MATRIX LEXICA: AN ALTERNATIVE DESCRIPTION OF LEXICAL DATABASES

E. Papakitsos, M. Gregoriadou

SUMMARY

In the work presented here, new methods for designing and implementing large lexical databases were examined. These lexical databases or machine readable dictionaries are expected to be organized in a way to provide fast access to the stored data and efficient memory management. Directed graphs can be used to describe and organize a lexical database of large magnitude in a compact manner. These data structures are called matrix lexica, where the letters are described as nodes of directed graphs and the lemmata as paths (set of edges). It is claimed that matrix lexica can efficiently support automated language applications, in the fields of lexicography, terminology, machine-translation and others, by providing high speed of resolution, sound mathematical foundation, low memory requirements and ability to handle distorted input in future developments. These methods were evaluated for Modern Greek as a target language.

0 INTRODUCTION

The overall target of the work presented here was the development of a general purpose automated system for the computational treatment of Modern Greek morphology. Such a computerized system is composed of two major subsystems:

(i) the subsystem which analyze the words, called "tagger" and

(ii) a database, containing information about the words, which is called "lexical database" or "lexicon".

It is expected from a tagger to provide one accurate analysis for every analysed word fast and with low complexity (in order to impove maintainability). The model of functional decomposition [1][4] was used to design and implement a tagger for Modern Greek and evaluate its performance using a large scale corpus (ECI-Greek Part), containing approximately 1,880,000 words. This tagger used a morpheme based lexicon having 7800 entries. The morpheme based lexicon is a lexical database that contains morphemes instead of words. The advantages of such a lexicon are: low computer memory requirements ability to simulate the natural mechanisms of language. The second advantage (ii) gives to the tagger the ability of analyzing novel words (neologisms) or words that usually are not included in other types of traditional dictionaries. The initial size of this lexical database includes 7800 entries that are morphemes of the following classes: free morphemes (1669), roots (5758), inflectional suffixes (149), suffixes and prefixes (approx. 200). A computerized system having the above mentioned features requires advanced storing and accessing methods in organizing the lexicon.

1 LEXICAL DATABASES

For supporting a tagger, two of the most frequently used data structures (and accessing methods) are hash tables [5] and tries or root lexica [3]. The design of the lexical database for our tagger followed the hash-table method. The entries are initially distributed in 1000 positions where collision occurs for up to 100 cases. Finding a better hashing-function was not a priority of the development although better hashing functions are available. Twelve data arrangements and searching algorithms were tested (serial, binary, block search and hashing upon arrays, lists and binary trees) before concluding to the superiority of hashing, which performs 150% faster than the second candidate method (array's binary search).

On the other hand, hashing wastes memory and maybe it is not the best way to deal with spelling errors, the appearance of the errors being significant in the corpus (2%). Tries also waste memory and provide a lower overall speed, but recent work on Finite State Automata (FSA) has demonstrated an ability to deal with most kinds of spelling errors [2], in a potentially effective manner. One target of this research was to find a database design which can probably combine the advantages of both hash tables and tries, without the corresponding drawbacks. Namely, a data structure and a searching method that may provide high resolution speed, minimum occupation of memory and an ability to support effectively spelling-checking. It is believed that methods based on directed graphs for organizing a lexical database (matrix lexica) can be very promising towards this direction.

2 MATRIX LEXICA

A matrix lexicon, which is designed as a directed graph, is arranged as a matrix of characters (Fig.1). The number of columns denotes the size of the alphabet and the number of rows denotes the length of a string (lemma). Every letter on a certain position of a string is a node and any string represents a unique path in the graph. A path is a set of edges starting from a node of the first row. An edge connects a node of one row only to a node of the next row from top to bottom, thus the graph is directed. The attributes of a lemma (or a pointer to them) can be found at the starting or the finishing node of its associated path.

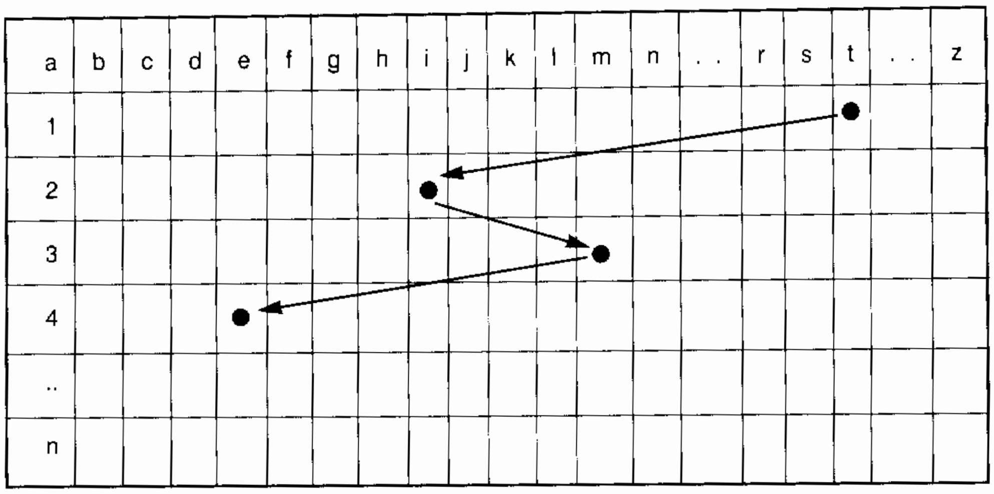

Figure 1. A matrix lexicon and the path of the word 'time'.

Due to collision, a node contains the attributes of many paths, but a path is uniquely represented by an integer which is allocated to it through some hashing function. This design can provide fast data access equal to a hash table because of the hashing function, and it can support a Finite State Automaton (FSA) in discovering a string node by node just like a trie. Additionally, it is very compact because the characters of the strings are not present (they are substituted by nodes) and the strings can be represented as two byte integers. This representation is of fixed size, which is less than half of the average size of lemmata (4.5 characters per string). Two relevant designs will be discused, the Character Combination Data Engine-CCDE [6] and the Cartesian Lexicon described in the next section.

The CCDE is proposed as a new general method of storing data without any preperation, i.e. without definning fields, their keys, indexed files, etc. CCDE can be described as a directed multigraph and it can be used to implement a matrix lexicon. It contains double the nodes of the previously described matrix, because terminal nodes (where a path finishes) are different from non-terminal nodes. Each path is marked by a degree associated with the lenght of it and with the number of common nodes with other paths. Because paths share common nodes and edges with other paths, each node contains information about the accepted and not accepted second-precedent edges. This happens in order to avoid generation of non-existent strings if the previous node is common with another path. The above information consists of the minimum and maximum degree of the accepted edges, and the minimum and maximum degree of the unaccepted edges. The degree numbers are calculated through an equation based on the combination of characters. It is claimed that this new method has the ability of remembering the stored characters.

3 THE CARTESIAN LEXICON

The "Cartesian Lexicon" can be described as a simple digraph and it is a matrix lexicon designed and implemented as part of this research, aiming at the development of lexical databases capable of supporting word-processing more efficiently. Here, the nodes are considered as points of a two dimensional space, where each node is identified by a pair of Cartesian coordinates (hence Cartesian Lexicon). A path is identified by its lenght, starting from the [0,0] point which is outside the matrix of letters (i.e , the 'a' of the first row is identified by [1,1] coordinates). A node can be terminal or not or both. If it is terminal, the features of all the associated paths ending there, are stored in some kind of list or array structure initiating there. For example, the strings 'time', 'lake' and 'line' have a length of 4 characters with the last one being 'e'. Their features will be stored in a list located at the 4th row and the 5th column (Fig.1: [4,'e']), since all their paths finish there. The features of each path are distinguished by allocating to them an integer characterizing the path. This integer can be the length of the path or the result of a hashing function, also denoting the position within the array where the features are stored. Taking the above example ('time','lake','line'), such a hashing function can use all the characters of those strings except the last one ('tim', 'lak', 'lin') which is common to all. In accessing the data after receiving an input string (eg. 'time'), the list containing its features is immediately located through the string's length (4) and the last character ('e'). Because the features of the other strings (eg. 'lake', 'line', etc) will be also located there, the actual position is computed by the hashing function. If the input string is somehow distorted then its features will not be found in the searched list and the correcting procedure will be invoked. For this purpose, a node is also associated with a 24-bit array denoting the acceptable edges connecting the nodes of the previous row to the current node. The correcting procedure will trace the path of the input string forwards (eg.'t'-'i'-'m'-'e', as it would have happened in a trie) or backwards (eg. 'e'-'m'-'i'-'t') through the nodes, to discover the error and to find eventually the correct location of the string's features.

Several configurations of the Cartesian Lexicon were evaluated in preliminary tests, differing in few details. Some of its discovered characteristics, using Modern Greek as the target language, were that lists may contain from 0 to 250 elements. The centrally located ones are the most populated with elements. A collision rate of up to 10% was observed, depending on the hashing function, but the optimization is yet an open subject.

4 APPLICATIONS

Except from spelling error correction, which was presented previously, a morpheme based lexicon can support the computerized morphological processing system in a variety of applications, such as automated lexicography and information retrieval, machine translation, handling of newly appearing terms, and others. A few examples are given below:

In information retrieval and automated lexicography applications, it is required by the computerized system to find words that have a particular relation with a given one. This is the case of derivatives and compounds. Given the word "χρόνος" it is expected from the computerized system to discover derivatives like "εκσυγχρονίζω". This is achieved because in the lexicon the roots ("χρον-") are connected through pointers to all their associated prefixes ("εκ-", "συν-", etc.) and to all their associated suffixes (like "-ιζω"). Thus their combinations, forming the derivative words of "χρόνος" like "εκσυγχρονίζω", "αναχρονιστικός", "διαχρονικός", etc, are accessible without being explicitly stored in the lexicon.

In machine translation applications, every morpheme is characterized by a unique number, which is called "key-number". The translation process can be achieved through a mapping of key-numbers in a "translation" file as follows:

[Greek Lexicon: 'ανθρωπ-' =4112]

["Translation" file: 4112=6087]

[English Lexicon: 6087='man'-Noun].

Machine translation is of course much more complex than the above example as it also requires a mechanism to simulate the environment, since a word of the source language can be mapped to many words of the target language depending on the environment.

The last given example is the ability of this computerized system to analyse new terms such as the words "διαδίκτυο" (internet) or "τηλεμετάδοση" (long-distance transmittion), which are not recognized by other systems, unless they are explicitly stored. The morpheme-based lexicon contains the morphemes "δια-", "δικτυ-", "-ο", "τηλε-", "μετα-", "δοσ-", "-η", and others. New words are generally made by different combinations of the finite number of the existent morphemes of a language. In that way, the morpheme-based lexicon does not have to explicitly contain the above new terms in order to analyse them, because the tagger is able to discover their constituent morphemes.

ACKNOWLEDGEMENTS

Thanks go to As.Prof. A. Ralli for her contribution in the linguistic part of this research and to Prof. G. Philokyprou for his comments and suggestions which improved the material presented here.

REFERENCES

[1] Allen J.,Hunnicutt M.S. and Klatt D. (1987), **"From Text to Speech:The MITalk System"**, Cambridge University Press.

[2] Ferro M.V.,Gil J.G.,Alvarino P.A. (1996), **"Finite state morphology and formal verification"**, Natural Language Engineering 2(4):303-304, Cambridge University Press.

[3] Knuth D. (1973), **"The Art of Computer Programming"**, Vol.3: Sorting and Searching, Reading, Mass: Addison-Wesley.

[4] Sproat R.W. (1992), **"Morphology and Computation"**, MIT, USA.

[5] Tremblay J.,Sorenson P. (1984), **"An Introduction to Data Structures with Applications"**, McGraw-Hill.

[6] Γελάτος Δ., Ζήβελδης Α. (1992), **"Νέος τρόπος αποθήκευσης πληροφοριών σε βάσεις δεδομένων"**, Ενημερωτικό δελτίο ΕΠΥ, Τεύχος 45, Αθήνα.

Ευάγγελος Παπακίτσος

Καθηγητής Μ.Ε.

Δημ. Ράλλη 28, 111 44 ΑΘΗΝΑ

Μαρία Γρηγοριάδου

Επίκουρη Καθηγήτρια

Πανεπιστήμια Αθηνών, Τμήμα Πληραφορικής, 157 71 ΑΘΗΝΑ.