Algorithms for Arabic name transliteration

by M. Arbabi
S. M. Fischthal
V. C. Cheng
E. Bart

An Arabic name can be written in English with many different spellings. For example, the name Sulayman is written only one way in Arabic. In English, this name is written in as many as forty different ways, such as Salayman, Seleiman, Solomon, Suleiman, and Sylayman. Currently, Arabic linguists manually transliterate these names—a slow, laborious, error-prone, and time-consuming process. We present a hybrid algorithm which automates this process in real time using neural networks and a knowledge-based system to vowelize Arabic. A supervised neural network filters out unreliable names, passing the reliable names on to the knowledge-based system for romanization. This approach, developed at the IBM Federal Systems Company, is applicable to a wide variety of purposes, including visa processing and document processing by border patrols.

Introduction

Transliteration is the process of formulating a representation of words in one language using the alphabet of another language. For example, words written in Arabic, Cyrillic, or Chinese can be rewritten using the Roman alphabet (a transliteration process known as romanization). The transliteration process depends on the target language (e.g., English, French, Spanish) and on the education and experience of the actual transliterator. Thus, a single foreign word can have many different transliterations.

Further variations may occur in the transliteration of proper names, since the owners of those names may take certain liberties with both the pronunciation and spelling of their names, for either aesthetic or devious reasons. The transliteration of personal names therefore poses an interesting challenge. Automated solutions to such a problem could be useful in several areas, including tourism and security (e.g., border control).

We focus on the process of transliterating Arabic personal names to the Roman alphabet, though the techniques described here should be applicable to other languages (both input and output). Aside from the problems cited above that are inherent in transliteration, the transliteration of Arabic (and other, primarily Semitic, languages) poses an additional challenge: Words are written without "short" vowels. As a result, the transliteration process includes a step called "vowelization," in which the appropriate short vowels are inserted into the "unvowelized" input name.

The process of Arabic name transliteration is illustrated in **Figure 1**. The first step is to enter the names into a database. Currently, we obtain these names from the telephone book of a major Middle Eastern city (Damascus, Syria). We then vowelize these names (the primary focus of this paper), since by convention names are published in the telephone book without short vowels, as is most Arabic text. Once the names are vowelized, they are converted into a phonetic representation. This representation is used to produce the equivalent multiple Roman spellings.

Various techniques have been used to automate different parts of the transliteration process, including the use of

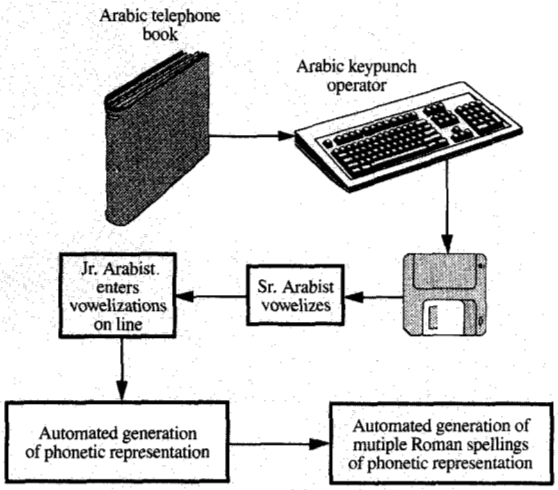

Figure 1

Existing process of Arabic name transliteration.

optical character recognition (OCR) for data entry (there is an Arabic OCR package available from IBM's Cairo Scientific Center), and the use of other tools for the generation of phonemes and alternative Roman spellings, as indicated in the figure. While vowelization is probably the most complicated part of the process, little has been done to automate it.

The complexity of the vowelization process is undoubtedly a major factor contributing to the lack of research in this area. It is also, however, a compelling motivation for finding an automated solution. The existing method of vowelization is costly and error-prone. A single Arabic name may be vowelized in several ways linguistically, but not every vowelization is necessarily used in the Arabic language. To select the correct vowelizations for a set of Arabic names is therefore a tedious job; the task can be further complicated by data entry errors made when the names are originally keyed into the computer. Thus, high error rates are inherent in the transliteration process when it is performed by humans. Detecting and correcting such errors is also labor-intensive.

Another problem with manual vowelization arises from the inconsistent application of heuristics by the Arabic linguist (Arabist). The Arabist who is charged with the task of vowelizing names may not follow a consistent set of rules, and may instead rely on intuition, past experience, and other less scientific knowledge sources. Furthermore, the results of vowelization may vary from one Arabist to another. Because the solution to the vowelization problem is not easily specified, the problem does not lend itself well to traditional methods of computer automation.

The specific problem addressed by this research is the automatic vowelization of Arabic names in ways that are consistent with Arabic morpho-phonology and actual usage.

Some existing commercial packages perform exhaustive vowelization (e.g., [1]). Reference [2] describes a system that performs rule-based vowelization of Arabic natural language words based on a database of Arabic lexemes (a "computational Arabic lexicon"). Our broad approach to vowelization is another form of exhaustive, rule-based vowelization, but it vowelizes a wide variety of names rather than words. In addition, we do not depend on a large database, thus reducing the storage requirements. Unfortunately, many Arabic names are not Modern Standard Arabic—they may be neologisms, corrupted Arabic or foreign. Our narrow approach uses a combination of expert systems and neural networks to achieve a good combination of coverage and accuracy, vowelizing only those names which can reliably be vowelized. The expert system uses a rule-based methodology for generating the vowelizations, while the neural network filters out those words which the expert system should not vowelize according to the rules of Modern Standard Arabic.

Computer-automated transliteration

The generation of Roman transliteration of Arabic names is accomplished in several stages. The names must first be entered into a database (in Arabic), then vowelized, since Roman text requires vowels. After vowelization, each name must then be given a phonetic, Roman representation as a base from which the multiple Roman spellings are produced. A detailed discussion of the process can be found in [3].

- *Creation of Arabic name database (Arabic OCR)*

The first step in Arabic–Roman transliteration is to enter the names into a database. This can be done manually by keypunch operators; however, it would be far more efficient in terms of time and physical resources for an optical character reader (OCR) to read in the names. Currently, we take these names from the telephone book of a large Middle Eastern city (e.g., Damascus, Syria), a method which is extremely conducive to OCR exploitation. However, reading Arabic characters is analogous to reading cursive Roman text, a difficult problem for OCR [4]. Some Arabic OCR programs (e.g., the IBM Cairo Scientific Center's) exist, and we expect to use them in the future. There is also ongoing research in this field [5]. The Arabic names we are now using are entered by human keypunch operators.

• *Vowelization of Arabic*

Arabic can be a difficult language to process for various reasons, one of which is that it is normally written without short vowels. This may seem an odd convention to an American or European audience, but is is very common in Semitic languages. When the target languages of the transliteration are European in origin (Germanic or Romance), and thus require short vowels, we must reintroduce short vowels into these Arabic names. In Arabic, the short vowels are "a," "i," and "u." Long vowels ("aah," "ee," "oo") are always written in Arabic, so they need not be added during vowelization.

• *Transliteration and Roman name generation*

Once we have the vowelizations in Modern Standard Arabic, the vowelized names can easily be converted into a standard, phonetic Roman representation. This can be done with a simple parser or table. This Roman representation is broken down into a group of syllables.

The syllabified phonetic representations are then used, through tables and syllable variation, to produce various spellings in English, French, Spanish, and other languages which use Roman alphabets. For example, "Sharif" in English might be "Cherife" in French.

Technical approach

Our approach is to use a hybrid application comprising a knowledge-based system (KBS) and artificial neural networks. The KBS approach is a good candidate for this problem because it allows the linguistic rules of vowelization to be codified simply in declarative form. Artificial neural networks (ANNs) provide the capability to develop and use statistical heuristics which are not known and/or cannot be represented as declarative rules. All of the neural networks we discuss are artificial. Both KBS and ANN methods are described further in the sections on knowledge-based systems and artificial neural networks.

The KBS is constructed of linguistic vowelization rules which can be applied to a large class of unvowelized Arabic names. This class of "conforming" names follows strict rules of vowelization which are based on the number and structure of the root radicals and their relative positions with respect to other consonants and long vowels.

An artificial neural network is used to filter from the input those names which otherwise would be vowelized inappropriately by the KBS (examples are discussed later). Such names are defined as "unreliable" because, while they appear to be "conforming," they do not typically result in valid Arabic words (i.e., words which are actually in use) when they are vowelized.

Figure 2 depicts vowelization using our hybrid knowledge-based and artificial neural network system.

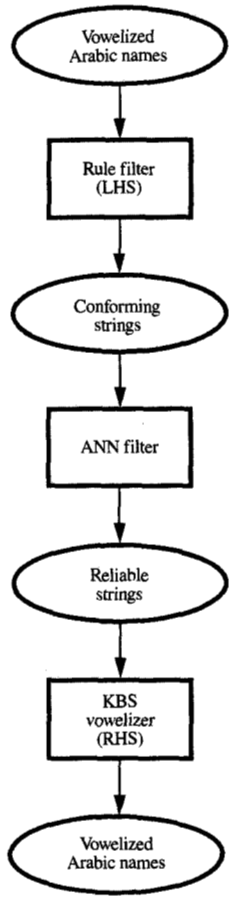

Figure 2

Internal flow of system for vowelizing Arabic names.

• *Knowledge-based systems*

Knowledge-based systems (KBSs) are an alternative means for designing and implementing software applications. While conventional programs are specified as procedures containing sequences of instructions which operate on data, KBS applications are declarative in nature; that is, a KBS is specified in terms of *what* data transformations must take place rather than *how* (step-by-step) to transform data. In addition, a KBS separates the domain knowledge from the general knowledge.

The most common KBS representation scheme is the IF–THEN rule; programs which consist of data and rules for operating on the data are typically described as rule-based programs. Often, the rules in a particular KBS are modeled after the heuristics used by experts who perform the task for which the KBS is built; for this reason,

knowledge-based systems are also often called "expert systems."

The emphasis of a KBS is on encoding the "knowledge" required to solve a specific problem, whether the knowledge be expert or not. This encoding typically takes the form of a rule-based program, but implementation is not restricted to rule-based languages—many KBS applications have been developed with third-generation languages such as PASCAL and C.

Regardless of the implementation language, the KBS methodology allows the programmer to specify a solution at a very high level of abstraction. This is possible because the knowledge for solving the problem (usually specified as rules) is separated from the control logic which carries out low-level processing that is not specific to the problem. The domain knowledge is often called the "knowledge base" (KB), while the component which performs the low-level processing and thus contains general problem-solving knowledge is the "inference engine" [6].

The inference engine is a generic component which controls sequencing and looping and provides a variety of other built-in functions such as pattern matching (i.e., the matching of program variables to actual data items). The inference engine is usually obtained as part of a KBS building tool such as CLIPS, and the rules are specified by the user, in a language appropriate to the tool (most tools have their own languages).

Some of the most common applications of knowledge-based systems are in the areas of diagnosis, planning, monitoring, and control.

- *Artificial neural networks*

While knowledge-based systems provide a mechanism for representing declarative knowledge, artificial neural networks provide a mechanism for representing empirical knowledge. Although statistical models provide the same basic functionality as artificial neural networks (encoding empirical knowledge), artificial neural networks provide the additional ability to learn or be trained. In this context, learning is the process by which patterns (or relationships) are discovered. This discovery process produces an approximation of a function that maps the network input to output. When use is made of a trained network, the same or similar input data will evoke the same outputs, so the network becomes a deterministic collection of mathematical formulas.

An artificial neural network is implemented as a network of interconnected nodes. This architecture is patterned after the brain, and each node represents a single neuron; this is the basis of the designation "artificial neural network." Nodes are organized into two or more layers, with an input layer at the bottom and an output layer at the top. Weighted connections provide the means of communication between nodes; when one node "fires" (has an input signal at or above a certain threshold), it can activate other nodes to which it is connected. The actual topology of the network (number of nodes, number of layers, weights of connections) is determined by the problem being addressed.

Artificial neural network applications are data-driven and are best used to solve problems which are not easily specified algorithmically or declaratively. Popular uses for artificial neural networks in applications include pattern recognition (e.g., in image analysis or credit-card fraud detection) and classification.

- *Hybrids*

Expert systems and neural networks integrate in varying degrees [7, 8]:

- *Transformational* A neural network discovers a set of functions, features, or general rules which are then recoded into facts and rules in a knowledge base.
- *Loosely coupled* Examples of such systems are the following:
  - A system in which a neural network provides inputs to an expert system.
  - A system in which an expert system preprocesses data for use by a neural network.
  - A blackboard system in which neural nets and expert systems are among the knowledge sources.
  - A system in which neural nets and expert systems pass data back and forth to one another, cooperatively solving a problem.
- *Tightly integrated* Examples of such a system include a neural network embedded within the inference engine or other key routine within the rule-based system, or expert systems used as activation functions or filters within a neural network.

Our solution to the vowelization problem contains a neural net which accepts data from a KBS and then passes analysis back to the KBS. Such a system would be described as "loosely coupled" on the basis of our described taxonomy.

- *CLIPS*

An efficient way to implement an expert system is with the use of a "shell." An expert system shell provides a means by which rules can be specified, an inference engine to interpret (or "execute") the rules, and various control mechanisms and utility functions. We chose to use the off-the-shelf expert system shell CLIPS (C Language Integrated Production System). The system was developed by the Software Technology Branch, NASA/Lyndon B. Johnson Space Center, and is available from COSMIC [9]. It is designed to facilitate the development of software to model human knowledge or expertise. CLIPS provides

three knowledge representations: rules, functions, and objects. It has also been designed for full integration with other languages such as C and Ada.

CLIPS knowledge bases can be run in either interpretive mode or compiled mode. The CLIPS interpreter can be used as a stand-alone application (i.e., invoked by the user from the command line), or it can be embedded in another application. The CLIPS code generator makes it possible to run knowledge bases in compiled mode by compiling the CLIPS knowledge representation language into C code which can then be compiled and linked with a driver program for stand-alone operation, or with another application for embedded operation.

- *System description*

The system can be divided into four parts: main program, knowledge base (KB) interpreter, knowledge base, and a set of utility routines (which includes the artificial neural network).

The main program and utility functions are written in C language, and are compiled and then linked with CLIPS interpreter object files to form one embedded CLIPS application.

The knowledge base is written in CLIPS and consists of two files loaded at run time by the main program and later jointly interpreted by CLIPS as one knowledge base.

The main program performs the following initializations for the utility functions:

- Initialize the composite name lookup table.
- Initialize the saturated name detector.
- Initialize the suffix lookup table.
- Initialize the artificial neural network.

(The saturated name detector and the artificial neural network are described in detail in the sections which follow.)

The main program then calls the following CLIPS application programming interfaces (APIs) to execute the embedded CLIPS expert system:

- Initialize CLIPS.
- Load knowledge base files.
- Reset CLIPS to prepare for running.
- Run CLIPS (the inference engine executes the knowledge base rules).

The CLIPS product comes with the source code used to build the product. Availability of the source code makes the product portable, embeddable, and customizable. To embed the CLIPS interpreter into an application, it is only necessary to compile the CLIPS source code (except for the "main" routine) and link the resulting object code with the application. CLIPS provides several APIs for communication outside the CLIPS interpreter; thus, another program is able to start (load), run, and share information with, CLIPS knowledge bases.

- *Knowledge base*

The knowledge base (KB) is implemented in the CLIPS knowledge representation language. CLIPS itself supplies the inference engine. The KB consists of rules and data ("facts"); rules typically consist of two parts: a left-hand side (LHS) and a right-hand side (RHS). The LHS is the condition of the rule (the "if" part), and the RHS is the action of the rule (the "then" part). The LHS can contain multiple conditions combined into a single condition by the logical operators "and," "or," and "not"; conditions are specified as patterns to which facts must be matched. When facts are matched to the LHS of a rule, they are said to be "bound" to the LHS of the rule.

Each rule in the knowledge base has a priority (or "salience"). Priorities are discriminators (not necessarily unique) that determine which rule to execute ("fire") when the condition (LHS) of more than one rule is satisfied (i.e., true). Thus, priority gives us a means of enforcing the order of event occurrences under the nonsequential paradigm of computation that is inherent in rule-based systems.

All of the vowelization rules (rules which actually generate vowelized names) have the default priority of 0. Control rules generally have priorities either higher or lower than the default. Rules with priorities lower than 0 tend to act more like default or postprocessing rules. Those with higher priorities usually play the role of preprocessors; i.e., they are responsible for the things that must be done before the actual vowelization.

Functional description of KBS

The KB system (KBS) interactively accepts an unvowelized name as input and generates all possible vowelizations for that name if it is a conforming Arabic name. The vowelizations are represented in ASCII according to our own character transcription scheme, in which each Arabic letter is paired with one ASCII character.

The KBS carries out several operations on the input name before the actual vowelization takes place:

- First, the KBS examines the composite name table, which contains religious names that contain the prefixed article "al" and are commonly used by Arabs. If the input name matches any of the names therein, the vowelized strings pre-stored with the name in the table are output, and the KBS proceeds to the next input.
- Next, the KBS makes sure that the input string is not corrupt, as in the case of corrupted data. For example, the letter "taa' marbuuTa" (denoted by "@" in our

scheme) should never occur in a non-final position in a word. When the KBS finds a "taa' marbuuTa" in a non-final position in the input name, it rejects the name as inappropriate data and proceeds to the next input.

- Finally, the KBS checks whether there is any possible suffix in the input name. If there is, the KBS creates, for every possible suffix, a separate object to represent this fact, i.e., that the name is possibly composed of a particular stem and a particular suffix. Thus, when the KBS gets to the actual vowelization, the vowelization process is performed for *all* of the possible stem/suffix combinations enumerated.

After the KBS has performed these three steps (table lookup, data filtering, and suffix enumeration), it is ready to insert short vowels for those stems and suffixes with its vowelization rules. Each vowelization rule produces a vowelized name as output by adding one or more short vowels to the unvowelized name (stem and suffix), according to linguistic rules. There is one rule which is an exception to this paradigm, however; this rule outputs the name without adding any short vowels if the name is "saturated."

A name is saturated if short vowels are not required for it to be pronounced correctly as an Arabic name, where correctness is defined in terms of the preservation of Arabic syllable structures. Such a name does not require vowelization, and so is output "as is," but it is also further processed by other vowelization rules because, although it is not necessary to vowelize such a name, it is possible to do so. A saturated name is still open for vowelization for several reasons: 1) the Arabic syllable structures allow for a certain degree of ambiguity; 2) two of the three long vowels can also be consonants or semi-vowels ("waaw" and "yaa' "); and 3) the other long vowel (" 'alif") sometimes really stands for a consonant (" 'alif hamza").

Broad versus narrow

The KB described above works well if our objective is to produce as many transliterations as possible for a given unvowelized name written in Arabic script. We place as many morphologically sound configurations as possible into the LHSs of the vowelization rules of that KB and generate for a particular configuration all the possible vowelizations that are morphologically sound. We designate this approach as the broad approach, the described KB as the broad KB, and the associated KBS as the broad KBS.

The broad approach serves well the exhaustive nature of the described objective; however, it presents some problems. First, there are still names which are Arabic but cannot be vowelized by the broad KBS. These nonconforming names cannot be treated with a rule-based approach, but other techniques for solving this problem are discussed in the section on future directions.

Another problem results from the existence of foreign (i.e., non-Arabic) names which are erroneously treated by the broad KBS as Arabic (because they *appear* to be conforming) and thus are vowelized incorrectly. This problem is also addressed in the section on future directions.

Finally, even among all the morphologically sound vowelizations that are generated by the broad KBS for a genuinely conforming Arabic name, some may not actually exist in the language. In other words, not all the nouns that are morphologically sound and possible are in actual use. This could be a significant problem in the transliteration of a large file of Arabic names and the associated storage of the romanizations in a database. Obviously, the broad approach would require a relatively large amount of storage space and hence a long search time for database retrievals.

Thus, we see that the broad KBS achieves the goal of completeness as well as possible, but at the cost of validity. The solution to this problem is therefore to restrict the KB so that the system generates fewer vowelizations for fewer names but yields a higher rate of accuracy.

The first restriction to impose is on the rules in the KB. By removing those rules with such LHS configurations that are known to produce many unused vowelizations, and removing from other rules RHS actions which generate vowelizations that are rarely actual words, we can achieve—on average—more valid output. The KB is now "narrower" than before.

Even with the described restriction, however, the KB is still not narrow enough. There are still some names which appear to be conforming (that is, they match the LHS of one or more of the vowelization rules) but which actually yield a significant number of invalid names when vowelized by the rules. Such names generally fall into two categories: those that are "foreign" to Arabic and those that are not vowelized according to the same linguistic rules as are represented by the KB rules. We describe these names as "unreliable." These names match the LHS name configurations, but a majority of the vowelizations produced by the rules are not in actual use.

Since we cannot find any heuristics for determining which unvowelized Arabic names are unreliable, we cannot use the knowledge-based approach to solve this problem. The answer to providing a more optimal overall solution therefore lies in supplementing the rules with an empirical approach. In other words, instead of algorithmically determining which names are unreliable, we might attempt to identify unreliable names on the basis of experience.

We could simulate the experience of recognizing an unreliable name by using the KBS described thus far to vowelize a large set of Arabic names, and then allowing an Arabist to classify the output from the KBS as "valid" (known to be in use) or "invalid." Extensive analysis of those unvowelized names which result in mostly invalid vowelizations might reveal subtle patterns which could subsequently be used by the system to avoid similar invalid output.

In practice, however, analysis of this sort is beyond human capability; furthermore, even if patterns could be identified, they would probably be very complex and difficult to represent using a knowledge-based approach. This kind of data-driven solution is an ideal application for the use of an artificial neural network. Accordingly, we add an artificial neural network to the described KBS to identify and filter out those unvowelized Arabic names which are unreliable. We designate the resulting KBS, with its own additional restrictions and its use of the artificial neural network, the narrow KBS.

Figure 3 shows how the narrow and broad methods interrelate, and their coverage of names.

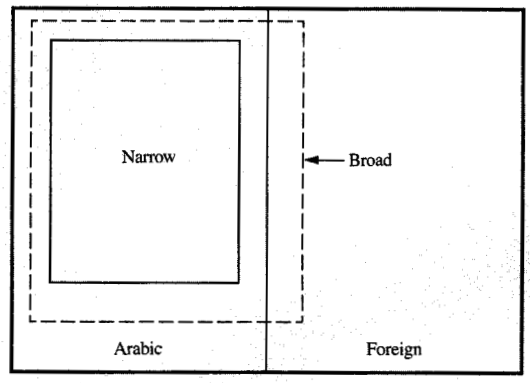

Figure 3

Comparison of the name coverages of the narrow and broad approaches.

Artificial neural network

As previously discussed, we use an artificial neural network in the narrow approach to eliminate those names which cannot be properly vowelized using the expert system. The network was trained using the cascade-correlation method [10], a supervised, feedforward [11] neural processing algorithm.

The artificial neural network accepts names with up to seven letters. Each letter of the name is represented by 35 input nodes corresponding to the 35 different letters of the Arabic alphabet which are supported by the network. Thus, there are 245 inputs (35 × 7) to the network, and only one output (see **Figure 4**).

The output of the network is binary; if the output node fires (with a threshold of 0.5), the name is declared reliable and the expert system vowelizes it. Otherwise, the name is regarded as unreliable and is set aside to be vowelized in some other way (either manually or, in the future, by means of some other type of automation).

The artificial neural network protects against the inaccurate vowelization of some names by the expert system, but a few still slip through. Also, the network occasionally filters out names that could be reliably vowelized by the expert system. On the other hand, the network accurately classifies the vast majority of names (see **Table 1**).

The artificial neural network was trained on roughly 2800 Arabic names and tested on another 1350. The names were a representative sample from the telephone book for Damascus, Syria, that also conformed to the narrow KBS. The cascade-correlation network that performed best

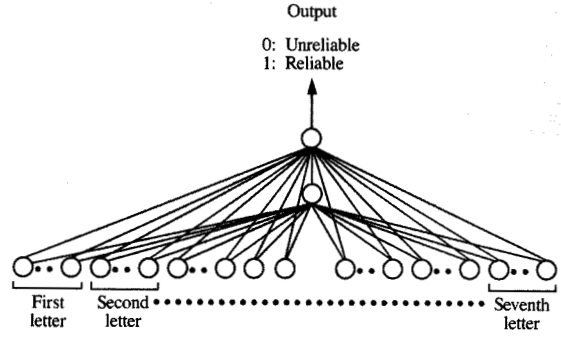

Figure 4

Architecture of the artificial neural network used in this work.

had only 491 connections, so the network was not "memorizing" the training set but actually created its own heuristic for predicting the right classes. This heuristic is primarily statistical, weighting the connections primarily on the basis of the frequency with which a particular letter in a particular position in the network is a member of a reliable name in the training set. Given the excellent generalization of the network, this heuristic appeared to be an effective one.

In the broad approach, 5069 (79%) of the 6415 sample Arabic names conformed to the Arabic name structure and

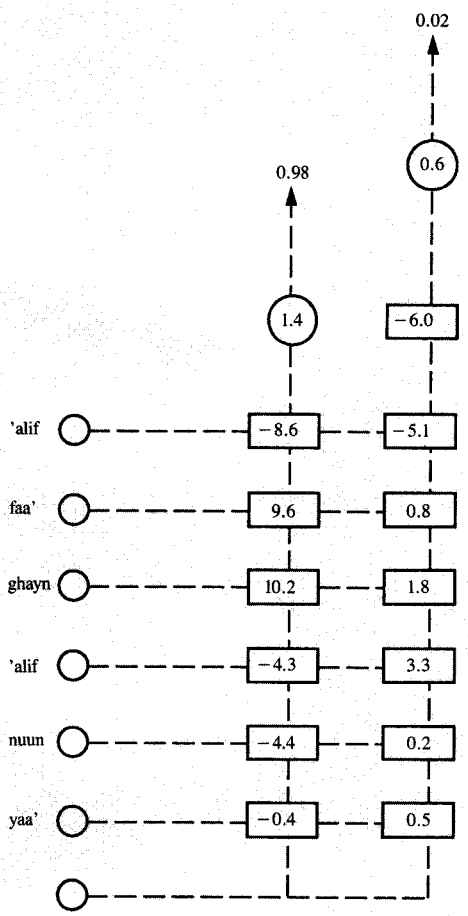

Figure 5

Filtering of the foreign name "Afghany" by the artificial neural network.

were therefore vowelized; the rest were nonconforming. Table 1 indicates that the narrow approach, in which 45.6% of the names were deemed reliable and thus vowelized, generates fewer inappropriate or extraneous vowelizations. It also shows that the neural net does an excellent job of filtering those names which the narrow KB should not vowelize.

By comparison, human experts had error rates of about 10% on similar samples before a laborious editing process reduced the errors to approximately 3–5%.

Discussion

The narrow approach vowelizes over 45% of the names presented to it, with a very low rate of generating inappropriate vowelizations. The broad approach covers closer to 80% of the names, but generates a much higher percentage of extraneous vowelizations. In both cases, we dramatically reduce the labor and time expenditure for vowelization and provide consistent patterns of transliteration (as opposed to the inherent inconsistency of human control). In addition, we greatly reduce the pre-edited error rate, particularly with the narrow approach.

The strength of the narrow approach is in its low rate of erroneous vowelizations. We have found that it performs exceptionally well at reducing the dictionary size (providing potential vast improvement in search performance), and produces fewer errors than human experts. However, many names cannot be vowelized using this approach. The broad approach vowelizes a much higher percentage of the names, but tends to generate many vowelizations that are not in use. On the other hand, both systems are orders of magnitude faster than human experts, who can only vowelize a handful of names in the time it takes for the computer-automated transliterator to generate thousands of vowelized, transliterated names.

For the narrow approach, the artificial neural network plays a crucial role in filtering unreliable names from the expert system. **Figure 5** shows how the network filters the foreign (probably Farsi, or Persian) name "Afghany." Note how the hidden node is the key to driving the output to 0. When the hidden node fires, it produces a strong, negative influence on the output node (weight of −6.0). The weights to the output are a strong negative one from the first character ('alif), and weak positive weights from the other inputs, which would normally produce a borderline result. However, the hidden node is driven hard by the presence of a faa' and ghayn in the second and third positions. The neural net has apparently determined that Arabic names do not normally contain this sequence of letters. In a standard feedforward neural network [11], the output of a hidden node is equal to the sum of its inputs (x_i) plus a trained bias (equivalent to a threshold).

Table 1 Narrow approach: accuracy and vowelization rates.

	Vowelized			*Filtered*		
	Correct	*Incorrect*	*Total*	*Correct*	*Incorrect*	*Total*
Number	2836	90	2926	3432	57	3489
Percent	96.9	3.1	45.6	98.4	1.6	54.4

From this, we obtain the following formula, which shows the output of the hidden node (bias $B = 1.4$):

$$B + \sum_{i=1}^{245} x_i = 1.4 - 8.6 + 9.6 + 10.2$$

$$- 4.3 - 4.4 + 0.4 = 4.3.$$

Thus, the output of the hidden node $O_H = 1/(1 + e^{-4.3}) = 0.98$. This name has only six letters, so none of the 35 bits for the seventh input is on. For each of the six letters that do exist, one of the 35 bits in the network is on and the other 34 are off.

The output node has a bias of 0.6; the summed weights add to 2.1. However, the weight from the hidden node is −6.0, so the input from that node is −5.9; when this is added to the other input weights and bias for the output node, the activation is actually −3.8. The output of the network is $1/(1 + e^{3.8}) = 0.022$, much lower than the threshold of 0.5 necessary to accept the name.

Figure 6 shows a name that is accepted, "Qaddafi." Note that the final (fifth) letter is a long vowel. The hidden node activation is −34.1, so the output, $1/(1 + e^{34.1})$, is virtually zero. Thus, the activation of the output node is just the sum of the weights from the input node plus the bias; the activation is thus 3.2 and the output 0.96, well above our threshold for vowelization of the name. Note that the presence of qaaf and daal in the first two positions strongly suppresses the hidden node, while the presence of 'alif and faa' and yaa' combines with the bias to override the qaaf and daal at the output layer.

The network performs a balancing act among the letters (and their positions), especially those most likely to indicate whether the name in question is reliable. We see in the above examples how certain inputs can strongly inhibit the hidden node from activating (or strongly activate the node). An area for additional investigation is in determining some of the overall statistical probabilities of combinations of Arabic letters (see [12] for some early work in this area).

We believe that the data set used to train the network is large enough and reasonably representative of a typical collection of Arabic names; hence, we feel confident that it will provide similar results on other test data sets in addition to the one we used.

While it is certainly no surprise that an expert system can be applied in mimicking the standard morphology of Arabic, the success of the hybrid system at modeling some of the humanly generated irregularities of Arabic is an intriguing aspect of the modeling.

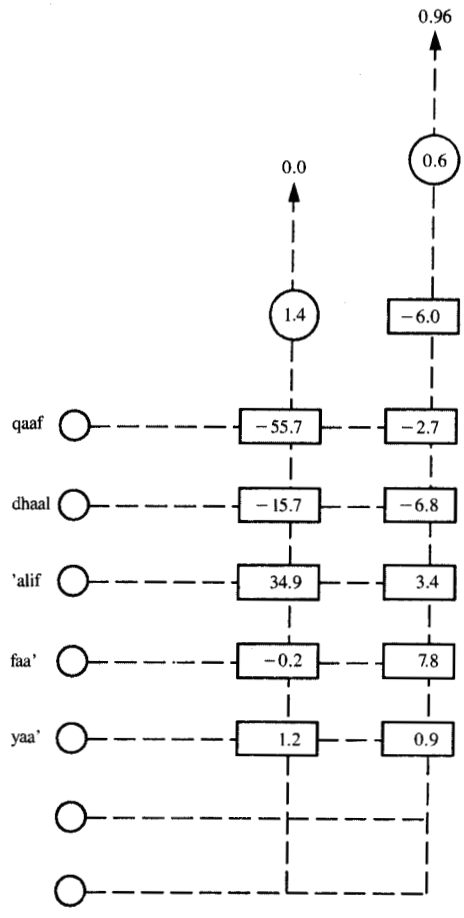

Figure 6

Acceptance of the Arabic name "Qaddafi" by the artificial neural network.

Future directions

An artificial neural network could also be used to "trim back" results of the broad approach, decreasing the dictionary size somewhat and eliminating many unreliable names from being inaccurately vowelized. In this instance, the network might be used after the expert system generates vowelized names and would weed out those names which are incorrect or otherwise unused. Another important task would be to automate the vowelization of those names which even the broad approach is unable to vowelize. An artificial neural network, possibly in combination with genetic algorithms and fuzzy sets, may prove highly effective on this relatively small (<20%) subset of names.

Furthermore, the present work should be extendable to other dialects of Arabic or even to languages which use the Arabic alphabet but are fundamentally a different language:

- *Persian* The official language of Iran, a country with a population of 60 million. This language is also spoken in Tadzhikistan (of the Commonwealth of Independent States) and by approximately half the population of Iraq.
- *Pashto* The official language of Afghanistan, which is also spoken by millions of people in Pakistan.
- *Kurdish* A language spoken by several million individuals in parts of Eastern Turkey, Northern Iraq, and Western Iran.
- *Urdu* A language which is very similar to Hindi in ordinary conversational use. Urdu is written in a modified form of the Arabic alphabet. Some 200 million individuals in India speak a variety of Hindi and Urdu.

The extension of the current work to these other languages would similarly involve the combined use of knowledge-based systems and neural networks to capture the rules of vowelization and exceptions to rules.

It would also be possible to apply the work to matching the names generated by this method to the list of names on a "stop list." Given an Arabic name written in Arabic, our method would automatically vowelize it and generate all of its romanized versions. Most official documents in Arabic (such as passports) contain the name of the individual in Arabic. This name could be scanned by OCR and read into the transliteration package. The matching process could be carried out by any of the primary current methods: Soundex* digraphs and "fuzzy" search.

The important feature of this approach would be the dropping of the requirement for a pre-existing database. The romanized variations of the Arabic name would be generated on line and matched against the "stop list" on a real-time basis. To implement the approach, the romanized names would have to be extended to include English, French, Italian, Spanish, and German variations. Non-standard Arabic names (Islamic names, foreign Arabic names, and corrupted Arabic names) could also be included.

Another approach to solving these problems could be the use of the fuzzy match method. This method is not what is commonly known as "fuzzy search"—an imprecise use of the term "fuzzy." As used in "fuzzy sets," the term normally refers to the mathematical procedure described in [13]. We believe the fuzzy set method applies to this problem. Following is the germ of an idea which may prove useful in making better matches:

1. Develop a membership function for each letter of the alphabet; a matrix showing how each letter is similar to another letter could be developed.
2. Develop membership functions for adjacent letters (two at a time)—a matrix showing how letters are similar to one another or to each other.
3. Evaluate the match by computing the combined membership function value.
4. The match with the highest value is the best match to the stop list. If this value is less than a threshold value, no match is made.

A fuzzy measure of the similarity between the two names "Abdel" and "Abdul," computed by summing the similarities of corresponding letters, results in a membership of 0.84 [(1.0 + 1.0 + 1.0 + 0.2 + 1.0)/5]. This membership value represents similarities between letters. For example, "E" and "O" have a degree of similarity of 0.2.

The membership function can be developed by phonetic similarities between various letters. Hence, the letter "A" has a membership value of 1.0 in the membership function for "A"; but the letter "E" has, perhaps, a 0.5 value in the membership function for the letter "A." These functions could also be developed by similarities between sound wave profiles. Neural networks can play a role in the evaluation of similarities between the two names.

Concluding remarks

The automated approach to name transliteration has rapidly become viable; it can perform much of the task with high accuracy and speed, outperforming its manual counterpart. However, some names still cannot be transliterated without manual intervention. Further study would be required to reduce the size of this set without producing too many extraneous vowelizations. Nevertheless, the study described here shows that automated transliteration is real and effective, given the appropriate tools and models.

References

1. W. B. Bishai, *A Computer Dictionary of Literary Arabic*, User's Manual, Arabic Software Center, Stoneham, MA, 1991.
2. T. El-Sadany and M. Hashish, "An Arabic Morphological System," *IBM Syst. J.* **28,** No. 4, 600–612 (1989).
3. P. Roochnik, "Computer-Based Solutions to Certain Linguistic Problems Arising from the Romanization of Arabic Names," Ph.D. Dissertation, Georgetown University, Washington, DC, 1993.
4. H. Y. Abdelazim and M. Hashish, "Automatic Reading of Bilingual Typewritten Text," *Proceedings of COMPEURO '89—3rd Annual European Computer Conference, VLSI and Computer Peripherals*, IEEE, Piscataway, NJ, Vol. II, pp. 140–144.
5. H. Goraine, M. Usher, and S. Al-Emami, "Off-Line Arabic Character Recognition," *IEEE Computer* **25,** No. 7, 71–74 (July 1992).
6. D. A. Waterman, *A Guide to Expert Systems*, Addison-Wesley Publishing Co., Reading, MA, 1986.
7. D. Bailey and L. Medsker, "Integrating Expert Systems and Neural Networks," Tutorial notes from *Proceedings*

*Soundex, developed by R. C. Russel in 1918, uses phonetic equivalence to establish name similarity (only surnames are used). Only consonants are used (excluding H, W, and Y), and each name is encoded with just four characters (a letter and three numeric characters).

of the Ninth National Conference on Artificial Intelligence, AAAI, 1991.
8. D. Hillman, "Integrating Neural Nets and Expert Systems," *AI Expert* **5,** No. 6, 54–59 (1990).
9. COSMIC Program No. MSC-21916, MSC-21927, MSC 21929, MSC 22078 CLIPS Version 5.1, COSMIC, 382 E. Broad St., Athens, GA 30602 (404-542-3265).
10. S. Fahlman and C. Lebiere, "The Cascade-Correlation Learning Architecture," *Advances in Neural Information Processing 2*, David Touretzky, Ed., Morgan Kaufmann, San Mateo, CA, 1990, pp. 524–532.
11. D. Rumelhart, G. Hinton, and R. J. Williams, "Learning Internal Representation by Error Propagation," *Parallel Distributed Processing*, D. Rumelhart and J. McClelland, Eds., MIT Press, Cambridge, MA, 1986, Vol. 1, Ch. 8.
12. W. M. Frazier, "A Demonstration of a Computer Technique for Investigating Arabic Morphology," Ph.D. Dissertation, University of Michigan, Ann Arbor, 1976.
13. L. Zadeh, "Fuzzy Logic," *IEEE Computer* **24,** No. 4, 83–91 (April 1988).

Received February 19, 1993; accepted for publication August 6, 1993

Mansur Arbabi Dr. Arbabi retired from the IBM Federal Systems Company facility in Gaithersburg as a Senior Member of the Technical Staff. He received a B.S. degree in electrical engineering from California State Polytechnic University, an M.A. degree in mathematics from American University, and a Ph.D. in operations research from Johns Hopkins University, with distinction. He joined IBM in 1958. In 1963, he received an IBM Outstanding Contribution Award for the first-ever computer aircraft flight planning capability; he later received an IBM Resident Study scholarship, under which he completed his doctoral studies. Dr. Arbabi organized the first technical teams for the investigation of artificial intelligence and neural network applications at the Federal Systems Company. He has published numerous papers and has taught courses in the fields of operations research, artificial intelligence, mathematical programming, and neural networks.

Scott M. Fischthal *IBM Federal Systems Company, AI Technology Center, 800 North Frederick Road, Gaithersburg, Maryland 20879 (fischthal@vnet.ibm.com).* Mr. Fischthal received a B.S.E. degree in electrical engineering and computer science from Princeton University in 1985 and an M.S. in computer science from Johns Hopkins University in 1990. He is currently a Ph.D. candidate in information technology at George Mason University. He joined IBM in 1985 and has worked in the AI Technology Center since 1987. His research is primarily in the fields of neural networks and machine learning. The author of numerous papers, Mr. Fischthal has lectured and taught classes in neural networks at Johns Hopkins University, IBM, and the Woodrow Wilson School of Government at Princeton University. He is a member of the Association for Computing Machinery, the Institute of Electrical and Electronics Engineers Computer Society, the American Association for Artificial Intelligence, and the International Neural Network Society.

Vincent C. Cheng *IBM Federal Systems Company, AI Technology Center, 800 North Frederick Road, Gaithersburg, Maryland 20879 (VINCENTC at WMAVM7).* Mr. Cheng is a Senior Associate Programmer in the Artificial Intelligence Technology Center (AITC) of the IBM Federal Systems Company. He received an M.S. degree in computer science from the University of Maryland, Baltimore County, in 1989. He subsequently joined IBM at the AITC, where he initially worked on The Integrated Reasoning Shell (TIRS). His main interest is in natural language processing. Mr. Cheng is a member of the American Association of Artificial Intelligence and the Association of Computational Linguistics.

Elizabeth Bart *IBM Federal Systems Company, AI Technology Center, 800 North Frederick Road, Gaithersburg, Maryland 20879 (BARTB at WMAVM7, bartb@vnet.ibm.com).* Miss Bart is a Staff Engineer in the Artificial Intelligence Technology Center (AITC). She joined IBM at the AITC in 1987, after receiving a B.S. in computer engineering from Case Western Reserve University. She received an M.S. in computer science from Johns Hopkins University in 1992, where artificial intelligence topics were the focus of her studies. Her current work is primarily in knowledge-based-system development. She is a member of the American Association for Artificial Intelligence.