ELECTRONIC WORKSHOPS IN COMPUTING

Series edited by Professor C.J. van Rijsbergen

Rainer Manthey and Viacheslav Wolfengagen (Eds)

Advances in Databases and Information Systems 1997

Proceedings of the First East-European Symposium on Advances in Databases and Information Systems, (ADBIS'97), St Petersburg, 2-5 September 1997

Multiway-Tree Retrieval Based on Treegrams

Hans Argenton and Ulrich Guntzer

Published in collaboration with the
British Computer Society

©Copyright in this paper belongs to the author(s)

ISBN: 3-540-76227-2

Multiway-Tree Retrieval Based on Treegrams

Hans Argenton, Ulrich Guntzer

Wilhelm-Schickard-Institut für Informatik

Universität Tübingen

Sand 13, 72076 Tübingen, Germany

e-mail: {argenton, guentzer}@informatik.uni-tuebingen.de

Abstract

Large tree databases as knowledge repositories become more and more important; a prominent example are the treebanks in computational linguistics: text corpora consisting of up to five million words tagged with syntactic information. Consequently, these large amounts of structured data pose the problem of fast tree retrieval: Given a database T of labeled multiway trees and a query tree q , find efficiently all trees $t \in T$ that contain q as subtree. This paper presents a generalization of the classical n -gram indexing technique for supporting fast retrieval of multiway tree structures: *Treegram indexing* covers database trees with subtrees of fixed height; each entry of the resulting index represents such a subtree together with the database trees that contain this subtree. The evaluation of a given query q preselects those database trees that contain all of q 's cover trees and, in turn, tests these candidates rigorously for containment of q . As an application of treegram indexing, we describe the VENONA retrieval system, which handles the BH^t treebank containing 508,650 phrase structure trees found in the morphosyntactical analysis of The Old Testament with altogether 3.3 million wordforms—results of a computational-linguistics project at the Ludwig-Maximilian's University of Munich.

1 Tree Retrieval

Nowadays, knowledge repositories grow rapidly along two dimensions: First, the volumes of data they contain become larger and larger; second, the data themselves become more and more structured as they convey information of increasing complexity. A typical example are the databases used for the linguistic analysis of large text corpora: They comprise a wealth of structured and connected information on different levels of description: data for phonology, morphology, syntax, semantics, and pragmatics. Multiway trees play a central role in representing this complex information because they are a common and well-understood data structure for describing hierarchies, such as the interlocking of syntactical constituents. One of the most well-known linguistic tree repositories is the Penn treebank of the University of Pennsylvania: Its fundament consists of a corpus containing 4.5 million words of American English; half of this corpus has been annotated for skeletal syntactical structure, cf. [11].

Large volumes of complex data raise again the problem of fast retrieval: Whereas information retrieval traditionally deals with unstructured information, retrieval techniques for highly structured data now become essential. In this article, we investigate the efficient retrieval of multiway tree structures at the cost of complex indexes—which means to adapt techniques of traditional information retrieval to information conveyed by structure rather than content. We concentrate on the following problem: *Given both a database of labeled multiway trees and a query tree, find all trees in the database that contain the query as a subtree*, where the notion of containment requires for each node of the query and its counterpart within the database tree both identical labels as well as identical child positions wrt. their parents.

To tackle this problem, we propose a generalization of the classical *n*-gram index ([16], [4]), which is used for supporting substring search in text databases: This method indexes a text according to the substrings of length n

contained in that text—its n -grams. The processing of a substring query involves two steps: First, each database text that contains all the n -grams of the query is selected as candidate for the answer. Then, each candidate is tested for containment of the whole query string. Although the containment test is a costly operation compared to index look-up, the number of candidates will generally be much smaller than the number of database texts considered—which justifies the costs of the index generation.

To generalize this technique for trees, we use subtrees of fixed height, the *treegrams*, in place of substrings of fixed length. This generalization to subtrees brings up a number of interesting questions: First, each node of a cover tree bears a label string in contrast to the n -gram case where each element only consists of a character; consequently, the indexer may find a very large number of different cover trees. Second, multiway trees have no limit for their number of sons: This too may render the number of different kinds of cover trees infeasible. Third, each node of an indexed tree is root not only for one but in general very many different cover trees: A cover tree may depict only part of the structure rooting in the indexed node; this kind of treegrams is essential for processing query trees that contain *don't-care* symbols or variables for whole subtrees. Taking all the treegrams of a node into the index also increases its size a lot.

These problems are independent from our specific treegram characteristic: the fixed height. In fact, there are other obvious ways to adapt n -grams to trees—for example, instead of height, take the number of nodes as characteristic for the cover trees. We investigated this alternative for binary trees (cf. [3]); the choice of fixed height, however, is more intuitive with respect to the solution of the third problem: We do not take all treegrams rooting in a node but only a few; with fixed height, there is one immediate candidate—the treegram that contains all the nodes available.

A successful tree-retrieval system has to cope with all these problems and furthermore with well-known requirements of the n -gram case: for example, to make use of the different selectivities of the occurring treegrams. We illustrate our approach to these points with the VENONA retrieval system, which is used in a computational-linguistics project of the Department for Old Hebrew Linguistics at Ludwig-Maximilian's University in Munich (cf. [13]): Part of that project is the work of the AMOS expert system (cf. [14], [15]), which analyzes The Old Testament's morphosyntax; the resulting BH^t (Biblia Hebraica transcripta) treebank comprises 508.650 phrase structure trees with maximum degree eight and maximum height 17, containing altogether 3.3 million Old-Hebrew words.

Figure 1 shows such a morphosyntax tree as part of a VENONA answer: The subtree with the black root and bold nodes represents the query for which VENONA retrieved this tree. Note that the query consists only of inner nodes of the database tree: it does not contain any selective Hebrew words; furthermore, the query does not comprise all sons of the APPV/R node on the second level of the database tree.

The rest of the paper is organized as follows: Section 2 gives a formal description of the problem we deal with, especially of the notion of containment that we use. Section 3 presents treegrams and outlines retrieval with a treegram index, Section 4 discusses the VENONA tree-retrieval system and its strategies for efficient query evaluation, and Section 5 concludes with an outlook to retrieving feature terms—a data structure that generalizes trees to directed acyclic graphs with labeled arcs.

2 The retrieval problem for positional multiway trees

The base entities of this problem are rooted multiway trees and labeled rooted multiway trees where children are distinguished by their position.

Definition 1: A rooted positional multiway tree t for a finite set of nodes V is either

- (1) the empty set, called the *empty tree*, $\emptyset$
- (2) one specially designated node of V , the *root* of t , together with a finite sequence $[t_1, \dots, t_n]$ —the *children* of t —where t_i is a rooted positional multiway tree over V_i and $\{V_1, \dots, V_n\}$ forms a partition of $V \setminus \{w\}$. t_i is the child with *position* i of t .

Definition 2: A *labeled rooted positional multiway tree* is a rooted positional multiway tree for node set V together with a label alphabet Σ and a label function $\lambda : V \rightarrow \Sigma$.

We regard an empty tree as absent; therefore we impose a **normal form** for rooted positional multiway trees requiring that the child with the highest position be nonempty. This reflects our view that two multiway trees having equal root labels and the same nonempty children at the same positions denote the same tree. Whenever we use the term *multiway tree* in the rest of the paper, it denotes a labeled rooted positional multiway tree in normal form.

Now, we state the definition of containment that forms the basis for the retrieval problem:

Definition 3: Let s and t be two multiway trees over label alphabet Σ for node sets V_s and V_t respectively. t *contains* s (written as $s \preceq t$) if there exists an injective embedding from V_s to V_t such that

- (1) nodes are mapped to nodes with identical labels and
- (2) a root of a child with position i is mapped to a root of a child with the same position.

Retrieval problem: Let $DB_\Sigma = \{t_1, \dots, t_n\}$ be a set of labeled positional multiway trees over some fixed alphabet Σ and let q be a tree having the same label alphabet. The problem is to find efficiently all trees $t \in DB(\Sigma)$ that contain q —that is, to compute efficiently the result set $R := \{t \in DB(\Sigma) | q \preceq t\}$.

The single problem $s \preceq t$ for given trees s and t states the *tree pattern matching problem*, cf. [9]. For a fixed alphabet Σ , the best algorithm known has a running time of $O(n\sqrt{m} \text{ polylog}(m))$, cf. [5], with $m = |V_s|$ and $n = |V_t|$. Unbounded alphabets cost an additional factor of $O(\log(m))$, cf. [9]; the case of label strings clearly corresponds to such an unbounded alphabet.

[7] gives an overview of other, more general classes of tree inclusion problems; we chose the classical tree pattern matching problem because in very many applications dealing with trees, the absolute position of a child is essential for characterising its role with respect to its parent. This is especially true for linguistic phrase structure trees, for which we designed the VENONA retrieval system described later. VENONA allows variables and don't-care symbols for children; we discuss this feature in Section 4. Section 5 gives an outlook to the retrieval of unordered trees and dags.

3 Multiway-tree retrieval based on treegrams

To tackle the outlined tree retrieval problem, we generalize the n-gram indexing technique: In place of substrings with fixed length, we use subtrees with fixed maximal height: *treegrams*. When we talk about treegrams, we want to give them an existence independent from a particular tree and its node set; therefore, we cannot directly define them as subtrees. Instead, we look at the equivalence classes of the mutual-containment relation $\simeq$: Let s and t be multiway trees, $s \simeq t$ iff $s \preceq t$ and $t \preceq s$. $B(\Sigma, h)$ denotes the set of these equivalence classes for all trees with height less than or equal h . For each equivalence class $C \in B(\Sigma, h)$, we choose one tree contained in C , which represents C .

Definition 7: A *treegram of height h* is the representative element of an equivalence class of $B(\Sigma, h)$. Let t be a multiway tree; $TG(t, h)$ denotes the set of all treegrams of height h contained in t .

Similar to the classical n-grams, our indexing technique relies on the following simple result due to the transitivity of the containment relation $\preceq$:

Lemma 1: Let s and t with $s \preceq t$ be multiway trees over the same label alphabet Σ , and $0 < h_s < h$ where h_s is the height of s . Then, we have:

$$TG(s, h) \subseteq TG(t, h)$$

Retrieval method: Let $DB(\Sigma) = \{t_1, \dots, t_n\}$ denote a set of multiway trees over the label alphabet Σ . For a treegram g of height h , we define $T(DB, g) := \{t \in DB | g \preceq t\}$. Assume that $T(DB, g)$ can be efficiently

Figure 1: AMOS phrase structure tree, part of a Venona answer. The string labels on the tree's leaves give Hebrew words in a scientific transliteration; inner nodes depict Grammar nonterminals. The contained query tree is marked by a black root and bold nodes. VENONA uses the xvcg grapher (cf. [10]) for the graphical presentation of the query result.

computed for a fixed h using the index relation $I_{DB}^h := \{(g, t) | t \in DB \wedge g \in TG(t, h)\}$. We compute the desired set $R = \{t \in DB | q \preceq t\}$ for a given query tree q such that q 's height is greater than or equal h as follows:

(1) Compute the set $TG(q, h)$.

(2) Compute the *candidate set* of q

$$\text{Cand}_h(q) := \bigcap_{g \in TG(q, h)} T(DB, g)$$

(3) Compute the *result set* $R = \{t \in \text{Cand}_h(q) | q \preceq t\}$

Lemma 1 states that the set $\text{Cand}_h(q)$ of those database trees that contain all the treegrams of the query q forms a superset of the database trees containing q ; the costly operation in this approach is the last containment test $q \preceq t$. The building of index I_{DB}^h is justified if in general the number of candidates will be much smaller than the number of trees in DB.

The evaluation method given above suffers from several drawbacks described in Section 1: With increasing height h , the treegrams themselves become very complex; furthermore, if we want to exploit Lemma 1 for the treegram index I_{DB}^h , then I_{DB}^h must contain every h -treegram of a database tree s . Note that there are in general many treegrams rooting in one node of a database tree because containment is an embedding. Together with complex treegrams, this may render I_{DB}^h infeasible. We address these problems in the next section.

4 Efficient query evaluation

The VENONA tree-retrieval system employs the outlined treegram-index approach for retrieval in tree databases; its main application is part of a computational-linguistics project: The Department for Ugaritic and Old Hebrew Linguistics at Ludwig-Maximilian's University in Munich analyzes The Old Testament at morphologic, morphosyntactic, and syntactic level—that means, it provides syntactic analyses for words, phrases, and sentences, respectively—cf. [13]. Within that project, the AMOS expert system (cf. [14], [15]) analyzes the morphosyntax; for that, AMOS employs a definite-clause grammar containing about 200 clauses. This grammar is still under development: At the moment, the output of the morphosyntactic analysis comprises a database of 508,650 phrase structure trees with a maximum degree of eight and a maximum height of 17; these syntax trees describe the application of the current grammatical theory to the whole text.

A typical problem in evaluating the adequacy of this grammatical theory consists in finding fast all instances of a given syntactic structure—that means, a tree q —contained in the database trees to get an overall view of q 's contexts; in particular, q may contain variables in place of subtrees or labels. If an efficient tree matcher tests every syntax tree for containment of q , presenting the first instance may take several minutes (testing all database trees takes about 15 minutes on a SUN SPARC 20); a central requirement, therefore, is to reduce this period to fifteen up to twenty seconds in the average.

If we want to employ to that purpose the treegram-index retrieval method given above, we have to address the problems stated in Section 1:

- (1) A single treegram may be very complex because both its label strings and its degree are unlimited; this leads to costly look-up operations if we store treegrams as they are.
- (2) In general, there are many treegrams rooting at a given node in a database tree: To accomodate queries with subtree variables, the index has to contain all matching treegrams for that subtree; thus, the index blows up and the retrieval process slows down.
- (3) Additionally, it is quite expensive to intersect the tree sets $T(DB, g)$ for all treegrams g contained in the query q . Although this minimizes the number of candidates, a minimal candidate set is often unnecessary: An efficient tree matcher can cope with a certain number of candidates—say 1,000—in reasonable time; thus, we have to process the treegrams according to their selectivity and stop when the current intersection falls below that matcher limit.

We discuss these points in turn:

4.1 Processing of a single treegram

VENONA addresses (1) by the following approach:

- First, node labels hash to an integer of a few bytes. This renders it impossible to state label patterns in the query; however, we considered this feature not essential for the current version: In the applications that we aim at, the node labels themselves should bear little information compared to the trees that they form. Consequently, they should contain no complex structure; whenever possible, use a subtree to that purpose—if this is not adequate, feature terms with their attribute-value pairs offer an excellent alternative. Section 5 gives an outlook to feature-term retrieval. Because our main application comprises syntax trees, most of the node labels are nonterminals, whose prefixes, infixes and suffixes are irrelevant for their meaning. A leaf, on the other hand, represents a Hebrew word: If its subdivision into parts is relevant for grammatical problems, the word itself should be the root of a subtree that expresses this inner structure.
- Second, VENONA deals only with treegrams of a maximal degree d ; if a tree is of greater degree, it will be transformed automatically to a d -ary tree. The employed algorithm is a generalization of the well-known transformation of trees to binary trees (cf. [8], p. 332). d 's value is a configurable parameter of the index-generation.
- Third, VENONA represents a treegram g as a set consisting of an integer encoding g 's structure and of g 's hashed labels—the *treegram-part set*; we describe this representation in more detail below. To look up complex treegrams efficiently, VENONA in turn applies the outlined retrieval method to an index recording for each treegram part all the treegrams containing this part.

4.2 Treegram representation

In contrast to traditional information-retrieval applications, structure is essential for tree retrieval; consequently, a central part of VENONA deals with the efficient representation and processing of the structure of treegrams. On the other hand, if label information turns out to be more selective than structure, it clearly should be preferred; therefore, Venona treats structure as yet another treegram part: This symmetric approach supports the selective evaluation of query treegram parts with only one index.

For describing a single treegram g , VENONA takes each of g 's hashed labels and combines it with the position of its corresponding node in a complete d -ary tree; an integer encoding g 's structure completes this representation. Because the number of different tree structures $TS_{\leq d, h}$ of maximal degree d and maximal height h grows doubly exponentially with h for given d , this encoding integer may become quite large; therefore, we use a bijection $R_{h, d}$ from $TS_{\leq d, h}$ to $\{0, \dots, |TS_{\leq d, h}| - 1\}$ for the efficient storage and comparison of treegram structures.

Definition 8: For fixed d , $MTS_d(h)$ is the number of different tree structures with maximal degree d and maximal height h ; $ETS_d(h)$ is the number of different tree structures with maximal degree d and exact height h .

Lemma 2: $MTS_d(h)$ and $ETS_d(h)$, respectively, are given by:

$$MTS_d(0) = 1 \tag{1}$$

$$MTS_d(h) = 1 + MTS_d(h-1)^d, \quad h > 0 \tag{2}$$

$$ETS_d(0) = 1 \tag{3}$$

$$ETS_d(h) = MTS_d(h) - MTS_d(h-1), \quad h > 0 \tag{4}$$

The empty tree, $\diamond$, is the only one of maximal or exact height 0; thus, (1) and (3) are valid. To prove (2), we notice that there is a bijection between multiway trees t of degree less or equal than d in normal form—that means with a nonempty tree at each highest child position—and multiway trees t' not in normal form whose

nodes all have exactly d children: For every node in t that has less than d children, add empty trees for the missing positions; vice versa, for each node of t' , begin from position d and cut off empty children until the first nonempty child appears.

Point (2) is valid for multiway trees of maximal height h with exactly d children for each node (empty children allowed): A nonempty tree has d children (which may all be empty) of maximal height $h - 1$. Thus, for each child, there are $MTS_d(h - 1)$ possibilities; since children are independent from another, this yields $MTS_d(h - 1)^d$ different nonempty tree structures. Adding 1 for the empty tree results in (2). Because of the bijection between multiway trees not in normal form whose nodes all have exactly d children and multiway trees of degree less or equal than d in normal form, (2) is valid for multiway trees of degree less or equal d . Finally, to get the number of all tree structures with exact height h , subtract the number of all tree structures with height less than h from $MTS_d(h)$. $\square$

In case of $d = 2$, $MTS_2(h) \approx 1.5^{2^h}$ (cf. [1]) in contrast to $2^{2^h - 1}$ if the structure of a tree t is coded as a bitvector for a complete binary tree that contains 1 at position i if t has a node at position i and 0 otherwise.

VENONA uses a ranking based on $MTS_d(h)$ that first enumerates all tree structures with maximal height $h - 1$ and then those of exact height h (cf. [2]).

Having dealt with the complexity of single treegrams, we now return to point (2) from the beginning of this section:

4.3 Prevention of index blow-up

To avoid the index blow-up caused by including in the treegram index every treegram rooting in a given node, VENONA uses only one treegram per node v : the treegram including *every* node found on the first h levels of the subtree rooted in v . This approach keeps the index small but introduces another problem: A treegram g of the query may not appear in the treegram index or its entry may not include all database trees that contain g —for example, if a leaf of the query occurs in g at a level less than h : Imagine a database tree containing only one instance of g in which the query leaf has children; in this case, the index treegram g' corresponding to g will contain at least the children's roots. Consequently, the database tree will not appear in g 's entry rendering the retrieval method incorrect.

Therefore, VENONA expands all query treegram *structures* at runtime by using yet another index that holds for each position of a complete d -ary tree all those database-treegram structures that contain a node at this position; for a given query treegram g , this expansion yields all database treegrams with a structure compatible to g . That approach keeps the treegram index small and preserves efficiency.

We have seen that VENONA applies its retrieval method in turn to subindexes—the treegram-part index and the node-position index—to cope with a large number of occurring complex treegrams. Despite these efforts, treegram look-up still is too expensive to be performed unnecessarily often; treegrams should, therefore, be evaluated according to their selectivity, and the evaluation should stop when the current number of candidates falls below a limit manageable for an efficient tree matcher in reasonable time: This is the last requirement from the beginning of this section.

4.4 Selective evaluation

The evaluation of a given query q proceeds along the following steps:

- (1) According to q 's degree and height, VENONA chooses a treegram index among those available for the tree database: The treegram height of the chosen index should be as close as possible to q 's height. The treegram degree should be greater than or equal to q 's degree; if this is impossible take the index with greatest degree less than that of q to minimize the cost of transformations.
- (2) VENONA collects q 's treegrams and represents them by sets of treegram parts. For a given query treegram, VENONA expands the structure number to a set of index treegram structures and removes those labels

that consist of a variable: Variables and the constraints that they impose belong to the matching phase.

- (3) VENONA sorts q 's treegrams according to their selectivity. Seemingly easy, this step has some problems in store: Actually, for a given treegram g , the treegram index includes the number of database trees that contain g ; therefore, it should be easy to use this information for the sorting of the query treegrams. Because of structure expansion or variables, however, every query treegram in general corresponds to a set of index treegrams; furthermore, there are very many different treegrams included in the index—for example, 82,000 of height 3 occur in the trees of the BH^t treebank. In fact, this is the main reason for the treegram-part index: Treegrams must be indexed themselves to render treegram comparison feasible. Even with this approach, it is not desirable, to infer all index treegrams for a given query treegram, if it may turn out that these index treegrams will not be considered further because they are not selective enough. Therefore, VENONA estimates a treegram's selectivity based on the selectivity of its treegram parts. Treegram-part selectivity does not denote the number of treegrams containing that part—this would be incorrect because a treegram part may occur in only a few but very unselective treegrams; rather, this selectivity corresponds to the average number of database trees that a treegram containing this part selects. For assessing a query treegram, VENONA takes the minimum of all its part selectivities; this estimate is relatively crude but efficient, and it works well enough to meet the requirement of small retrieval time.
- (4) VENONA estimates how many query treegrams it has to evaluate to yield a candidate set small enough for the tree matcher; only for those it determines the corresponding index treegrams. Again, this estimate is rather crude: Apart from the inaccurate selectivity of the individual treegrams, there is no way to infer their joint selectivity: Treegrams belonging to the same tree highly depend on each other. As a rule of thumb, VENONA assumes that the more treegrams have already been evaluated the less will be the influence of another one. In analogy to the candidate set, the set of index treegrams for a selected query treegram is not reduced to its minimum; looking up treegram parts and intersecting their corresponding treegrams applies, in turn, the retrieval method to a lower level. Consequently, this process stops when the current set of index treegrams falls below an estimated limit; this limit states that within the joint processing of query treegrams, the set of index treegrams for the current query treegram selects a sufficiently small number of database trees to reduce the candidate set below its limit in turn. As with the others, this estimate is inaccurate but works well.
- (5) VENONA processes these selected treegrams until the candidate set has the desired size—if necessary, falling back on some of the treegrams put aside.
- (6) Finally, the tree matcher selects the answer trees from these candidates.

4.5 Experiences with Venona

For the BH^t application, raw data comprises 161 megabyte. The index for treegrams of degree 9 and height 3 together with the corresponding subindexes amounts to 53 megabyte; generation of the indexes took five hours on a Sun SPARC 20.

To assess retrieval behavior, we selected randomly 100 database trees of maximal height 6 as queries; calculating the candidate set and testing every candidate took three seconds in the average; worst cases took seven seconds. These query trees included Old-Hebrew wordforms, which in general were quite selective; we also selected randomly 100 database trees of maximal height 7 where we substituted any Old-Hebrew wordform by a don't-care symbol. Calculating the candidate set and testing every candidate for these queries took 24 seconds in the average; worst cases took 94 seconds. The experiences of the BH^t project in Munich were also very encouraging: The query tree shown in Figure 1 does not contain selective Old-Hebrew words and its nonterminals are not very selective either; thus its selectivity depends entirely on its labelled structure. Its processing took 12 seconds yielding 37 candidates and one result tree.

5 Conclusion

As a generalization of the well-known n-gram technique, we used treegrams—that means, index trees with fixed height—to answer efficiently containment queries for large databases of multiway trees. We consider two directions as promising for future work:

First, multiway trees used for knowledge representation describe the role of a child by its position; furthermore, because a child can have only one parent, identity constraints for subtrees cannot be expressed. Often, however, named arguments and shared substructures are more appropriate to represent knowledge: Within the field of linguistics, this led to the use of feature-term formalisms, which employ directed acyclic graphs with labeled arcs, cf. [12], [6]: The tree labels now contain feature-value pairs instead of strings; the values in these pairs may be in turn feature terms containing feature-value pairs. Because order is irrelevant for features and their values, this development requires additional interesting retrieval strategies.

Second, VENONA did not exploit the fact that a grammar generated the morphosyntax trees of the BH^t . Phrase structure grammars, however, impose strong constraints concerning structure and positions for nonterminals; consequently, only comparatively few of all possible treegram structures and nonterminal treegrams will appear in the database. A retrieval system dedicated to a given grammar or class of grammars can check a query treegram's wellformedness without using the treegram-part index; in addition, it can represent and expand treegram structures more efficiently. Thus, it may use treegrams of greater height for indexing, which in turn results in a more selective index.

References

- [1] Aho, A. V. and Sloane, N. J.A: *Some Doubly Exponential Sequences*, Fibonacci Quarterly 11, pp. 429–437, 1973
- [2] Argenton, H.: *Indexierung und Retrieval von Feature-Bäumen am Beispiel der linguistischen Analyse von Textkorpora*, PhD Thesis, Department of Computer Science, University of Tübingen, 1997
- [3] Argenton, H. and Becker, P.: *Efficient Retrieval of Labeled Binary Trees*, IEICE Transactions on Information and Systems, Vol. E78-D, No. 11, Special Issue on Advanced Database Technologies, pp. 1433–1438, 1995 also given as a talk on the International Symposium on Advanced Database Technologies and Their Integration (ADTT'94), Nara, October 1994
- [4] Cavnar, W. B.: *Using an N-Gram-Based Document Representation With a Vector Processing Retrieval Model*, in: Harman, D.: Proceedings of the Third Text Retrieval Conference (TREC 3), 1995
- [5] Dubiner, M., Galil, Z. and Magen, E.: *Faster Tree Pattern Matching*, Journal of the ACM, Vol. 41, No. 2, pp. 205–213, 1990
- [6] Gazdar, G., Klein, E., Pullum, G. K. and Sag, I. A.: *Generalized Phrase Structure Grammar*, Basil Blackwell, Oxford, 1985
- [7] Kilpeläinen, P.: *Tree Matching Problems with Applications to Structured Text Databases*, PhD Thesis, Department of Computer Science, University of Helsinki, 1992
- [8] Knuth, D. E.: *Fundamental Algorithms*, Vol. 1, Addison-Wesley, 1973
- [9] Kosaraju, S. R.: *Efficient Tree Pattern Matching*, Proc. 30th IEEE Symposium on Foundations of Computer Science, pp. 178–183, 1989
- [10] Lemke, I. and Sander, G.: *Visualization of Compiler Graphs*, Design Report D3.12.1-1, USAAR-1025-visual, ESPRIT Project #5399 Compare, Universität des Saarlandes, FB 14 Informatik, 1993

- [11] Marcus, M. P., Santorini, B. and Marcinkiewicz, M.A.: *Building a Large Annotated Corpus of English: the Penn Treebank*, Computational Linguistics, Vol. 19, No. 2, pp. 313–390, Special Issue on Using Large Corpora 1993.
- [12] Pollard, C. and Sag, I. A.: *Head-Driven Phrase Structure Grammar*, CSLI, Chicago, 1994
- [13] Richter, W.: *Grundlagen einer althebräischen Grammatik*, Vols. 1-3, EOS, St. Ottilien, 1978–80
- [14] Richter, W.: *AMOS and Its Environment—Our Experiences*, Proc. Computers and Poetic Texts. Symposium on the Use of the Computer for the Study of Literary Texts in Middle Eastern Languages, Bern, 1992
- [15] Specht, G. and Freitag, B.: *AMOS: A Natural Language Parser Implemented as a Deductive Database in LOLA*, in R. Ramakrishnan (ed.): *Applications of Logic Databases*, pp. 197–215, 1995
- [16] Suen, C. Y.: *N-gram Statistics for Natural Language Understanding and Text Processing*, in IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI, Vol. 1, No. 2, pp. 164–172, 1979