The Design and Implementation of a Legal Text Database[⋆] [⋆⋆]

Diomidis Spinellis

SENA S.A.
Kyprou 27
GR-152 37 Filothei
Greece

Abstract. We describe the design and implementation of a legal text database. The database of provides a number of Greek Council of State decisions in the form of a computer-accessible medium (CD-ROM). A graphical front-end is provided which allows the rapid retrieval of cases based on arbitrary keywords combined using boolean operators. The database was populated by automatically converting the word-processor files into a random text retrieval data structure. The system has been designed and implemented with goals of wide availability, accessibility, extensibility, and user-friendliness.

1 Introduction

The supreme administrative court in Greece is the Council of State. It was established in 1928 after the model of the French Counceil d' Etat. Except for its advisory function with regard to delegated legislation, the Council of State is primarily (unlike perhaps its French prototype) a court of law. It is an administrative court of first and last instance with jurisdiction over applications for judicial review "petitions for annulment" of administrative acts for violation of law or abuse of discretionary power. It is also the supreme court which decides final appeals against judgments of the lower administrative courts [3]. The decisions of the Council of State carry significant weight, and are important to legislators, judges, lawyers, and civil servants. Up to 1983 each year's decisions were printed and distributed by the Council of State. The constantly increasing volume of decisions made such an endeavour impractical and currently the only official way supported for access to all decisions is through the archiving department of the Council of State. We decided to implement a text database of Council of State decisions to provide a widely accessible, practical, and user-friendly platform for their dissemination.

[⋆] Dimitris Karagiannis Editor, *DEXA 94: 5th International Conference on Database Expert Systems Applications*, pages 339–348, Athens, Greece, September 1994. Springer Verlag. Lecture Notes in Computer Science 856.

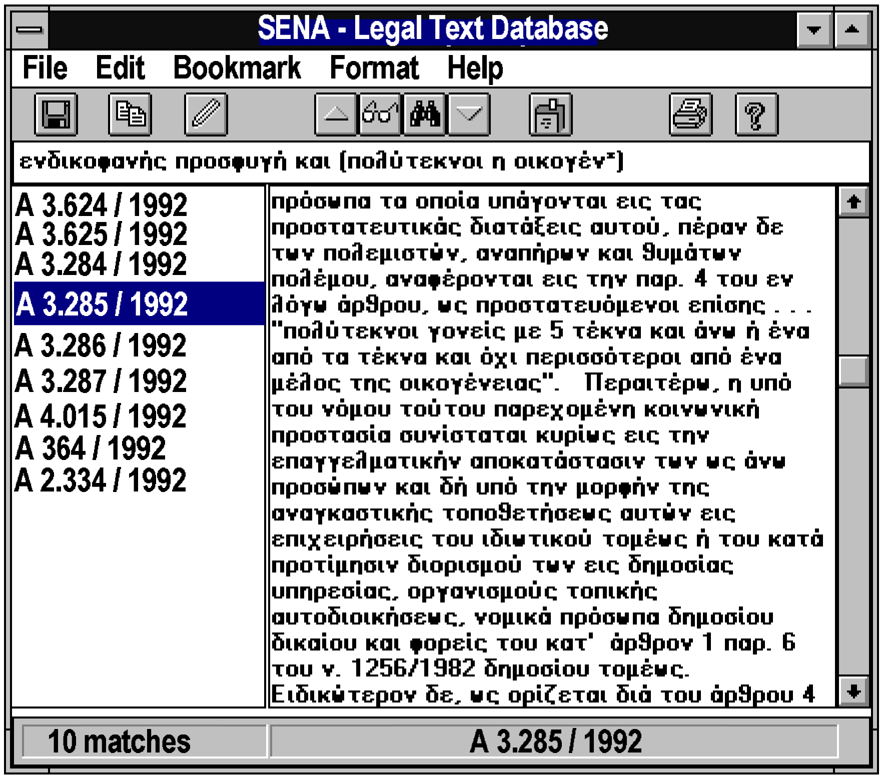
[⋆⋆] This is a machine-readable rendering of a working paper draft that led to a publication. The publication should always be cited in preference to this draft using the reference in the previous footnote. This material is presented to ensure timely dissemination of scholarly and technical work. Copyright and all rights therein are retained by authors or by other copyright holders. All persons copying this information are expected to adhere to the terms and constraints invoked by each author's copyright. In most cases, these works may not be reposted without the explicit permission of the copyright holder.

2 Functional Description

2.1 Retrieval Software

The database is stored on a CD-ROM. The retrieval software provides a user interface that runs under the Microsoft Windows graphical environment (Fig. 1), although front-ends for text-based operating environments such as Unix and MS-DOS are also available. The interface window contains the following items:

- the menu with the database commands,
- the *tool bar* providing rapid access to the most commonly used operations,
- the query area,
- the result selection area,
- the text viewing area, and
- the *status bar* containing query status and navigation information.



ενδικοφανής προσφυγή και (πολύτεκνοι η οικογέν*)

A 3.624 / 1992
A 3.625 / 1992
A 3.284 / 1992
A 3.285 / 1992
A 3.286 / 1992
A 3.287 / 1992
A 4.015 / 1992
A 364 / 1992
A 2.334 / 1992

πρόσωπα τα οποία υπάγονται εις τας
προστατευτικάς διατάξεις αυτού, πέραν δε
των πολεμιστών, αναπήρων και θυμάτων
πολέμου, αναφέρονται εις την παρ. 4 του εν
λόγω άρθρου, ως προστατευόμενοι επίσης . . .
"πολύτεκνοι γονείς με 5 τέκνα και άνω ή ένα
από τα τέκνα και όχι περισσότεροι από ένα
μέλος της οικογένειας". Περαιτέρω, η υπό
του νόμου τούτου παρεχομένη κοινωνική
προστασία συνίσταται κυρίως εις την
επαγγελματικήν αποκατάστασιν των ως άνω
προσώπων και δή υπό την μορφήν της
αναγκαστικής τοποθετήσεως αυτών εις
επιχειρήσεις του ιδιωτικού τομέως ή του κατά
προτίμησιν διορισμού των εις δημοσίας
υπηρεσίας, οργανισμούς τοπικής
αυτοδιοικήσεως, νομικά πρόσωπα δημοσίου
δικαίου και φορείς του κατ' άρθρον 1 παρ. 6
του ν. 1256/1982 δημοσίου τομέως.
Ειδικώτερον δε, ως ορίζεται διά του άρθρου 4

10 matches
A 3.285 / 1992

Fig. 1. Text Retrieval Interface Window

query	$\rightarrow$	*orexpr*
orexpr	$\rightarrow$	*andexpr* $\mid$ *orexpr* **or** *andexpr*
andexpr	$\rightarrow$	*basic* $\mid$ *andexpr* [**and**] *basic*
basic	$\rightarrow$	**not** *basic* $\mid$ **word** $\mid$ **(** *query* **)**

Table 1. Query language BNF grammar

In a typical session, the user performs a search by entering a query in the query area. The query can consist of target words connected by the *and, or, not* boolean operators, and grouped using brackets. A BNF description of the query language is given in Table 1. Words can be terminated by the wildcard character * to denote that any word starting with the characters specified is to be matched. In addition, the search is not sensitive to character case, and the Greek stress and diairesis character modifiers. Apart from the above no other phonetic or semantic equivalences [8] are taken into account, nor does our system ignore certain ‘stop-words’: we found that our users were often confused by more complicated matching schemes. The terminating wildcharacter option is a simple, yet powerful way to solve the problem of words found with multiple endings — a common phenomenon in the highly inflective Greek language.

When the query is processed the system will report the number of matching texts. A text is defined as a single Council of State decision. Typical texts are between 4K and 40K in length — a search granule convenient both for the end-user, and for searching, storing and reporting. When an excessive number of matches is found, the report list is pruned to a convenient length. The user can then select a case from the report list by double clicking on its code number, and the full text of the case appears on the text viewing area.

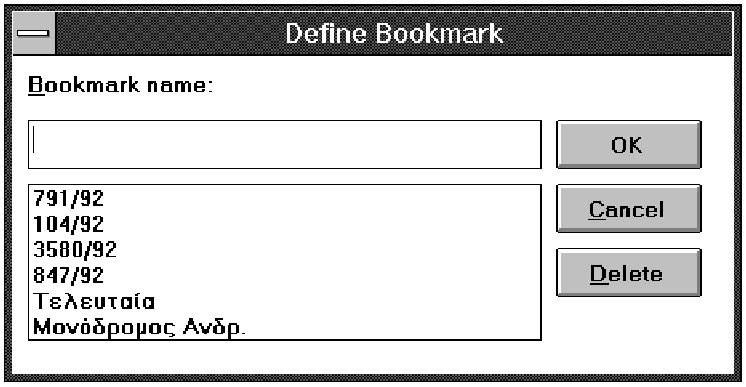
The user can the scroll through the text, search for a word within the text, or quickly move to previous or next matching texts using the tool-bar arrow buttons. The text found can also be saved in a file using a variety of formats, printed, or stored to the clipboard — the Windows common area for user-specified inter-process communication.

Although the database is stored on a read-only medium, its contents are not static. The user can define *bookmarks* throughout the database (Fig. 2) to specify specific points of interest which can then be visited by selecting their name from the bookmark menu. Furthermore, a user can add an *annotation* to a case decision. An annotation (Fig. 3) is a text entered by the user (either directly, or by pasting text from the clipboard) that corresponds to a specific case. Whenever that case is retrieved the user can also read its accompanying annotation.

The system provides a hypertext-structured, context-sensitive help system to guide the user through its operation.

2.2 Database Construction Tools

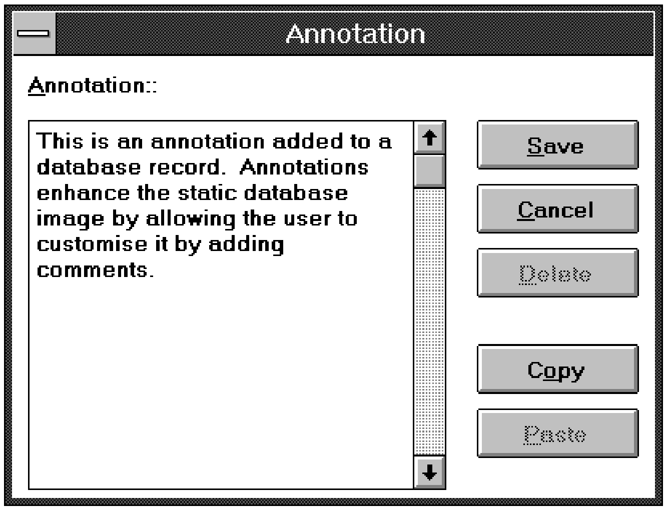
The database construction tools are used to populate the database using the case decision texts. The conversion of the case decision texts into the database is performed only


Define Bookmark

Bookmark name:

791/92
104/92
3580/92
847/92
Τελευταία
Μονόδρομος Ανδρ.

Fig. 2. Bookmark Definition Window


Annotation

Annotation::

This is an annotation added to a database record. Annotations enhance the static database image by allowing the user to customise it by adding comments.

Fig. 3. Annotation Window

once for every issue of the database by an expert operator; therefore a simple command-line-based interface guides the indexing operation. The conversion is performed in three phases:

1. Texts are converted from the word processor format used by the Council of State clean-writting department into plain ASCII format and concatenated into a single file separated by a special record separator character.
2. The text file is scanned and the following output is generated:
   - a dictionary of all words contained in the database (the *lexicon* file) together

  with their frequencies,
  - indices for every separate record of the file, and
  - the memory requirements for storing the data structures of the next phase.
3. The text file is scanned for a second time building the search structure that is used by the random text retrieval algorithms. The text file together with the random text retrieval search structures and a domain definition file form the released database.

3 System Design

The system consists of the database population tools which are used to create a database distribution and the database access system which is used by the end-users. The database access system is also split between the front-end user interface, and the back-end search engine.

The database population tools perform the processor and memory intensive task of creating the random text retrieval data structures. These data structures are based on an *inversion* of the text data whereby every word points to the documents that contain it. The method used for the inversion is a modified implementation of the one described in [9]. During the inversion a compressed dictionary and word index data structure is created in *main memory*, eliminating time-consuming access to disk. This is very important, as traditional techniques such as those described in [7] require large amounts of temporary space and multiple random passes over the data, resulting in slow inversion times. The amount of data generated by the Council of State is about 300MB every decade, and is constantly increasing. For this reason an efficient inversion process in both time and space was selected.

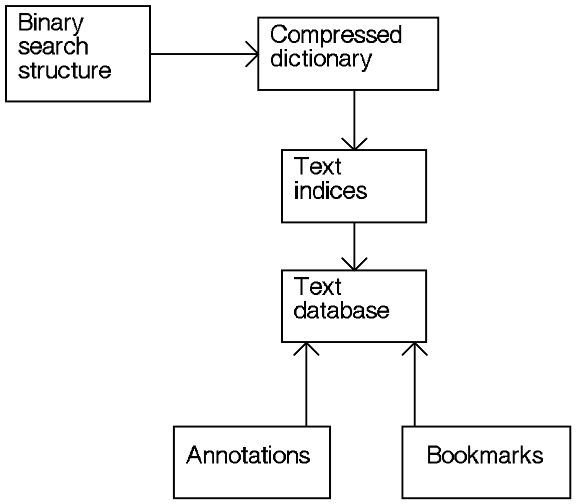


Fig. 4. Database file structure relational diagram

The database access system front-end is structured as a series of procedures associated with events generated by the windowing system. Each event (such as a button click, a menu command, or a text selection) triggers the appropriate procedure which handles the event and on completion returns control back to the windowing system. All low level interaction, such as the processing of scrollbar events, the scrolling of the text, and the editing of the query is automatically handled by the windowing system. The front-end also maintains the dynamic — user-defined — part of the database (bookmarks and annotations), as a parallel superimposed structure to the static part (Fig. 4).

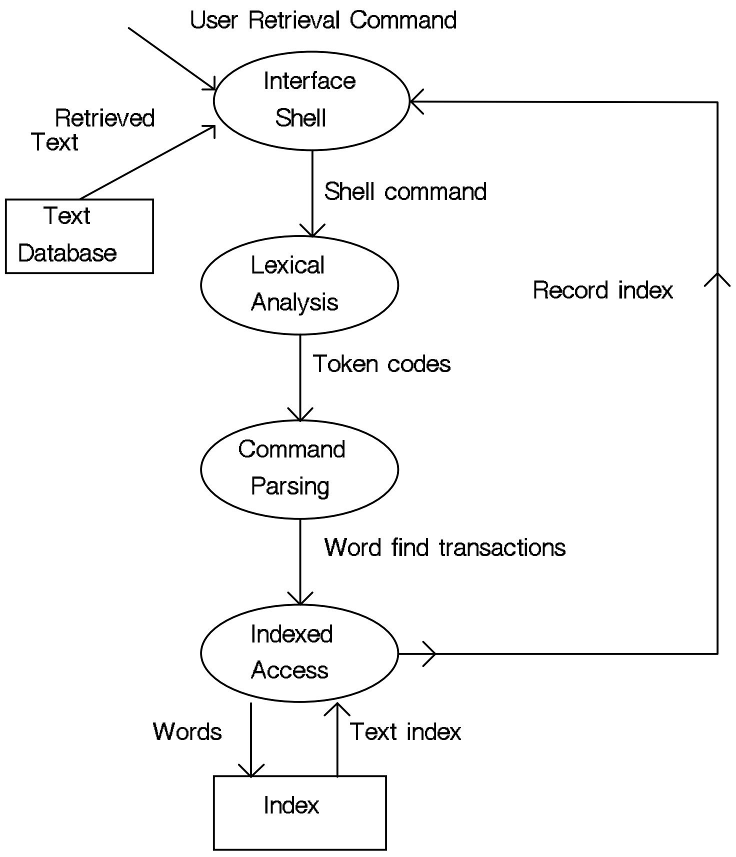


Fig. 5. System data flow diagram

The database access system back-end receives the queries from the front-end (Fig. 5), performs lexical analysis to split them into words and operators, and parses the query into an expression of search terms and logical connectives. The database is searched for every term and an appropriate *match vector* containing a map of the search hits is generated as a result of that search. The *match vectors* are combined and evaluated using boolean algebra rules according to the user-entered logical connectives to form the final *match vector* which is the result of the query. The result is maintained as state within the back-end, and a separate set of functions are used by the front-end to access the *match vector* as an opaque type. The words are searched by performing a binary search based on a structure that points to a compressed word dictionary, which in turn is used to index the specific text records (Fig. 4). The compressed search structures (described in detail in section 4.1) are important for minimising the query response times performed using the database's CD-ROM distribution medium by clustering data together eliminating expensive (150ms) disk seek operations [10, p. 218].

4 Implementation Details

4.1 Database Population

The conversion of the texts from the word processor format into ASCII is done by a small filter program prototyped in the Perl programming language [11], and subsequently ported into C for efficiency reasons. In the test run of the system three years of data comprising 190MB of word processor data were processed to create a single 53MB file of ASCII text.

The first pass of the inversion process scans the database building a word dictionary containing the number of documents that contain each word. That number is used in the second phase to create an efficient prefix encoding method [5, 4] based on a maximum guaranteed *gap* measure between successive word occurrences. Specifically, according to [9], given that the database contains N documents and a word appears in p of them, we can encode the word gaps using a b-block code in the most efficient way if we use as b_w (the code's prefix constant) the following:

$$b_w = 2^{\lfloor \log_2 \frac{N-p}{p} \rfloor}$$

In addition, the number of bits B_w needed to encode all the word gaps for a given word using a block code b (the word's B_w) is then:

$$B_w(b) \leq p \cdot (1 + \log_2 b) + \frac{N-p}{b}$$

The first phase creates the dictionary containing the B_w for every word, so that in the second phase this data structure can be filled by the gaps between the word occurrences. The main memory required by the second phase is thus given by $\sum_{i=1}^{N} B_{w_i}$. The results of the first pass on the trial data appear in Table 2. It is apparent that the main memory requirements (which are the only additional storage requirements apart from the text data) are quite modest.

The second phase of the inversion process reads the dictionary file and scans the data text building a search structure which contains:

- number of words,
- size of the compressed word dictionary,

Number	Description
6,250	documents
51,143,208	number of characters
6,853,722	number of words
100,510	number of different words
835,054	bytes size of all distinct words
32	bytes size of largest word
483,660	bytes in compressed word dictionary
14,614,289	bits upper limit on memory usage

Table 2. Results of the first inversion phase

- size of the compressed gap bitvector (in bytes),
- number of documents,
- offsets into the compressed word dictionary,
- $(p_{w_i}, \sum_{j=1}^{i} B_{w_j})$ pairs for every word,
- byte array containing $\log_2 b$ for every word,
- "1 in K" compressed word dictionary,
- document gap bit encodings, and
- document indices.

The "1 in K" compressed dictionary is used to spare memory by only storing words in groups of K words as prefix/suffix differences from the first word in the group. In our exemplar data this encoding achieved a 44% decrease in dictionary storage space with $K = 4$. The second phase works by maintaining for every word a bit pointer r into the document gap bit encoding bit vector. For every word i in the document, its gap from the previous document is encoded and appended to the bit vector at the position $r + \sum_{j=1}^{i} B_{w_j}$.

4.2 Query Processing

Queries are first translated into tokens by a simple hand-crafted lexical analyser based on a state machine. The query is then processed by a recursive descent parser [1, p. 181] which simultaneously evaluates the query by performing word searches and combining the *match vectors* described in section 3 using logical operators on bit vectors. The result of the query is then stored in a single bit vector and can be accessed by two functions: one returning the number of matches, and the other the document index for a given match.

Words are searched by performing a uniform binary search [6, p. 411] in the compressed dictionary. The search comparison function is modified to scan forward K words to deal with the dictionary's compression. Furthermore, the string comparison function is modified to treat strings ending with the wildcard character as equal to strings with the same prefix. After a matching word is found in the dictionary the ordinal numbers of the documents in which the word occurs can be easily calculated by successively adding the word's gap measures decoded from the word gap bit vector structure.

4.3 Implementation Metrics

The implementation effort of the system was relatively modest at about 5000 lines of code. The relative size of each part of the system is summarised in Table 3.

System Part	Lines of Code
Document conversion	124
Inversion first pass	452
Inversion second pass	915
Query processing engine	1001
User interface shell	2570

Table 3. Implementation of system parts

For the text data described in Table 2 the first pass of the inversion process (lexicon construction and word frequency counting) took 8 minutes, while the second pass (search structure creation) took 27 minutes. It is apparent that our process will still terminate in manageable execution times even for an order of magnitude more data (both passes use $O(\log_2 n)$ algorithms where n is the number of distinct words in the text).

5 Related Work

Legal professionals depend on accurate and complete access to legislative and case texts. Their needs are targeted by a number of commercial data providers at the national and international level. Most databases are accessed via public switched networks, although some are now distributed in CD-ROM format [2]. In Greece two companies, Databank and HellasLex, sell legal database services — including court decisions — accessible via dial-in modems. Our implementation differs in scale and approach: it can be considered as a CD-ROM publication of recent Council of State decisions.

6 Conclusions and Further Work

We have presented the design and implementation of a Legal Text Database. Using a small set of well researched data structures and algorithms together with an industry-standard windowing interface we were able to construct an efficient, user-friendly and responsive system that satisfies the needs of many legal professionals.

We are currently expanding the system adding more cases, and examining other sources of legal texts that can be incorporated into our system. The generality of our approach based on random text retrieval allows the modular and effortless addition of arbitrary texts while still providing the professional with a reasonably structured interface for retrieving the information he or she requires.

Acknowledgements

We would like to thank the President of the Council of State Mr. B. Botopoulos, the Associate Judge Mr. B. Aravantinos for their invaluable support of this project, and the EDP operator Mr. B. Koliopoulos for the technical assistance he provided.

References

1. Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers, Principles, Techniques, and Tools*. Addison-Wesley, 1985.
2. Chadwyck-Healey Ltd. New CD-ROM: EUROCAT. *Eur-OP News*, 3(2):3, October 1993.
3. Prodromos D. Dagtoglou. Constitutional and administrative law. In Konstantinos D. Kerameus and Phaedon J. Kozyris, editors, *Introduction to Greek Law*, chapter 3, pages 21–91. Kluwer, second edition, 1993.
4. R. C. Gallager and D. C. Van Voorhis. Optimal source codes for geometrically distributed alphabets. *IEEE Transactions on Information Theory*, 21(2):228–230, March 1975.
5. S. W. Golomb. Run-length encodings. *IEEE Transactions on Information Theory*, 12(3):399–401, July 1966.
6. Donald E. Knuth. *The Art of Computer Programming*, volume 3 / Sorting and Searching. Addison-Wesley, 1973.
7. Michael Lesk. Some applications of inverted indexes on the Unix system. In *Unix Programmer's Manual*, chapter Volume 2A. Bell Laboratories, 1988.
8. Michael Lesk. Word manipulation in online catalog searching: Using the UNIX system for library experiments. In *Proceedings of the EUUG Spring 88 Conference*, pages 135–147, London, April 1988. European UNIX User Group.
9. Alistair Moffat. Economical inversion of large text files. *Computing Systems*, 5(2):125–139, Spring 1992.
10. Ken C. Pohlman. *The Compact Disc Handbook*. Oxford University Press, 1992.
11. Larry Wall and Randal L. Schwartz. *Programming Perl*. O'Reilly and Associates, Sebastopol, CA, USA, 1990.

This article was processed using the LaTeX macro package with LLNCS style