Efficient Multilingual Phoneme-to-Grapheme Conversion Based on HMM

Panagiotis A. Rentzepopoulos*
University of Patras

George K. Kokkinakis*
University of Patras

Grapheme-to-phoneme conversion (GTPC) has been achieved in most European languages by dictionary look-up or using rules. The application of these methods, however, in the reverse process, (i.e., in phoneme-to-grapheme conversion [PTGC]) creates serious problems, especially in inflectionally rich languages. In this paper the PTGC problem is approached from a completely different point of view. Instead of rules or a dictionary, the statistics of language connecting pronunciation to spelling are exploited. The novelty lies in modeling the natural language intraword features using the theory of hidden Markov models (HMM) and performing the conversion using the Viterbi algorithm. The PTGC system has been established and tested on various multilingual corpora. Initially, the first-order HMM and the common Viterbi algorithm were used to obtain a single transcription for each word. Afterwards, the second-order HMM and the N-best algorithm adapted to PTGC were implemented to provide one or more transcriptions for each word input (homophones). This system gave an average score of more than 99% correctly transcribed words (overall success in the first four candidates) for most of the seven languages it was tested on (Dutch, English, French, German, Greek, Italian, and Spanish). The system can be adapted to almost any language with little effort and can be implemented in hardware to serve in real-time speech recognition systems.

1. Introduction

Phoneme-based speech recognition systems incorporate a phoneme-to-grapheme conversion (PTGC) module to produce orthographically correct output. Many approaches have been used, most of which compare the phonemic strings to a (usually application-specific) dictionary containing both the phonemic and the graphemic form of every word the system can handle (Laface, Micca, and Pieraccini 1987; Levinson et al. 1989, etc.). Considering the effort and cost required to create such a dictionary, this is a serious limitation, especially for inflectionally rich languages such as Greek and German. Another very important issue when searching for words in a dictionary is the number of candidates resulting from each phonemic input. Depending on the language and the errors of the recognizer, this number may be very large, rendering the disambiguation of the words by a subsequent language model a time-consuming and unreliable task.

The domain of application is another factor that strongly influences conversion performance; a general dictionary can omit the specialized words of specific domains (e.g., legal, engineering, or medical terminology) and vice versa. Finally, applications that must handle a large number of proper names (e.g., directory service applications) generally cannot include all the possible names. The only remedy in such situations

* Wire Communication Laboratory, University of Patras, Patras, GR 26500 Greece

would be to increase the size of the reference dictionary, so that every possible input word is included. A final consideration is the type of errors a dictionary-based PTGC system introduces when it encounters a word that is not contained in the dictionary: the system will produce the closest existing word (in its dictionary) as the best candidate, which may give a completely incomprehensible (if not wrong) meaning to the input phrase.

Another approach to phoneme-to-grapheme conversion is the use of linguistic and/or heuristic rules (Kerkhoff and Wester 1987). This method works on a phoneme or syllable basis and can give adequate results in languages where the spelling is very similar to the pronunciation (such as Italian). Nevertheless, languages with diphthongs or double letters cannot benefit from this method, since it creates long lists of homophonic candidates that are all correct (in the sense that they are pronounced as the input word) but that do not exist in the language. In Greek, for example, where the phoneme /i/ is the sound of five different graphemes (η, ι, υ, $\epsilon\iota$, $o\iota$) and the phoneme /l/ can come from λ and $\lambda\lambda$, the phonemic form /m'ila/ would produce a list containing the following 10 transcriptions: $\mu\acute{\iota}\lambda\alpha$, $\mu\acute{\eta}\lambda\alpha$, $\mu\acute{\upsilon}\lambda\alpha$, $\mu\epsilon\acute{\iota}\lambda\alpha$, $\mu o\acute{\iota}\lambda\alpha$, $\mu\acute{\iota}\lambda\lambda\alpha$, $\mu\acute{\eta}\lambda\lambda\alpha$, $\mu\acute{\upsilon}\lambda\lambda\alpha$, $\mu\epsilon\acute{\iota}\lambda\alpha$, and $\mu o\acute{\iota}\lambda\lambda\alpha$ all having the same pronunciation. From this list, only two represent existing orthographically correct words; "$\mu\acute{\iota}\lambda\alpha$" 'speak!' and "$\mu\acute{\eta}\lambda\alpha$" 'apples.' Previous work has shown that an average of 30 graphemic candidates is produced by this transcription for every input phonemic word (Rentzepopoulos 1988).

To overcome the disadvantages of the above mentioned methods, a novel statistical approach to the problem of PTGC, which is based on hidden Markov models (HMM), has been investigated and is presented in this paper. Although statistical approaches have already been widely applied in several fields of natural language processing, they have not been considered for PTGC. The proposed method is language independent, does not use a dictionary, and can be applied with only minimal linguistic knowledge, thus reducing the cost of system development. Initially, the first-order HMM and the common Viterbi algorithm were used to provide a simple transcription for each input word. In its current version, the method is based on second-order HMM and on a modified Viterbi algorithm, which can provide more than one graphemic output for each phonemic input, in descending order of probability. The multiple outputs make it possible to apply a language model in sentence level for disambiguation at a subsequent stage. This version of the algorithm raised the number of correctly transcribed phonemes to 97%–100% for most of the languages the system was tested on. The proposed system assumes that the word boundaries are known; that is, it is a subsequent stage in an isolated-word speech recognition system. The PTGC method can work as a stand-alone module or in co-operation with a look-up module with a small to moderate size dictionary containing the most common words of the language. In the latter case, the look-up module employs a distance threshold: when the difference between the input and the words in the dictionary is greater than this threshold, control is passed to the HMM system, which converts the input phoneme string to graphemes.

The basic theory, the pilot implementation, and the proposed final system are presented in Section 2. The evaluation procedure and the error-measure methodology are described in Section 3. In Section 4, the experimental results of the system are presented and the nature of the errors is discussed. The multilingual aspects of the algorithm and experimental results for seven languages are also given in this section. Finally, some conclusions are drawn about the system and topics for further research and hardware implementation are discussed in Section 5.

2. Description of the System

Before the presentation of the proposed system, a brief overview of the theory used and the issues addressed in its application are given. These include the basic hidden Markov model theory, the Viterbi algorithm, the N-best algorithm and the solutions used to make the PTGC system fast and efficient, adequate for real-time applications.

2.1 The First Order Hidden Markov Model

An HMM can model any real-world process that changes states in time, provided that the state changes are more or less time independent (Hannaford and Lee 1990; Rabiner 1989). An HMM is used to describe statistical phenomena that can be considered sequences of hidden (i.e., not directly observable) states that produce observable symbols (Lee 1989). These phenomena are called **hidden Markov processes**. A hidden Markov process is described by a model λ that consists of three matrices A, B, and π. Matrix A contains the transition probabilities of the hidden states, matrix B contains the probability of occurrence of an observation symbol given the hidden state, and vector π contains the initial probabilities of the hidden state. In mathematical terms:

$$A = \{\alpha_{ij} : i = 1 \ldots N, j = 1 \ldots N\}, \quad \alpha_{ij} = P(q_t = S_j \mid q_{t-1} = S_i) \tag{1}$$

$$B = \{\beta_j(m) : j = 1 \ldots N, m = 1 \ldots M\}, \quad \beta_j(m) = P(O_t = v_m \mid q_t = S_j) \tag{2}$$

$$\pi = \{\pi_i : i = 1 \ldots N\}, \pi_i = P(q_1 = S_i) \tag{3}$$

where N is the number of possible hidden states and M is the number of all the observable events. Obviously the dimension of matrix A is $N \times N$, that of matrix B is $N \times M$, and π is a vector of N elements.

In equations (1)-(3), q_t is the hidden state of the system at time t, S_i is the i^{th} possible hidden state of the system, O_t is the observation symbol at time t, and v_m is the m^{th} possible observable symbol.

For the application of HMM theory to PTGC, the correspondence of the natural language intraword features to an HMM can be found on the following basis:

> The natural language pronunciation can be considered as the output (observation) of a system that uses as input (hidden state sequence) the spelling of the language (Rentzepopoulos, Tsopanoglou, and Kokkinakis 1991).

In this formulation, the sequence of phonemes produced by the system can be seen as the observation-symbol sequence of an HMM that uses the graphemic forms as a hidden-state sequence. With this statement, the PTGC problem can be restated as follows:

> Given the observation-symbol sequence $O(t)$ (phonemes) and the HMM λ, find the hidden-state sequence $Q(t)$ (graphemes) that maximizes the probability $P(O \mid Q, \lambda)$.

A formal technique for finding the single best state sequence is based on dynamic programming and is the well-known Viterbi algorithm (Forney 1973; Viterbi 1967).

In a word-level implementation, the algorithm must find the hidden-state sequence (i.e., word in its orthographic form) with the best score, given the model λ and the observation sequence O (i.e., word in its phonemic form). This algorithm proceeds

recursively from the beginning to the end of the word calculating for any time (in the case of PTGC, time is the position of a phoneme/grapheme in the word) the score of the best path in all possible hidden-state sequences that end at the current state.

The model's parameters can be estimated using the definition formulas, since both the hidden-state and the observation-symbol sequences are known during the training phase of the conversion system. Thus there is no need of a special estimation procedure like the Baum-Welsh algorithm (Rabiner 1989), which is used when the hidden-state sequence is not known. In general:

$$a_{ij} = \frac{n'(q_{t-1} = S_i, q_t = S_j)}{n(q_t = S_j)} \tag{4}$$

$$b_j(m) = \frac{n'(q_t = S_j, O_t = v_m)}{n(q_t = S_j)} \tag{5}$$

$$\pi_i = \frac{n'(q_1 = S_i)}{n(q_1)} \tag{6}$$

where $n(x)$ is the number of occurrences of x in the training corpus and $n'(x)$ is an estimation of the number of occurrences of x in the application corpus. The size of the training corpus and the sparseness of the resulting matrices can lead to different approaches in the definition of the estimation function $n'(x)$. If a reasonably large text is available for training, then $n'(x) \cong n(x)$. On the other hand, if the training data are insufficient (something that would result in a very sparse transition matrix) then a smoothing technique should be used for the estimation function $n'(x)$ (Katz 1987; Ney and Essen 1991).

2.2 Pilot System

To implement the above algorithm in PTGC, some decisions had to be made about the states, observation symbols, and transition probabilities. These decisions are listed below.

a. Every hidden state should produce one observation symbol. To achieve this, all the possible graphemic transcriptions of phonemes were coded as separate graphemic symbols (e.g., π and $\pi\pi$ are two different graphemic symbols even though they are both pronounced /p/).

b. The transition probability matrix (A) should be biased to contain at least one occurrence for every transition and no zero elements.

Consider first (a). According to the physical meaning given to the hidden states and the observation symbols of the HMM used, there cannot be hidden states (graphemes) that do not produce an observable symbol (phoneme). This is only partially correct for natural languages including mute letters and diphthongs. To overcome this problem, the hidden-state alphabet and the observation-symbol alphabet should contain not only single characters (single graphemes or phonemes respectively) but also clusters. This way, it is guaranteed that there will be no case where a sequence of graphemes produces a sequence of phonemes of a different length. The rules for the segmentation of a phoneme string to a sequence of symbols conforming to the above condition are manually defined off-line according to the procedure presented below in an informal algorithmic language (Figure 1).

```
Let G = {g1, g2, ... gM} be the set of phonemes and
    P = {p1, p2, ... pN} the set of graphemes of the language

repeat
    Pt = Z
    for i=1 to M
        Pt = Pt ∪ phoneme_transcriptions_of (gi)
    end for
    P = P ∪ Pt
    Gt = Z
    for i=1 to N
        Gt = Gt ∪ grapheme_transcriptions_of (pi)
    end for
    G = G ∪ Gt
until (Pt == Z and Gt == Z)
```

Figure 1
Segmentation rule development algorithm.

The meaning of this algorithm is the following: If a pair of phonemes is written as either a single grapheme or a pair of graphemes, then this pair is considered a single state. The same holds for the reverse procedure when a pair of graphemes is pronounced as either a single phoneme or a pair of phonemes. For example (in Greek):

grapheme ξ is pronounced /ks/ e.g., *ξύδι*-ksídi 'vinegar'

grapheme κ is pronounced /k/ e.g., *καλό*-kaló 'good'

grapheme σ is pronounced /s/ e.g., *σαφή*-safí 'lucid'

graphemes κσ are pronounced /ks/ e.g., *έκστασση*-ékstasi 'ecstasy'

In this example the pair of phonemes /ks/ is considered a single phonemic symbol. Accordingly, the pair "κσ" is also considered a single graphemic state since it is pronounced as /ks/. As can be seen, in order to disambiguate the case of κσ the phonemic symbol /ks/ and the graphemic state κσ must be introduced.

This algorithm is the only language-specific part of the PTGC system and its formulation requires only familiarity with the spelling of the language and not sophisticated linguistic knowledge. The rules are incorporated in the PTGC system using an automated procedure as a separate input function that parses the input strings into states.

Now consider (b), concerning the transition probability matrix. Matrix A is established according to formula (4) through training in appropriate corpora using as $n'(x) = \max(n(x), 1)$. The bias described in (b) is necessary so that the algorithm does not discard a new transition but instead assigns a bad score to it (Rabiner 1989). The bias is one occurrence for each transition that has never occurred in the training corpus

and the model is normalized so that it fulfills the statistical constraints, i.e.:

$$\alpha_{ij} \geq 0, \quad \sum_i \alpha_{ij} = 1 \tag{7}$$

This estimation is allowed since the training corpora are reasonably large and the bias of one occurrence per transition has no significant effect on the validity of the actually nonzero matrix elements.

Initially, a system based on a first-order HMM was implemented, and the results of its evaluation, detailed in Section 3, were promising. For Greek, this system gave an average score of 78% correctly transcribed words, while at the phoneme level the score reached 95.5% (Rentzepopoulos and Kokkinakis 1991). Similar rates were achieved in four other languages (English, French, German, and Italian) (Rentzepopoulos and Kokkinakis 1992).

The model implemented as above showed some disadvantages:

- It did not have enough detail.
- It could not produce more than one solution (homophones).

Therefore, a higher-order HMM and a multiple-output conversion algorithm were employed in order to overcome these disadvantages and achieve better results.

2.3 Second Order HMM

Since the results of the first order HMM system were encouraging, we decided to develop an improved version of the system. Two areas were selected for possible advancement: first, to make the system contain more detail in the modeling of the language, and second, to use a system that could produce more than one output solution for each phonemic input (homophones). This would offer a choice between alternatives, making it possible to find the best solution at a following stage.

The first improvement was accomplished using a second-order HMM. This is a model that contains conditional probabilities of the form:

$$\alpha_{ijk} = P(q_t = S_k \mid q_{t-1} = S_j, q_{t-2} = S_i) \tag{8}$$

i.e., the probability of occurrence of state S_k when the two previous states are S_i and S_j at $t-2$ and $t-1$, respectively. The complete model needs a new matrix of conditional probabilities that contains the probability of state-pairs in time $t = \{1, 2\}$:

$$\rho = \{\rho_{ij} : i = 1 \ldots N, j = 1 \ldots N\}, \quad \rho_{ij} = P(q_1 = S_i \mid q_2 = S_j) \tag{9}$$

So the complete model λ consists of $\{A, B, \pi, \rho\}$. The second-order HMM can be translated into a first-order HMM with an extended state space, in which state pairs are used as single states.

To use the above model, a new version of the Viterbi algorithm should be employed, one which can recursively calculate the intermediate values of the probability measure d using the second-order HMM. A second-order HMM has been introduced before (Kriouile, Mari, and Haton 1990) for other problems in the field of pattern analysis and speech recognition. In He (1988) the Viterbi algorithm is presented for a second-order HMM using the transformation of the model to a first-order with extended state space. The algorithm that was developed here uses the features of the Viterbi algorithm in a slightly different way, tailored to the needs of the PTGC problem as described in Section 2.5.

2.4 Multiple-Output (N-best) Conversion Algorithm

The Viterbi algorithm produces the overall best state sequence by maximizing the overall probability $P(O \mid Q)$. If the N best state sequences are needed, then the algorithm must be modified to keep the N best state sequences from q_1 through q_t. Schwartz and Austin (1991) and Schwartz and Chow (1990) present the N-best algorithm in detail. The following consideration is the basis of the multiple-output conversion algorithm:

> Let $Q_t^E = \{Q_1(t), Q_2(t), \ldots, Q_E(t)\}$ be the globally E best hidden-state sequences that end at state $q_t = S_i$ at a given time t. By "best" we mean, as usual, those sequences having the highest probability. If one of the globally E best hidden-state sequences that starts at $t = 1$ and ends at $t = T$ passes from state S_i at time t then it must have one of the members of Q_t^E as part of the path from time 1 to t.

To prove this, only the following assumptions are required: $Q_x(t)$ is a state sequence that ends at time t at state S_i; $Q_x(t) \notin Q_t^E$; and $Q_x(t)$ is part of one of the E globally best hidden-state sequences. Clearly $Q_x(t) \notin Q_t^E \Leftrightarrow P(Q_x(t)) < P(Q_i(t))$, $\forall i \in 1 \ldots E$. The probability of the complete state sequence $Q_m(T)$ $(1 \leq m \leq E)$ which contains $Q_x(t)$ would be:

$$
\begin{aligned}
P(Q_m(T)) &= \pi_{q_1}\beta_{q_1}(O_1) \cdot \prod_{\tau=2}^{T} \alpha_{q_{\tau-1}q_\tau}\beta_{q_\tau}(O_\tau) \\
&= \pi_{q_1}\beta_{q_1}(O_1) \cdot \prod_{\tau=2}^{t} \alpha_{q_{\tau-1}q_\tau}\beta_{q_\tau}(O_\tau) \cdot \prod_{\tau=t+1}^{T} \alpha_{q_{\tau-1}q_\tau}\beta_{q_\tau}(O_\tau) \qquad (10) \\
&= P(Q_x(t)) \cdot \underline{\prod_{\tau=t+1}^{T} \alpha_{q_{\tau-1}q_\tau}\beta_{q_\tau}(O_\tau)}
\end{aligned}
$$

Since $q_t = S_i$, the underlined part of (10) is independent from $P(Q_x(t))$. But $P(Q_x(t)) < P(Q_i(t))$, $\forall i \in 1 \ldots E$. This means that there are at least E more paths leading to state S_i at time t that are more probable than $Q_m(T)$, which is a path among the first E most probable paths; a contradictory statement.

Summarizing, we have shown that we only need to keep the locally (at any time t in $1 \ldots T$) E best paths as we go along the possible state sequences for every possible state. When we arrive at the end, we only need to keep the E globally highest probabilities and trace back the states that resulted in these.

2.5 Final system

The final version of the conversion system uses the previously mentioned methods, i.e., the second-order HMM and the N-best version of the Viterbi algorithm along with a transformation that is necessary to speed up the execution of the conversion.

The algorithm as described previously has many disadvantages for a PTGC system from the implementation point of view. The values of the parameters of the model are in the range of 100. This implies that, considering storage, we need to keep in memory $100 \times 100 \times 100$ double precision floating point numbers for matrix A along with the other data of the model and the algorithm. To be exact we need for:

A: N^3	double precision floating point numbers
B: $N \times M$	double precision floating point numbers

π: N double precision floating point numbers

ρ: N^2 double precision floating point numbers

δ: $N^2 \times E \times T$ double precision floating point numbers

ψ: $N^2 \times E \times T$ short integers

(for the meaning of δ and ψ see the algorithm presented in the Appendix).

In the above, if we substitute the values of the Greek PTGC system (N=140, M=70, T=30, E=4) for the symbols, we can see that we need no less than 45,708,320 bytes for storing these data. Aside from the problem of storage, the computer has to execute the inner part of the second-order-multiple-output conversion algorithm $N^3 \times E \times T/2$ times (the average length of a word is about $T/2$), i.e., 219,520,000 times per word. As is clear from the presentation of the algorithm, this part contains a rather time-consuming sorting procedure plus a floating point multiplication. It is obvious that this is an unacceptable time delay for real-time applications.

To decrease the algorithm execution time and storage needs we introduced the following improvements:

a. Taking advantage of the relative sparseness of matrix B, we first determine if $B_j(t)$ is nonzero and only then does the algorithm proceed to the rest of the processing. This has decreased the execution time of the conversion by nearly 100 times.

b. We do the same for matrix A. This means that if the indices (i, j, k) indicate a zero transition probability then the algorithm proceeds without trying to calculate the overall probability, thus eliminating a floating point multiplication.

c. Since at every time point the intermediate variable $d(t)$ is calculated only from $d(t-1)$ we keep only two copies of d_{ij}, one for t and one for $t-1$.

Finally, the fact that only multiplications are involved in the processing of the conversion algorithm led us to transform the algorithm to use only additions. In the Appendix, the algorithm we implemented is presented.

3. Testing

The proposed system has been tested and evaluated in two separate procedures: the training process and the conversion process. The training process has been performed using dictionaries that contain both the graphemic form and the phonemic form of words along with the frequency of occurrence of the words in the corpora that were used for the creation of the dictionary. The output of the training process was a file containing the model parameters (transition matrix, initial state probabilities, etc.). The conversion process has been performed using various portions of texts not included in the training texts, for which the phonemic form and the graphemic form were known. The phonemic form was converted into orthographic (graphemic) form using the algorithm and then compared with the original. To thoroughly test the performance of the system, a series of experiments was conducted. These experiments were designed so that the following set of factors could be examined:

- training and testing material domain: general, specialized, name directory
- type of the phonetic form: correct phonemic strings, corrupted speech
- language: Dutch, English, French, German, Greek, Italian, Spanish
- conversion algorithm version: first-/second-order HMM

For each experiment the following figures were measured:

- number of words converted correctly
- number of phonemes converted correctly
- rank of the correct word (for the multiple-output versions of the conversion algorithm)
- time response
- memory requirements

Details about the factors presented above and the quantities that were measured are given below.

3.1 Evaluation factors

For all languages tested, the models were created using full-form dictionaries set up during the EEC ESPRIT project 291/860 "Linguistic Analysis of the European Languages" (ESPRIT 1987) from corpora of about 300,000 words. These dictionaries cover three domains: office environment, newspapers, and law. The input to the conversion process was separate 10,000 word texts not included in the training dictionaries. The testing material was taken from the above domains. Furthermore, for Greek, two additional dictionaries of proper names provided by the ONOMASTICA (LRE 61004) project were used for training and testing the algorithm in a name directory environment. This was done to get a more accurate indication of the system's performance in applications where a *complete* dictionary can never be available.

A second set of training and testing material was created from the above using a phoneme confusion matrix that simulated the output of a speech recognizer. The confusion matrix relates the input (correct) with the output (corrupted) phonemes employing probabilities of the form $m_{ij} = P(O_{\text{out}} = P_j \mid O_{\text{in}} = P_i)$. The texts used for the training and testing phase were corrupted according to these probabilities. Different confusion matrices were applied to show the degradation of the performance as a function of the input phonemic corruption. In Section 4.2, the two sets of results are presented and compared.

Finally, one more experiment per language was performed using a first-order HMM, so that the ambiguity in each of the languages tested could be revealed and a comparison of the performance of the two HMM models could be made.

3.2 Performance Criteria

For every experiment carried out, the success rate of the conversion algorithm was measured for each output candidate (the system was asked to produce a maximum of four candidates if available) in two levels:

a. Errors at word (state sequence) level: The system counts one error for every phonemic *word* not converted correctly to its graphemic form.

b. Errors at symbol (state) level: The system counts one error for every graphemic *symbol* (unit grapheme) that does not match with the corresponding symbol of the correct graphemic transcription.

As an example, if instead of the correct transcription of the Hellenic word /trapézi/ 'table' to the graphemic form τραπέζι, the system produces τραπέζη, this counts for one error per one word (i.e., 100% error) and for one error per seven symbols (or 14.3% error). This distinction was made because the first error type (word error) is more important from the user's point of view, while the second type (symbol error) is a more objective measure of the performance of the system.

In addition to these error types, the average symbol errors per incorrect word were counted. This is a measure of the quality of the system output since it shows whether an incorrect word is easily comprehensible or not.

Another very important feature was also measured: the position in the output list of the correct candidate. The distribution of this variable is very important, so that decisions about the trade-off between speed and accuracy can be made. Finally, for each experiment, the amount of time and computer memory needed were counted, to get a measure of the applicability of the algorithm in real-time applications when using general- or special-purpose hardware.

4. Results

4.1 Explanation of tables and charts

In all tables and charts some symbols have been used to designate the different parameters of the experiments. More precisely, *Exp n* ($n = 1, 2, 3$) designates the experiment type as follows:

Exp 1: uses a first-order HMM with correct phonemic representation of the input

Exp 2: is like *Exp* 1 but uses a second order HMM

Exp 3: is like *Exp* 2 but with corrupted phonemic representation simulating the output of a speech recognizer

The letter combinations *E*1, *E*2, and *NE* show the domain of the experiment: *E*1 experiments use the office environment corpora for training and assessment, *E*2 the law corpora, *NE* the newspaper corpora. For the name corpus (Table 8) *N*1 shows experiments using a corpus of surnames and *OD* experiments using a corpus of street names. It must be noted that in all experiments the testing material was not included in the training of the model although it may belong to the same domain.

In Tables 1 through 8 the model parameters for all the models created for the experiments mentioned above are presented. The columns show the density (i.e., the number of nonzero elements) of the respective model parameters (initial hidden-state probability vector π, initial hidden-state pair probability vector ρ, observation-symbol probability matrix B, and hidden-state transition probability matrix A). The values are percentages. The matrix density is a way of measuring the saturation of the model, that is, whether the model is sufficiently objective or is too dependent on the nature of the training material. One can see from these tables the important differences between the languages on which the experiments were performed.

Table 1
Model parameters for Dutch.

	π	ρ	B	A
E1	53.21	4.29	2.38	0.52
E2	39.45	2.48	1.46	0.22
NE	52.29	4.88	2.28	0.58

Table 2
Model parameters for English.

	π	ρ	B	A
E1	61.46	5.02	6.05	0.50
E2	61.46	5.75	7.08	0.58
NE	65.63	8.72	9.41	1.19

Table 3
Model parameters for French.

	π	ρ	B	A
E1	57.79	4.43	3.11	0.21
E2	60.42	4.51	3.29	0.22
NE	64.02	5.02	3.36	0.32

Table 4
Model parameters for German.

	π	ρ	B	A
E1	59.26	3.56	2.16	0.31
E2	62.96	4.00	2.34	0.37
NE	61.48	4.86	2.78	0.47

Table 5
Model parameters for Greek.

	π	ρ	B	A
E1	48.89	2.51	1.56	0.25
E2	55.56	3.28	1.61	0.33
NE	60.74	5.09	1.80	0.52

Table 6
Model parameters for Italian.

	π	ρ	B	A
E1	82.00	10.00	3.10	2.38
E2	86.00	13.88	3.10	3.32
NE	70.00	9.84	2.78	1.99

Table 7
Model parameters for Spanish.

	π	ρ	B	A
E1	64.286	5.329	1.83	0.661
E2	72.619	6.76	1.863	0.771
NE	65.476	6.08	1.775	0.706

Table 8
Model parameters for names.

	π	ρ	B	A
N1	49.47	5.25	2.12	0.60
OD	64.21	10.15	2.87	1.78

In Tables 9 to 17, a summary of the conversion results is presented for the three sets of experiments carried out. The columns have the following meaning:

d:	$\overline{l_s/l_w} \times 100\%$ where l_w is the size of a word in error (in characters), l_s is the number of incorrect characters in the word, and $\overline{l_s/l_w} \times 100\%$ is the mean value estimated over all wrong words. This number is a measure of the similarity of wrong words with the corresponding correct words (percentage). A small percentage indicates a high similarity.
$1(s)$:	symbol conversion success rate for the first position (percentage).
$1(w)$:	word conversion success rate for the first position (percentage).
1-2, etc:	word conversion success rate accounting for all the referenced positions (percentage).

Figures 2 through 10 give an analytic overview of the results in each language. The legends of these figures have the form cc/n where cc is a two letter code for the corpus domain (E1/E2/NE/N1/OD, as described in the beginning of this section) and n is either 1 for a first-order model or 2 for a second-order model. For example, the legend E1/1 means that text of the domain E1 (office environment) was used with a first-order HMM for the experiment.

Table 9
Conversion results for Dutch.

	d	1(s)	1	1–2	1–3	1–4
Exp 1	21.82	93.12	68.49	84.67	89.26	91.92
Exp 2	18.80	98.00	87.62	96.29	97.27	97.60
Exp 3	26.32	78.40	76.23	86.66	90.46	92.72

Table 10
Conversion results for English.

	d	1(s)	1	1–2	1–3	1–4
Exp 1	19.69	95.51	62.95	75.56	81.84	83.51
Exp 2	16.96	97.60	74.53	86.50	88.10	89.14
Exp 3	23.74	78.08	64.84	77.85	81.93	84.68

Table 11
Conversion results for French.

	d	1(s)	1	1–2	1–3	1–4
Exp 1	16.99	96.02	64.23	79.24	82.25	83.70
Exp 2	16.04	97.42	76.36	85.59	87.24	88.31
Exp 3	22.46	77.94	66.43	77.03	81.13	83.90

Table 12
Conversion results for German.

	d	1(s)	1	1–2	1–3	1–4
Exp 1	17.90	95.01	69.94	83.95	90.93	92.79
Exp 2	15.42	97.34	82.81	96.11	97.89	99.05
Exp 3	21.58	77.87	72.04	86.50	91.04	94.09

Table 13
Conversion results for German with no capital letters.

	d	1(s)	1	1–2	1–3	1–4
Exp 1	17.40	95.27	72.77	86.93	91.50	93.27
Exp 2	15.50	99.20	95.00	99.20	99.70	99.90
Exp 3	21.70	79.36	82.65	89.28	92.72	94.91

Table 14
Conversion results for Greek.

	d	1(s)	1	1–2	1–3	1–4
Exp 1	15.17	96.45	72.17	89.03	92.41	94.05
Exp 2	14.32	97.70	85.80	96.17	98.02	99.23
Exp 3	20.05	78.16	74.65	86.55	91.16	94.27

Table 15
Conversion results for Italian.

	d	1(s)	1	1–2	1–3	1–4
Exp 1	13.74	98.94	92.30	99.54	99.91	99.95
Exp 2	13.67	99.83	98.30	99.95	99.99	100.00
Exp 3	19.14	79.86	85.52	89.96	92.99	95.00

Table 16
Conversion results for Spanish.

	d	1(s)	1	1–2	1–3	1–4
Exp 1	15.49	97.89	86.39	96.84	98.66	99.31
Exp 2	14.37	98.98	93.03	96.59	99.90	99.99
Exp 3	20.11	79.18	80.94	86.93	92.91	94.99

Table 17
Conversion results for names.

	d	1(s)	1	1–2	1–3	1–4
Exp 1	12.30	96.24	69.36	83.16	89.39	92.33
Exp 2	11.88	97.83	81.44	93.60	96.86	98.28
Exp 3	16.63	78.26	70.85	84.24	90.08	93.37

In Figures 11 to 13, a summary for each type of experiment is shown in order to compare the performance between the languages. In Figure 14 the average number of times an output position is occupied is given for all the languages. Finally, in Figure 15, the degradation of performance as a function of the corrupted input words is shown. The differences in performance between the languages and the types of domains and models used are discussed in the following section.

4.2 Comments on the Performance of the Proposed System

One can initially observe the number of times the algorithm produced a word in each position 1 to 4 (Figure 14). This number decreases very fast from the first to the last position for most of the languages, which shows that the system does not produce extreme spellings of the input words (even though these may be allowed by the language). The second very interesting feature revealed in Tables 9 to 17 is that the improvement in the system's performance decreases rapidly from the first to the last position of the output, which means that the majority of correct suggestions is included in the first two positions. Column *1(s)* shows that the percentage of erroneous symbols is very small indeed, while column *d* shows that even though a word may be incorrect, only a small percentage of its symbols may be wrong (about 15% on average in *Exp* 2), which proves that the output of the algorithm is very easily human-readable even when it contains errors.

The performance of the algorithm varied widely, depending on the language being tested. This is due to the differences in spelling in each language and, consequently, to the training the model required. As described in Section 3.1, the available material for training were 300k-word corpora for all languages. This amount was sufficient for some languages (Dutch, German, Italian, Greek, and Spanish) but insufficient for

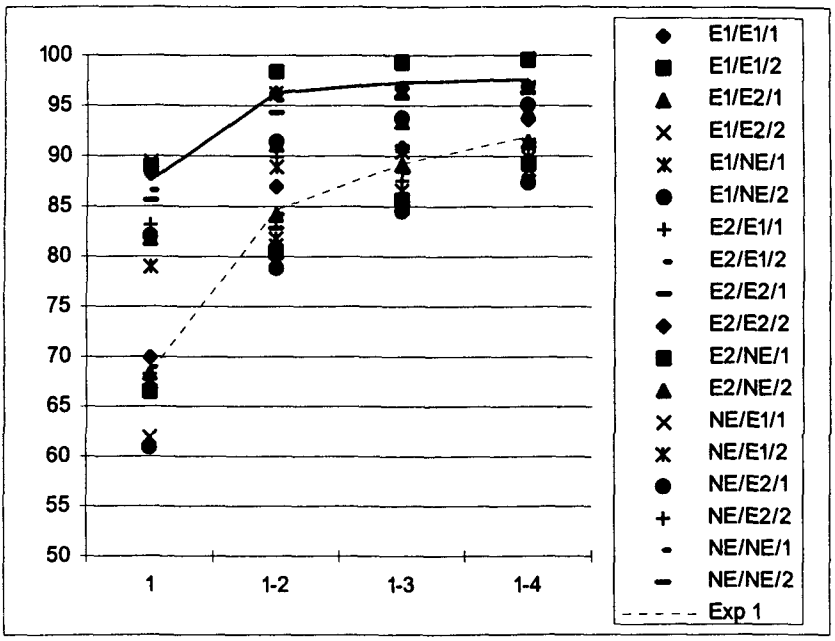

Figure 2
Overview of results for Dutch.

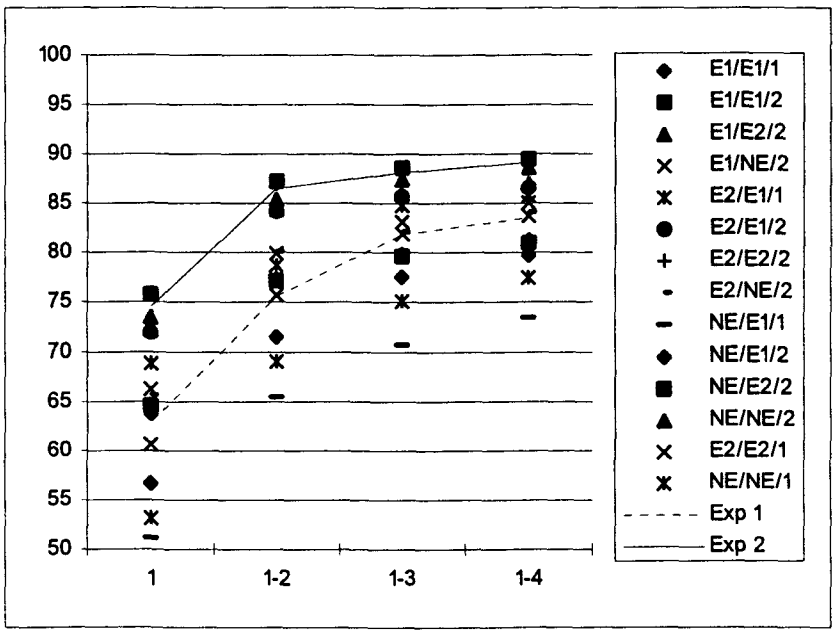

Figure 3
Overview of results for English.

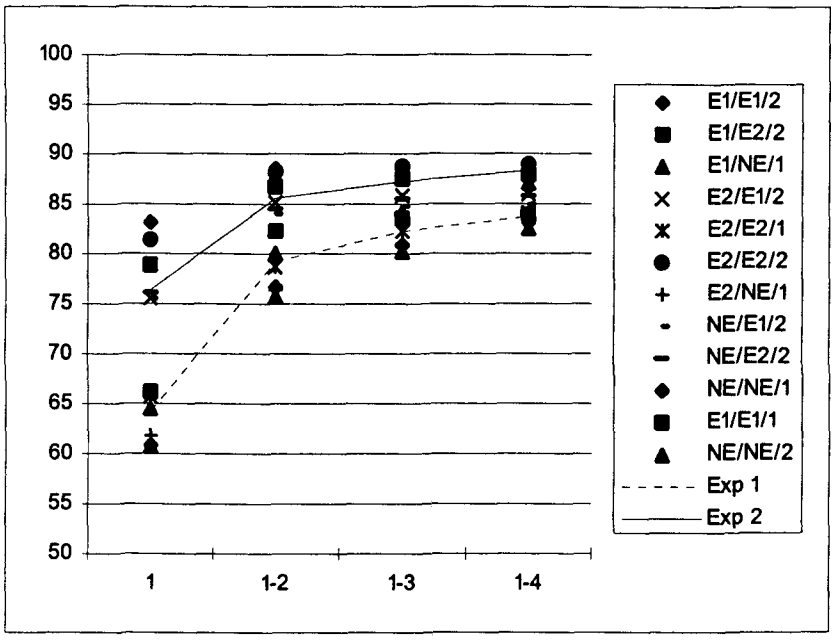

Figure 4
Overview of results for French.

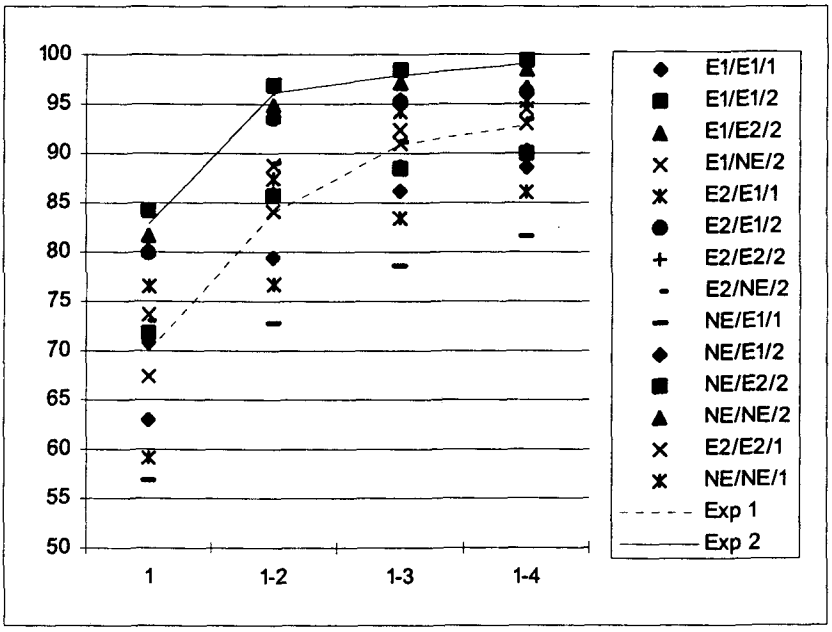

Figure 5
Overview of results for German.

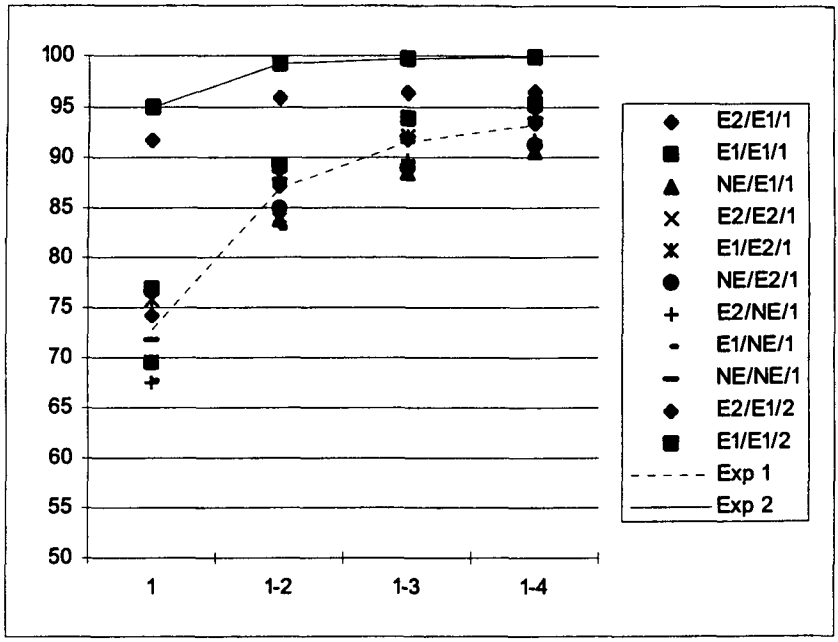

Figure 6
Overview of results for German (no capitals).

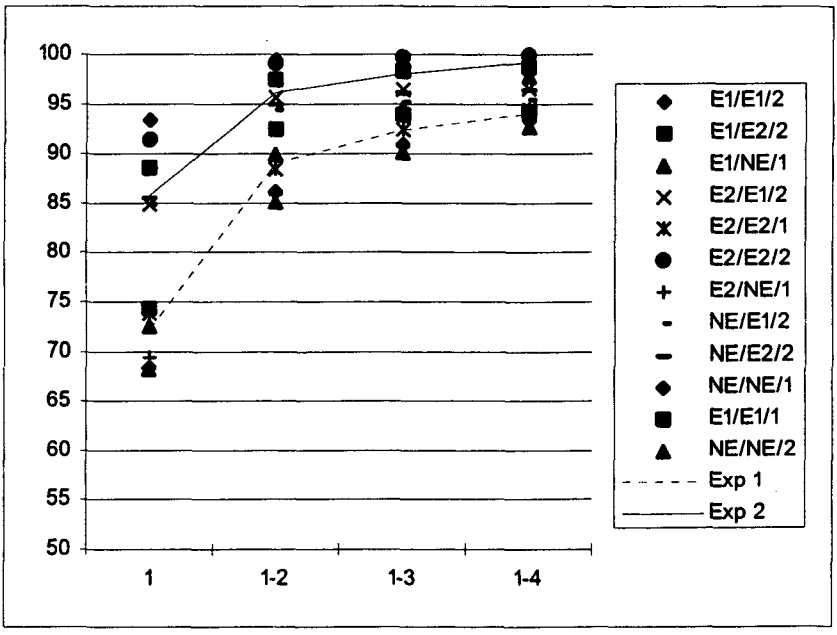

Figure 7
Overview of results for Greek.

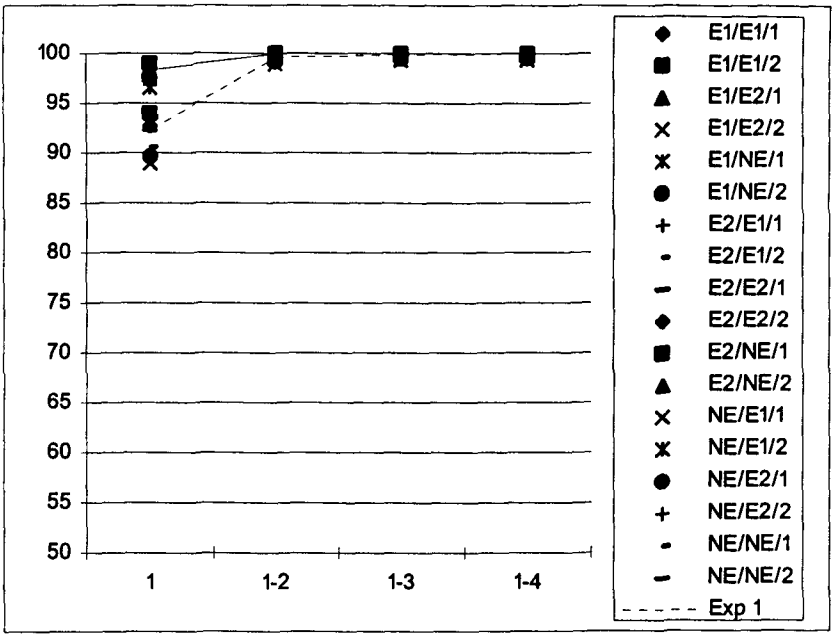

Figure 8
Overview of results for Italian.

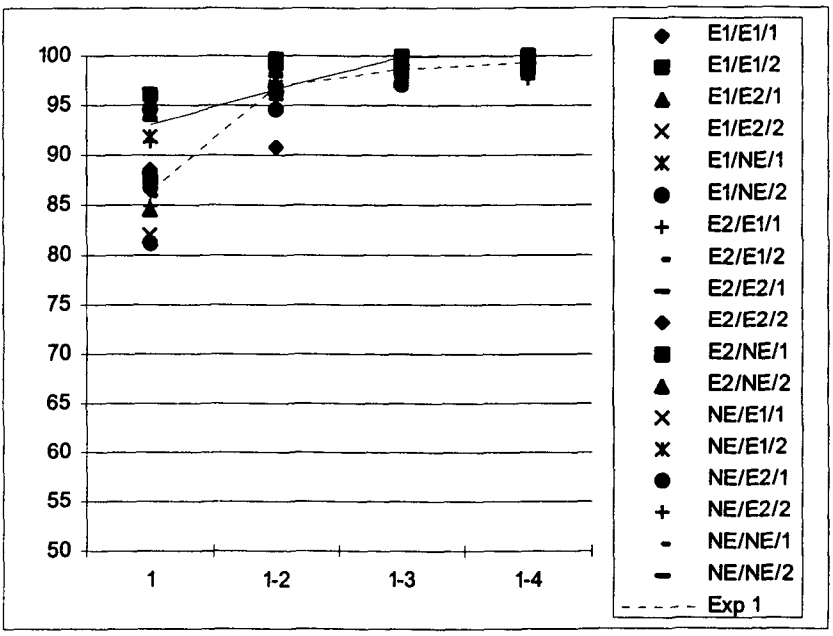

Figure 9
Overview of results for Spanish.

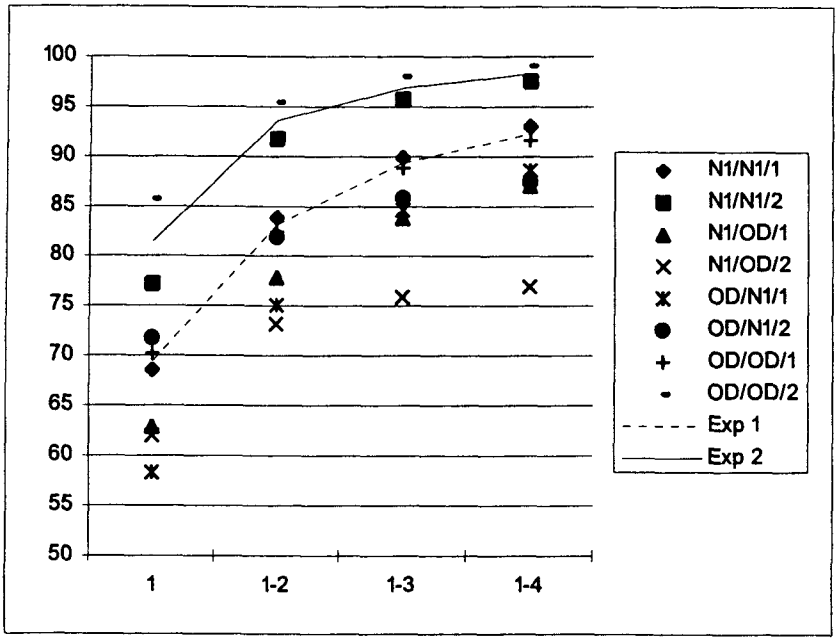

Figure 10
Overview of results for names.

others (English and French). A more detailed presentation of the algorithm's behavior in the languages tested follows.

For Dutch, the model gives relatively good results (97.6% for four output candidates). Spelling in Dutch is rather straightforward for etymologically Dutch words, but words of foreign origin are usually spelled as in the lauguage of their origin. These words are responsible for most of the errors encountered.

The model performed worse for English than for the other languages mainly because the relationship between pronunciation and spelling is less regular. This resulted in fewer grapheme transitions in the training corpus and meant that the standard training period was insufficient. Another problem is that compound words usually keep the initial pronunciation of their components (e.g., in words such as "whatsoever", "therefore", etc.); this leads to many errors for an algorithm like the one proposed here, which has no information about the origin and etymology of each word. Similar work (Parfitt and Sharman 1991) shows the same problems in a slightly different context. Of course, more training of the model would improve performance.

With French, there is a special problem, which does not occur with other languages: there exist many homophones that are distinguished only by the presence or absence of various mute letters at the ends of the words. This feature significantly increases the number of states that have to be defined. Consequently, the available training material was inadequate for the creation of a correct model, and led to poor performance.

The model performed well with German. The only drawback was the decision about the type of the first letter (uppercase or lowercase); nouns always start with a capital letter while other words do not. This is the primary cause of the errors introduced in the experiments with German. Experiments ignoring this ambiguity significantly improved the German results as can be seen from a comparison of Figure 5

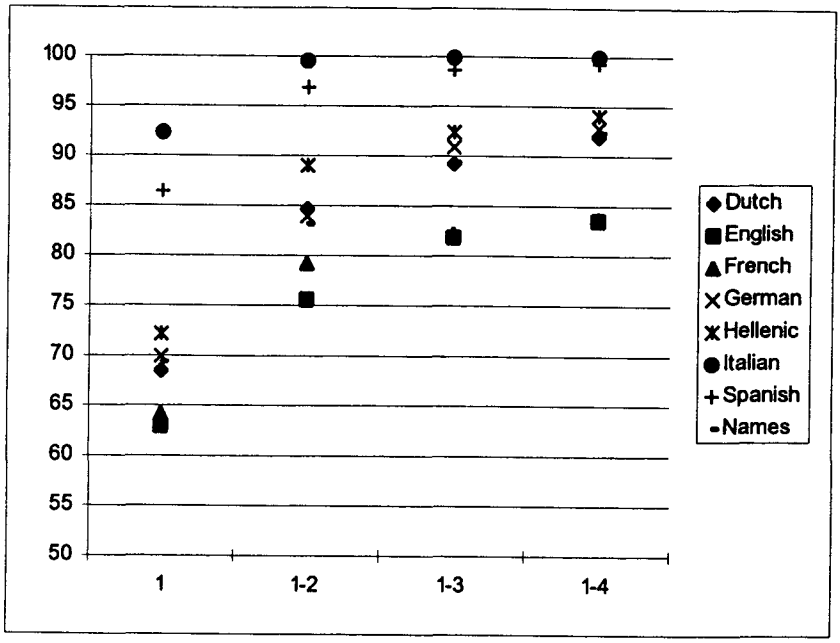

Figure 11
Summary of performance: Experiment 1.

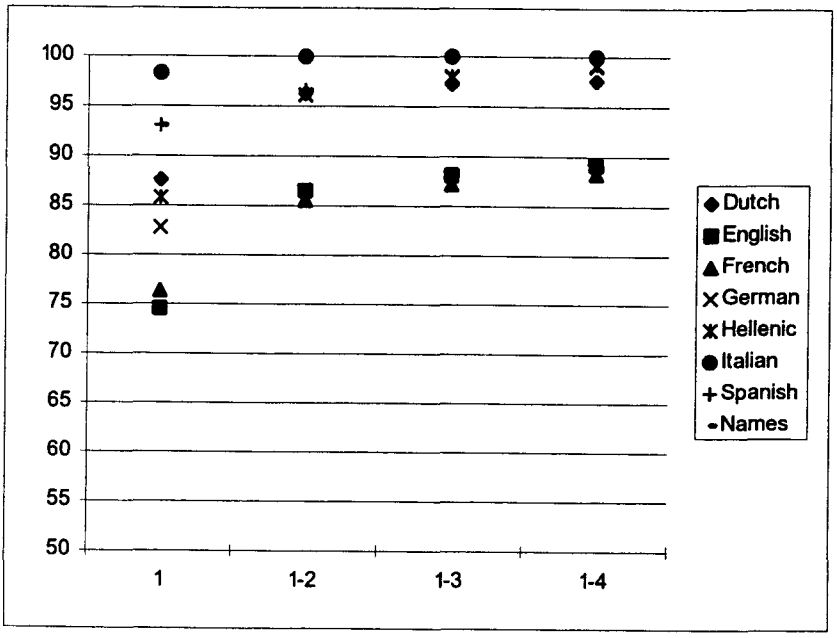

Figure 12
Summary of performance: Experiment 2.

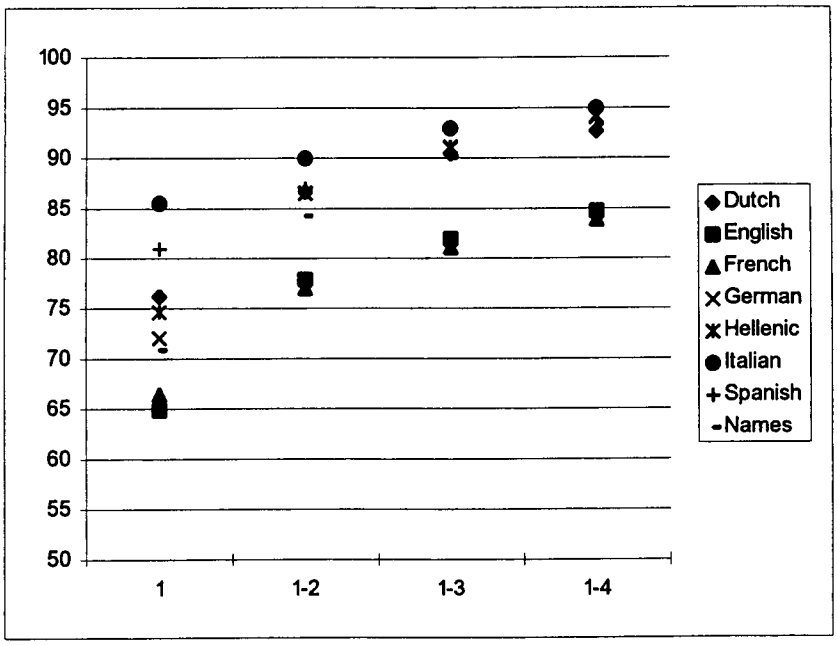

Figure 13
Summary of performance: Experiment 3.

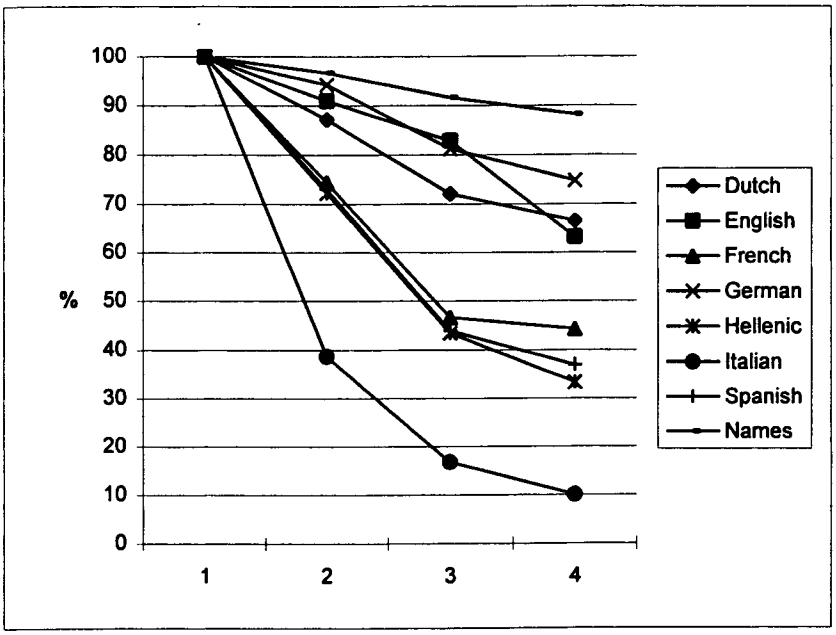

Figure 14
Occupation of output candidate positions.

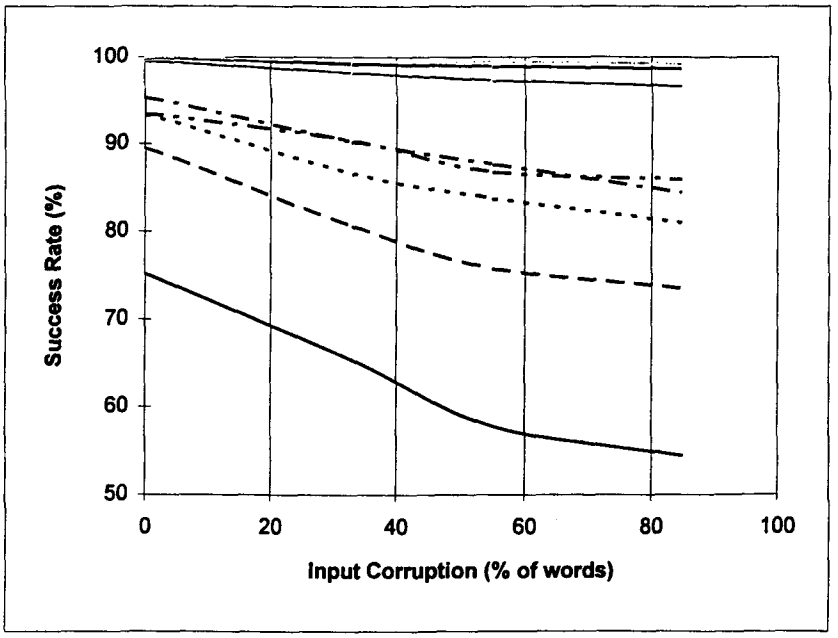

Figure 15
Degradation of output vs input corruption.

and Figure 6.

With Greek, the model behaved quite well, reaching more than 99% success for the second-order HMM experiments with up to four output candidates. Figure 7 illustrates the difference between the performance of *Exp* 1 and *Exp* 2 (order of model) in the first output position. These results are the consequence of two contradictory features of the Greek language:

a. every grapheme is usually pronounced in the same way (i.e., corresponds to one phoneme), and

b. every phoneme usually has more than one possible spellings regardless of its neighboring phonemes.

As an example, the phoneme /i/ can be transcribed as ι, η, υ, ει, and οι in almost any context (Petrounias 1984; Setatos 1974). Other problems arise from the consonants, which can be either single or double without any change in the pronunciation.

Finally, the model gave extremely good results with Italian and Spanish, reaching more than 99% success for the second order model and up to two or three output candidates for known and unknown text experiments, respectively. This is because there is usually a one-to-one correspondence between phonemes and graphemes in these languages.

Another dimension of the analysis of the results is the domain of the experiment. The model behaved best in experiments that used the newspaper corpora, which are more casual in style and richer in vocabulary than the other domains. These corpora usually contain more grapheme transitions, which give greater detail about the spelling mechanism of the language, and provide the most efficient training possible. The

experiments on the Name corpora resulted in lower scores than the corresponding experiments on the Hellenic general corpora reaching 92.3% for *Exp* 1, 98.3% for *Exp* 2, and 93.4% for *Exp* 3 for four output candidates. The main difference in the success rate (Table 17) is due to the size of the training corpora (the training, especially with the street names, was inadequate) and to the fact that names are usually spelled or pronounced in a more arbitrary way than other words.

Finally, as expected, the model performed worse in experiments using as input a simulation of a speech recognizer output (distorted speech) than in the corresponding experiments using a correct phonemic representation of the words. However, by measuring the ambiguity introduced by the speech recognizer output, it can be seen that the PTGC system in fact improved the performance of the overall system (recognizer simulator and PTGC). This was also expected, since in the training phase the model is trained using the correct graphemic form of the words, which is later reproduced in the conversion experiments. Evidently, the performance of the algorithm depends on the amount of distortion introduced in the input phonemic string. Figure 15 shows the degradation of the success rate of the algorithm as a function of the corruption of the input stream. The dashed lines refer to a first-order HMM experiment, while the solid lines refer to a second-order HMM experiment. The input degradation does not affect the overall system performance very much (in any of the four output positions) even when more than 85% of the input words have at least one incorrect phoneme. It must be noted that, in this case, about 30% of the input symbols (unit phonemes) have been replaced by erroneous ones but still the score of the first four positions remains above 98%.

5. Conclusion

We have presented a system for phoneme-to-grapheme conversion (PTGC) at the word level that uses the principles of hidden Markov models to statistically correlate the graphemic forms to the phonemic forms of a natural language. A first- and second-order HMM have been created and the Viterbi and N-best algorithm have been used for the conversion. In the latter case, experimentation showed that no more than two solutions (output candidates) are necessary to produce the correct output with an accuracy higher than 96% for most of the languages the system was tested on. If four output candidates are allowed, then this rate reaches 97% to 100%. Moreover, it must be noted that the success rate of the system, although already good enough, can be further improved by better training on a larger corpus of selected texts.

An important advantage of the system presented here, in comparison to rule-based or dictionary look-up systems, is that it produces only one (or at least very few) graphemic suggestions for each phonemic word. In the first case (one suggestion), no language model is needed to disambiguate potential homophones at sentence level. In the second case (a few suggestions), the execution speed of the system is substantially higher than in rule-based or dictionary-based systems, due to the small number of suggestions per word. The prototype system, which was implemented on a 486-based personal computer, responded at an average rate of one word per second for *Exp* 2 (second-order HMM) and about ten times faster for *Exp* 1 (first-order model). The fact that the algorithm scans the input word linearly (once from the beginning to the end) means that it can work in parallel with other modules of speech recognition systems and produce output with a very short delay after the end of the input.

Another advantage of this system is that it can work in any language in which the pronunciation of the words is statistically dependent only on their spelling. The only language-specific part of the system, i.e., the algorithm for the segmentation rule

definition, is straightforward and does not need any special linguistic knowledge but only familiarity with the target language to be processed.

The system is not limited by any dictionary. This is a significant advantage in very large or unlimited dictionary applications. An implication of this property is that the system does not try to match the input utterance to the closest word (by some measure of distance) contained in the dictionary but rather tries to find its most probable spelling. In this sense, the output of the PTGC system never misleads the final human user about what the input was.

Note also that the system is symmetric between the two forms of a natural language: graphemic and phonemic. This implies that without any modification, the algorithm can be used in the reverse order (i.e., for a grapheme-to-phoneme conversion system, widely used in text-to-speech [or speech synthesis] systems) by just interchanging the phonemic with the graphemic data of the training procedure.

Last but not least, the fact that the system is not rule based but uses an algorithm based on probabilities makes it possible to implement the system in hardware, resulting in a system adaptable to any real-time speech recognition system. As can be seen in the equations of the appendix the algorithm is highly parallel since the values of $d_{jk}(t)$ are independently computed from the values of $d_{ij}(t-1)$; this means that these calculations can be performed concurrently. In this manner, the response time of the complete algorithm can be proportional to N^2 rather than to N^3, yielding a system that can serve as a module for any real-time speech recognition system.

In conclusion, the proposed method has the following advantages:

- It is language independent, making it adaptable it to any language with little effort.
- It does not need a dictionary and thus is free of any restrictions.
- It gives only one or very few transcriptions per word.
- It can be implemented in hardware and serve in real-time speech recognition systems.

Appendix: Implementation Notes

The fact that only multiplications are involved in the processing of the conversion algorithm led us to convert the algorithm to use only additions. Instead of using probabilities, we used their negative logarithm, thus yielding distances. This transformation offers two advantages: First, a four-byte integer representation is used for each number instead of a ten-byte floating point representation, without any loss of accuracy, thus reducing memory requirements. Second, a substantial increase in processing speed is achieved, since the fixed point addition is faster than floating point multiplication.

Clearly, since the probability P is a number between 0 and 1, $-log(P)$ is a number in the range $0 \ldots \infty$. In order to reduce computation, one of the two library-supplied logarithm functions had to be used, i.e., $\log_{10}$ or $\log_e$. It can easily be seen that if $a > b$, then $-\log_a(P) < -\log_b(P)$. For this reason the natural logarithms (base $e = 2.71828$) were chosen instead of decimal logarithms.

To benefit from the above transformation, a fixed point arithmetic should be used (floating point addition is as troublesome as floating point multiplication if not more). At this point, we had to make decisions taking into account implementation-specific parameters. The system was implemented using the C programming language on

a 486-based computer in protected mode, thus exploiting its full 32-bit architecture. The probabilities are first calculated using the greatest available floating point representation, which is 80 bits for a long double floating point number. The smallest nonzero value of P in this representation (and effectively its best resolution) is 3.4×10^{-4932} corresponding to the greatest value of $-\log(P)$, which is 11,355.126. An unsigned long integer has a 32-bit dynamic range, which results in a maximum value of $2^{32} = 4,294,967,295$. Since for every state we need to add two distances, one from matrix A and one from matrix B, we must be sure that there will be no overflow after all the additions that must be made for each word. The system as tested uses a maximum number of 30 states per word, a constant that has not yet been surpassed by any word in all the languages in which it was tested. This means that the maximum distance value must be $2^{32}/60 = 71,582,788$, which results in a scaling factor $f = 71,582,788/11,355.126 = 63,040$. By multiplying every distance with this factor and truncating it to its integral part, it is guaranteed that there will be no overflow in the execution of the Viterbi algorithm. This fact allows the elimination of code that would check for overflow during the algorithm, resulting in a much faster code. For reference, the complete algorithm converted to work with logarithms (as it was implemented) is presented below:

Let α', β', π' and ρ' be the HMM parameters after the above transformation and normalization (e.g., $a'_{ijk} = f \cdot \lfloor \log_e(a_{ijk}) \rfloor$ where $f(= 63,040)$ is the factor that was used to facilitate fixed point arithmetic). Then we inductively compute the locally minimum distance δ' and the path ψ' as follows:

Initialization

$$\delta'^{1}_{ij}(2) = \pi'_i + \rho'_{ij} + \beta'_i(O_1) + \beta'_j(O_2), \quad 1 \le i,j \le N \tag{11}$$

$$\delta'^{e}_{ij}(2) = 2^{32}, \quad \begin{matrix} 1 \le i,\ j \le N \\ 2 \le e \le E \end{matrix} \tag{12}$$

$$\psi'^{e}_{ij}(2) = 0, \quad \begin{matrix} 1 \le i,\ j \le N \\ 2 \le e \le E \end{matrix} \tag{13}$$

Recursion ($2 \le t \le T$):

$$\delta'_{jk}(t+1) = \min_i{}^*(\delta'_{ij}(t) + \alpha'_{ijk}) + \beta'_k(O_{t+1}) \quad 1 \le i,j \le N \tag{14}$$

$$\psi'_{jk}(t+1) = \arg\min_i{}^*(\delta'_{ij}(t) + \alpha'_{ijk}) \quad 1 \le i,j \le N \tag{15}$$

Sequence backtracking:

$$d^* = \min_{ij}{}^*\delta'_{ij}(T), \quad 1 \le i,j \le N \tag{16}$$

$$(q_{T-1}, q_T) = \arg\min_{ij}{}^* d'_{ij}(T), \quad 1 \le i,j \le N \tag{17}$$

$$q^e_t = \psi'^{e}_{q_{t+1}q_{t+2}} \begin{cases} 1 \le t \le T-2 \\ 1 \le e \le E \end{cases} \tag{18}$$

where $d*$ is a vector containing the E minimum distance values that correspond to the E state sequences $Q_e = \{q_t\}$, $t = 1 \ldots T$, $e = 1 \ldots E$, which are returned as the best (most probable) state sequences.

Acknowledgments

The text corpora used for the training and assessment of the system were gathered and transcribed to their phonemic form during the EEC ESPRIT-1 Project 860 "Linguistic Analysis of the European Languages." The name directory corpora were obtained from the LRE-61004 ONOMASTICA project. The authors wish to thank Dr. Anastassios Tsopanoglou and Dr. Evaggelos Dermatas for their comments on the paper and their valuable assistance in the final preparation of the manuscript.

References

ESPRIT Project 291/860. 1987. Linguistic analysis of the European languages. *Technical Annex*. European Commission Framework Programme "European Strategic Programme for Research in Information Technology" (ESPRIT).

Forney, G. David Jr. 1973. The Viterbi algorithm. In *Proceedings of the IEEE*, 61(3).

Hannaford, Blake and Paul Lee. 1990. Multi-dimensional hidden Markov model of telemanipulation tasks with varying outcomes. In *Proceedings of the IEEE International Conference on Systems, Man & Cybernetics*, pages 127–133.

He, Yang. 1988. Extended Viterbi algorithm for second order hidden Markov process. In *Proceedings of the IEEE 9th International Conference on Pattern Recognition*, pages 718–720.

Katz, Slava M. 1987. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 35(3).

Kerkhoff, J. and J. Wester. 1987. FONPARS1 User Manual. Internal publication 291/860, *ESPRIT project*.

Kriouile, Abdelaziz, Jean-François Mari, and Jean-Paul Haton. 1990. Some improvements in speech recognition algorithms based on HMM. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, pages 545–548. ICASSP.

Laface P., G. Micca, and R. Pieraccini. 1987. Experimental results on a large lexicon access task. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, pages 809–812. ICASSP.

Lee, Kai-Fu. 1989. Hidden Markov models: Past, present and future. In *Proceedings of Eurospeech*, 1:148–155.

Levinson, S. E., M. Y. Liberman, A. Ljolje, and L. G. Miller. 1989. Speaker independent phonetic transcription of fluent speech for large vocabulary speech recognition. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, pages 441–444. ICASSP.

LRE-61004, Multi-language Pronunciation Dictionary of Proper Names and Place Names. *ONOMASTICA Technical Annex*. European Commission Framework Programme "Linguistic Research and Engineering."

Ney, Hermann and Ute Essen. 1991. On smoothing techniques for bigram-based natural language modelling. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, pages 825–828. ICASSP.

Parfitt, S. H. and R. A. Sharman. 1991. A bi-directional model of English pronunciation. In *Proceedings of Eurospeech*, 2:801–804.

Petrounias, Evaggelos. 1984. *Modern Greek Grammar and Comparative Analysis* (in Greek). University Studio Press, Thessaloniki, Greece.

Rabiner, Lawrence R. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, 77(2).

Rentzepopoulos, Panagiotis. 1988. *Phoneme to Grapheme Conversion Using Rules* (in Greek). *Electrical Engineering Diploma thesis*. University of Patras, Patras, Greece.

Rentzepopoulos, Panagiotis and George Kokkinakis. 1991. Phoneme to grapheme conversion using HMM. In *Proceedings of Eurospeech '91*: 797–800.

Rentzepopoulos, Panagiotis and George Kokkinakis. 1992. Multilingual phoneme to grapheme conversion system based on HMM. In *Proceedings of the International Conference on Spoken Language Processing* 2:1191–1194. ICSLP.

Rentzepopoulos, Panagiotis, Anastassios Tsopanoglou, and George Kokkinakis. 1991. A statistical approach for phoneme to grapheme conversion. In *Proceedings of the 1st Quantitative Linguistics Conference* QUALICO.

Schwartz, Richard and Steve Austin. 1991. A comparison of several approximate algorithms for finding multiple (N-BEST) sentence hypotheses. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, pages 701–704. ICASSP.

Schwartz, Richard and Yen-Lu Chow. 1990.

The N-best algorithm: An efficient and exact procedure for finding the N most likely sentence hypotheses. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, pages 81–84. ICASSP.

Setatos, M. 1974. *Phonology of the Modern Greek Koini* (in Greek). Edited by Papazisis (editor). Athens, Greece.

Viterbi, A. J. 1967. Error bound for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions in Information Theory*, 13(2).