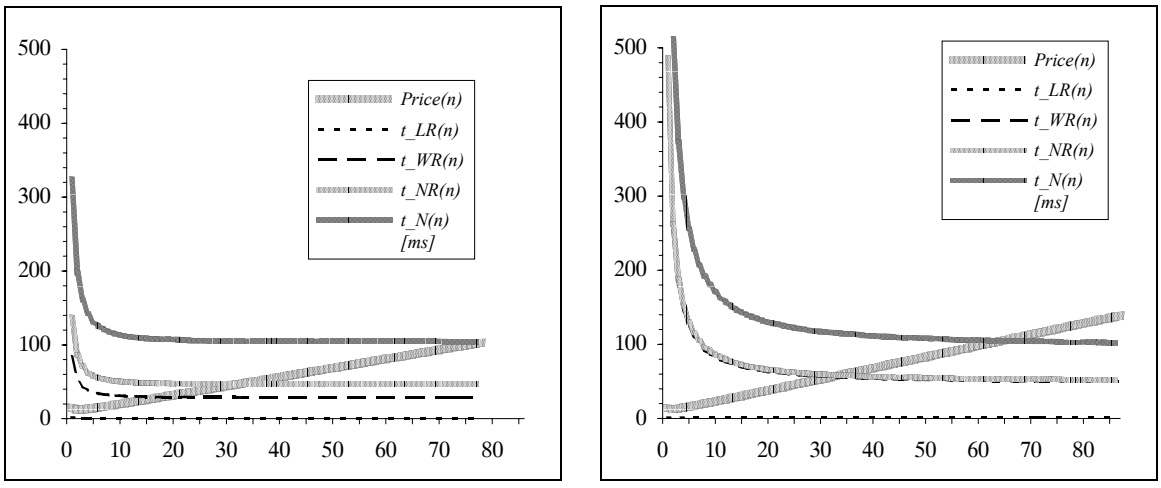

Figure 3: Graphics of the dictionary access-time estimations: for the relative times (measured in numbers of records (11)) – ***t_LR***(n) , ***t_WR***(n) and ***t_NR***(n) as well as for the absolute access-time ***t_N***(n) (measured in ms), as functions of the dictionary parameter choice: $n=NP_{max}$, $1 \le n \le m_{max}$:

(On the left) for the GIS exp. ($m_{max}=78$) ; (On the right) for the EBD exp. ($m_{max}=308$)

5 Conclusion

A constructive method is proposed for an algorithmic and structural representation of the set of strings over a given finite alphabet, taking into account a priori statistics on the most probable differences between similar strings. The method can be considered as an alternative to the known methods for the so-called "approximate string matching", which make use of the Damerau-Lewenstein metric [8] defined on the set of strings (this metric is also known as Wagner-Fisher's metric [1, 2], generalized editing metric [12], etc.). A similar metric existence is presupposed by the proposed method rather than directly used, i.e. the set of strings is treated by a principal recently known as "clustering without metrics". Similarity between strings is defined constructively through a probabilistic model of the significant subsets of strings, taking into consideration the a priori statistical data for closeness between strings. This approach of a "topological" construction of clusters of strings closest to a given string, allows the proposed method adaptation to wide spectrum of

applications concerning the approximate string matching problem. Important features of the method are:

- Good adaptivity to various applications: Four step-by-step extending models are defined (model_Ø, _A, _B and _( ) for correction of classes of most common mistypings.
- Possibility for model set-up and current training: The goal function contributions (15), can be recalculated (13) in case of changes in the model of the most probable input errors (4). The same is valid for the word statistics in updating the dictionary (12÷14).
- Relatively good performance: Additional memory requirements for the dictionary maintenance are relatively low: except on the number of words, these depend linearly (in Models_A and _B) and quadratically (in Model_() on the average word length, as only the delete-a-character operation is used for representations in dictionary. The search time is also proportional to the number of words in the input text, and it is practically independent from the total of words or texts referred to in the dictionary.
- General disadvantage: Errors of weight more than 2, cf. (8), are not effective for a direct presentation by the method. Splitting long words and processing their parts can be a possible extension of the method in those cases of high error weights.

The method is developed to a software implementation – a fault-tolerant direct access by a text key to records of a conventional database, but it can be also used for the needs of text processing. It can also be applied successfully to image recognition and more precisely to graphics recognition [3], in cases where recognition methods under research can require string matching, e.g. to handwritten (and/or printed) texts recognition [2, 3, 14]. Adding more complex structures, like statistical data, syntactic/semantic rules, etc. is not principally limited in an upgrade of the method. It looks a good prospect to combine the method with the advantages of, e.g., a HMM technique representing errors in long words.

References

[1] Bunke H.: A Fast algorithm for Finding the Nearest Neighbour of a Word in a Dictionary. In Proc. of ICDAR'93 (author's preprint is available only).

[2] Bunke H.: Recent Advances in String Matching. In H. Bunke (ed.) Advances in Structural and Syntactic Pattern Recognition. Proceedings, Bern, 1992. Series in MP&AI, Vol. 5, 3-21, London: World Scientific.

[3] Dimov D. T.: An Approximate String Matching Method for Hand-writing Recognition Post-Processing Using a Dictionary. In Fundamentals in HWR, S. Impedovo (Ed.) NATO ASI Series "F": "C&S Sci", Vol. 124, Berlin: Springer-Verlag, 1994, 323-332.

[4] Dimov D. T. Fault-tolerant file key-access method for the most probable typist errors. In Working papers serie of IIT – BAS, ISSN-1310-652X, WP/13, December 1995, 40p.

[5] Dimov D. T., and Hr. Radev: A Distributed Graphic Visualization System in a Computer Network. In: D. I. Mladenov (ed.), Distributed Intelligence Systems – Methods and Applications. IFAC Proceed. Series: Pergamon Press, No. 4, 1989, 273-278.

[6] English-Bulgarian Dictionary, in R. Russev (ed.), Reading, Sofia, Bulgaria: "NARODNA PROSVETA" Publ. house, 1965, 495p.

[7] Gilloux M.: Hidden Markov Models in Handwriting Recognition. In Fundamentals in HWR, S. Impedovo (Ed.) NATO ASI Series "F": "C&S Sci", Vol. 124, Berlin: Springer-Verlag, 1994, 265-288.

[8] Hall P. A. V., and G. R. Dowling: Approximate String Matching. ACM Computing Surveys J., Vol. 12, No. 4, 381-402 (1980).

[9] Hull J. J.: A Database for Handwritten Text Recognition Research. IEEE Trans. on PAMI, Vol.16, No.5, 550-554 (1994).

[10] Knuth D. E.: Fundamental Algorithms, 2-d Edition. The Art of Computer Programming, Vol.1, Reading, MA: Addison-Wesley, 1973 (Russian transl. "MIR", Moscow, 1976).

[11] Ozkarahan E.: Database machines and Database Management. Prentice-Hall, Inc., Englewood Gliffs, 1986 (Russian transl. "MIR", Moscow, 1989).

[12] Paquet T., and Y. Lecourtier: Recognition of Handwritten Sentences Using a Restricted Lexicon. Pattern Recognition J., Vol. 26, No. 3, 391-407 (1993).

[13] Peterson W. W., and E. J. Weldon Jr.: Error Correcting Codes. The MIT Press, Cambridge, MA, 1972 (Russian transl. "MIR", Moscow, 1976).

[14] Weigel A.: Using Approximate String Matching for Handwriting Recognition. The 8th Scandinavian Conf. on Image Analysis, May 1993. Proceedings, Vol. 1, 1993, 397-404.