Integrating Contents and Structure in Text Retrieval*

Ricardo Baeza-Yates Gonzalo Navarro

Department of Computer Science
University of Chile
Blanco Encalada 2120 - Santiago - Chile
{rbaeza,gnavarro}@dcc.uchile.cl

Abstract

The purpose of a textual database is to store textual documents. These documents have not only textual contents, but also structure. Many traditional text database systems have focused only on querying by contents or by structure. Recently, a number of models integrating both types of queries have appeared. We argue in favor of that integration, and focus our attention on these recent models, covering a representative sampling of the proposals in the field. We pay special attention to the tradeoffs between expressiveness and efficiency, showing the compromises taken by the models. We argue in favor of achieving a good compromise, since being weak in any of these two aspects makes the model useless for many applications.

1 Introduction

Textual databases are deserving more and more attention, due to their multiple applications: libraries, office automation, software engineering, automated dictionaries and encyclopedias, and in general any problem based on storing and retrieving textual information [20].

The purpose of a textual database is to store text documents, structured or not. A textual database is composed of two parts: content and structure (if present). The content is the text itself; the structure relates different parts of the database by some criterion.

Any information model for a text database should comprise three parts: text, structure, and query language. It must specify how the text is seen (i.e. character set, synonyms, stopwords, hidden portions, etc.), the structuring mechanism (i.e. markup, index structure, type of structuring, etc.), and the query language (i.e. what things can be asked, what the answers are, etc.).

The major challenge of any system related to information retrieval is to help its users to find what they need. Textual databases are not as relational databases [16], in which the information is already formatted and meant to be retrieved by a "key". In this case, the information is there, but there is no easy way to extract it. The user must specify what he/she wants, see the results, then reformulate the query, and so on, until is satisfied with the answer. Anything that helps users to find what they want is worth considering.

Traditionally, textual databases have allowed to search their contents (words, phrases, etc.) or their structure (e.g. by navigating through a table of contents), but not both at the same time. Recently, many models have appeared that allow mixing both types of queries.

Mixing contents and structure in queries allows to pose very powerful queries, being much more expressive than each mechanism by itself. By using a query language that integrates both types of queries, the retrieval quality of textual databases can be potentiated.

Because of this, we see these models as an evolution from the classical ones. Suppose, for example, a typical situation of "visual memory": a user remembers that what he/she wants was typed in *italics*, short before a figure that said something about "earth". Searching for the word "earth" may not be a good idea, as well as searching all figures or all the text in italics. What really would help is a language in which we can say "I want a text on italics, near a figure containing the word 'earth' ". This query mixes content and structure of the database, and only new models can handle it.

This work studies this new kind of models. They are not in general as mature as the classical ones. Not only they lack the long process of testing and maturing that traditional models have enjoyed, but also many of them are primitive as software systems, having been implemented mainly as research prototypes. However, the promise they represent in terms of new query facilities and the challenges they pose are so exciting that they deserve a serious study.

On one hand, the "content" of the database is not formatted, but in natural language form. This means that no traditional model relying on formatted data (e.g. the

*This work has been supported in part by FONDECYT grants 1940271 and 1950622.

relational model) or assuming uninterpreted data objects and relying only on their (formatted) attributes (e.g. multimedia databases [4]) is powerful enough to represent the wealth of information contained in the text. The information has to be extracted from the text, but not in a rigid way (see also [38]).

On the other hand, there is no consensus on how the structuring model of a database should be. There are a number of possible models, ranging from no structuring at all to complex interrelation networks. Deciding to use a structuring model involves choosing also what kind of queries about structure can be posed.

Finally, there is no consensus on how powerful a model should be. The more powerful the model, the less efficiently it can be implemented. We are going to pay special attention to this expressiveness/efficiency tradeoff, since being weak in any of these two aspects makes the model impractical for many applications.

The purpose of this work is thus to analyze how the problem of mixing content and structure is addressed in the new models, to compare and classify different approaches, and to discuss their good and bad points. We do not discuss how this aspect should be included in a full model, nor suitable end-user interfaces.

See [31] for another survey, which studies a number of approaches to structured text retrieval. That study is broader, since it also covers indexing and editing aspects, but its focus on data models and query languages is not so deep as ours. Moreover, we only share two of all the models under study.

Throughout this work, we use the term "match" or "match point" to mean a position of the text that matches a searched word or pattern. A "region", "area" or "segment" is a contiguous portion of the text. A "node" is a structural component (e.g. a chapter), and it normally is associated with a text area.

This work is organized as follows. In section 2, we review traditional approaches, showing why they do not serve our purposes. In section 3, we analyze a representative sampling of the new proposals. In section 4, we compare the models and discuss their tradeoffs. Finally, in section 5 we draw our conclusions.

2 Traditional Approaches

This section briefly presents the classical approaches to textual information retrieval (IR). By "classical" we mean that they do not integrate content and structure in queries. However, they address other problems that the new models do not solve in general (e.g. tuples and joins, relevance ranking, and implementation issues such as security, fault-tolerance and concurrency).

The Relational Model [16] expresses the relationships present in a database by a fixed structure of tables, in which the data is organized. By developing an efficient and versatile set of operators to manipulate those tables, this model has been successfully applied to a wide range of information management problems.

However, this model is not suitable for expressing the fuzzy, complex and highly variable structuring present in a textual database [22, 27], not to mention the extraction of information from contents.

Some proposals for integrating the relational model with a textual query language can be found in [42] and TDM [18]. There are also commercial products, such as Oracle SQL*TextRetrieval. However, those embedded systems do not have the best possible performance, since the text is usually stored together with other type of data.

The Traditional IR Model [40, 41] was the first in recognizing the particular information requirements posed by text, and the need to create a model oriented to it. In this model, a database is organized as a set of documents, which are assigned *keywords*, that is, words or phrases meant to describe the semantic contents of the document. Queries are in terms of those keywords, and by examining the correlation between the words of the query and the keywords of the document, the relevance of the document for that query is established. Therefore, the answer to the query consists of a sequence of documents (ranked according to the computed relevance). There are many variations on this topic, for example the boolean, the probabilistic, or the vector model. [20, chapters 11, 12, 14 and 15].

In some cases, we can also query on contents. The only structure allowed on the content consists of non-nested, non-overlapped "fields", regions which cover the whole document. Those fields can only be used to restrict the areas in which match points are to be found.

The problem with this approach is that although it does a very good job in capturing the contents of documents, their fine structure is lost, since they are seen as "black boxes" whose only description are their keywords, their (restricted) fields and their content.

The Full Text Model queries only by contents. A query is a *pattern*, which is searched in the whole database. The answers are the match points and the documents containing them [20, chapter 10]. This search may not use indexes, in which case it has to traverse the whole text database; or it may use some kind of index (e.g. inverted files [20, chapter 3], signature files [20, chapter 4], suffix or PAT arrays [20, chapter 5] and [34], etc.). The problem of this model is that it is not possible to query on the structure of documents. Most commercial products combine full text retrieval with the IR model (e.g. Fulcrum, Topic, BRS, etc.).

Hypertext [12] organizes the database as a graph where nodes are small portions of the database and edges connect nodes related by some design criterion. In this case, the idea is not to use querying but to navigate through the edges. Edges may not only express associations by semantic similarity, but also the structure of the text, cross references, etc. Hypertexts model query by structure well, but not by content; moreover, not always a navigational approach is acceptable. Recently, some models combining a semantic network [24] with structured text have appeared [43], resulting in a hypertext with some facilities to query on the text and its structure.

Related to this we can mention also graph query languages [13] and object-oriented databases [29]. Some attempts have been made to integrate structured text searching into object-oriented databases (e.g. [9]), which generally result in expressing the structure as a (hierarchical) network, linked by part-of attributes. Queries are translated into path expressions in the general language of the database. This approach is powerful but inefficient, since it does not fully consider the semantics of inclusion [14].

3 Novel Models

We present a sample of novel models, which cover different approaches to integrate queries on content and structure in a uniform syntax. This analysis is summarized in Table 1.

The other models studied in [31] are TDM [18] (already mentioned as integrating the relational model with a textual query language), TOMS [17] (a more navigational variation of Lists of References, see later), an old version of Overlapped Lists (see later), MdF (a purely-theoretical model) and others not oriented to querying but to organizing the database, structured editing, etc. (the Bayan system, MAESTRO, Grif and MULTOS).

3.1 A Hybrid Model

Perhaps one of the simplest approaches is [2], which has been partially implemented in SearchCity [1]. The model sees the database as composed of a set of documents (or files, if no structure is defined), which may have fields. Those fields need not to fully cover the text, and can nest and overlap.

The query language is an algebra over pairs (D, M), where D is a set of documents and M is a set of match points in those documents. There are a number of operations to obtain match points: prefix search, proximity, etc. There are operations for union, intersection, difference and complement of both documents and match points; to restrict matches to only some fields, and to retrieve fields containing some match point. Since it is not possible to determine whether a field is included in other (except under certain assumptions on the hierarchy) we say that the model is "flat", and since it is not possible to make certain compositions of expressions involving fields, we say that it is not "compositional".

This model is more expressive than traditional ones, mixing the best of classical IR and full text retrieval, making a first incursion on the problem of mixing queries on structure and contents. This model can be implemented very efficiently, thanks to its simplicity.

3.2 PAT Expressions

This model [39] has been implemented in the PAT Text Searching System [19]. The only index is on match points, there is no indexing on structure. For that purpose, the language allows dynamic definition of structures, based on match point expressions for the beginning and end of regions. It also allows to use externally computed regions. Although the dynamic definition approach is flexible, it relies on specific markup requirements: it must be possible to define regions by simple expressions on match points. For example, it does not allow to recognize the structure of C code. This model has been applied successfully to the computerization of the Oxford English Dictionary (the OED project [3, 21]), because it uses an SGML-like markup.

Structures can have substructures of other type; this fact is not explicit, but derived from the inclusion relationship between regions. Recursive structures (e.g. sections having other sections inside) are not allowed, each structure owns a set of non-overlapping areas of the text. When an operation (e.g. union) results in overlapping regions, only their start points are taken. Thus, the algebra mixes points and regions. This causes a lot of trouble and lack of orthogonality and compositionality in the language, as is pointed out in [39]. Sometimes, it is even impossible to determine statically whether an an expression will yield points or regions (e.g. set union and difference, and "followed by").

Although it is not supposed to depend on the underlying implementation of the index, the operations defined on the text are oriented to the use of a PAT array [20, chapter 5]. Indeed, some operations are included mainly because they are easy to implement with a PAT array, although, as it is pointed out in [39], they are rarely used and difficult to grasp and even to specify (e.g. the longest repeated string).

Despite these drawbacks, the model is a good example of structuring and querying documents by mixing content and structure. What is most important, since all operations are based on the PAT array, they are extremely fast. Operations on areas are also fast, since they are non-overlapping and non-recursive. Thus, it

Model	Structuring mechanism	Contents query language	Structural query language
Hybrid Model [2]	IR-like documents + fields + text. Fields can nest and overlap, but this cannot be later queried. It is a flat model.	Query = matches + documents. Almost all the language is oriented to matches, which are seen as their start point. Expresses distances. Has separate set manipulation tools for matches and documents.	Only to restrict match points to be in a given field or to select fields including match points (selected fields are then seen as match points). Very simple in general.
PAT Expressions [39]	Dynamic definition of regions, by pattern matching. Each region is a flat list of disjoint segments.	Powerful matching language. Handles points and regions. Has set manipulation operations. Expresses distances.	Simple, since structures are flat. Can express inclusion, set manipulation and some very specialized operations.
Overlapped Lists [10]	A set of regions, each one a flat list of possibly overlapping segments.	Not specified. Words and regions are seen in a uniform way, by an inverted list metaphor.	Results can overlap, but not nest. Can express inclusion, union and combinations.
Lists of References [33]	A single hierarchy with attributes in nodes and hypertext links.	Text queries can only be used to restrict other queries.	Results are flat and from the same constructor. Can express inclusions, complex context conditions and set manipulation.
Proximal Nodes [37]	A set of disjoint strict trees (views), Views can overlap. Nodes cannot be dissociated from segments.	Text is a special view. Text queries are leaves of query syntax trees. Text content is accessed only in matching subqueries, thereafter it is seen just as segments. There are powerful distance operators, and special set operators for text.	Can express inclusion, positions, direct and transitive relations, and set manipulation. Can express complex context conditions if they involve proximal nodes.
Tree Matching [27]	A single tree, with strict hierarchy. No more restrictions.	Not specified, orthogonal to the model. It can only be used to restrict sets of nodes of the tree (leaves of patterns). Weak link between content and structure.	Powerful tree pattern matching language. Can distinguish order but not positions nor direct relationships. Can express equality between different parts of a structure, by using logical variables. Set manipulation features via logical connectives.

Table 1: Features of novel models.

achieves high efficiency at the cost of some restrictions, which are reasonable for some applications.

3.3 Overlapped Lists

A recent work with some resemblance to PAT Expressions is [10, 11], an evolution from an older idea [6, 7].

The original idea was to have flat lists of disjoint segments, originated by textual searches or by "regions" like chapters, for example. Both searches were unified by using an extension of inverted lists [20, chapter 3], where areas were indexed the same way as words. The operations were simple: select regions (not) containing other regions; select regions (not) contained in other regions; select a given region or word; and other operations closer to traditional IR (e.g. relevance ranking).

This new work enhances the algebra with overlapping capabilities, some new operators and a framework for an implementation. With these enhancements, the model becomes a reworking of PAT expressions that solves elegantly its semantic problems.

The new operators allow to perform set union, and to combine areas. Combination means selecting the minimal text areas including two segments, for any two segments taken from two sets. A "followed by" operator imposes the additional restriction that the first segment must be before the second one. An "n words" operator generates the set of all (overlapping) sequences of n words of the text.

These operations produce nested and overlapped results. The last ones are allowed, but nesting is avoided by selecting the minimal segments from those that nest. It is not clear whether this feature is good or not to capture the structural properties that information has in practice.

The implementation relies in four primitives, that are used to iterate on the operands to produce the result. Since both the operands and the result can be linearly ordered, the implementation can be very efficient.

3.4 Lists of References

In [33, 32], a model is proposed to achieve uniform definition and querying of structured databases, by means of a common language. It is strongly based on SGML [25], although the ideas are not dependent on it.

The language is somewhat outside the scope that we are studying, since it does not only include data definition features, but also hypertext-like linkages and some operations closer to object-oriented databases (by allowing nodes to have attributes that can also be queried). It also incorporates "external procedures" to the query language, much as in object-oriented databases.

Although the structure of documents is hierarchical (with only one strict hierarchy), answers to queries are flat (only the top-level elements qualify), and all elements must be from the same type (e.g. only sections, or only paragraphs).

Answers to queries are seen as lists of "references". A reference is a pointer to a region of the database. This integrates in an elegant way answers to queries and hypertext links, since all are lists of references. The model has also navigational features to traverse those lists.

This model is very powerful, and because of this, has efficiency problems in its implementation [33]. To make the model suitable for our comparisons, we consider only the portion related to querying structures. Even this portion is quite powerful, and allows to efficiently solve queries by first locating the text matches and then filtering the candidates with the structural restrictions .

3.5 Proximal Nodes

In [35, 37] a model is proposed that finds a good compromise between expressiveness and efficiency. It does not define a specific language, but a model in which it is shown that a number of useful operators can be included, while achieving good efficiency.

Many independent structures can be defined on the same text, each one being a strict hierarchy, but allowing overlaps between areas delimited by different hierarchies (e.g. chapters/sections/paragraphs and pages/lines).

A query can relate different hierarchies, but returns a subset of the nodes of one of them only (i.e. nested elements are allowed in the answers, but no overlaps). Each node has an associated segment, the area of the text it comprises. The segment of a node includes that of its descendants. Text matching queries are modeled as returning nodes from a special "text hierarchy".

The model specifies a fully compositional language with three types of operators: (1) text pattern-matching; (2) to retrieve structural components by name (e.g. all chapters); and (3) to combine other results. The main idea behind the efficient evaluation of these operations is a bottom-up approach, by first searching the queries on contents and then going up the structural part.

Two indices are used, for text and for structure, meant to efficiently solve queries of type 1 and 2 without traversing the whole database. To make operations of type 3 efficient, only operations that relate "nearby" nodes are allowed. Nearby nodes are those whose segments are more or less proximal. This way, the answer is built by traversing both operands in synchronization, leading in most cases to a constant amortized cost per processed element.

It is shown that many useful operators fit into this model: selecting elements that (directly or transitively) include or are included in others, that are included at a given position (e.g. the third paragraph of each chapter), that are after or before others; union, intersection, difference; and many powerful variations. Operations of type 1 form a separate text matching sublanguage, which is independent of the model. In [36, 35], the expressiveness of this model is compared against others and found competitive or superior to most of them.

The model can be efficiently implemented, being linear for most operations and in all practical cases (this is supported by analysis and experimental results). The time to solve a query is proportional to the sum of the sizes of the intermediate results (and not the size of the database). A lazy version is also studied, which behaves better in some situations. This model is as efficient as many others which are less expressive.

3.6 Tree Matching

In [26, 27] a model relying on a single primitive, tree inclusion, is proposed. The idea of tree inclusion is, seeing both the structure of the database and the query (a pattern on structure) as trees, to find an embedding of the pattern into the database which respects the hierarchical relationships between nodes of the pattern. The approach is not meant to be comprehensive in expressiveness, but to study the properties of that primitive.

Two variants are studied. *Ordered inclusion* forces the embedding to respect the left-to-right relations among siblings in the pattern, while *unordered inclusion* does not. The leaves of the pattern can be not only structural elements but also text patterns, meaning that the ancestor of the leaf must contain that pattern.

Tree inclusion is a way to query on structural properties in which the user does not need to be aware of all the details of the structure, but only on what he/she is interested. This stands for "data independence".

Simple queries return the roots of the matches, and the language is enriched by Prolog-like variables, which can be used to express requirements on equality between parts of the matched substructure, and to retrieve another part of the match, not only the root. Logical variables are also used for union and intersection of queries, as well as emulate tuples and joins capabilities.

Although the language is set-oriented, the algorithms work by sequentially obtaining each match. The use of logical variables makes the problem intractable (NP-hard in many cases), and even without them, unordered tree inclusion is NP-complete. Ordered tree inclusion of a pattern P into a textual database T takes $O(|P||T|)$, and $O(|T|)$ if the structure is not recursive (i.e. no node can be the ancestor of an equally labeled node). See [26, 28] for the complexity study.

3.7 Parsed Strings

This approach has also been used for the OED project [3], but in different problems [22]. Those problems are related to *transforming* a database, or to generating new views by processing the data and structure. It has been successfully applied to the Short OED (SOED) project [5], for example, in which the goal is to extract a shorter version from the original dictionary.

Since it has to be a data manipulation rather than a plain query language, the approach is quite different (and we left it aside in some comparisons). The language used to express a database schema is a context-free grammar, that is, the database is structured by giving a grammar to parse its text. The fundamental data structure is the *p-string*, or parsed string, which is composed of a derivation tree plus the underlying text. The parsing process implicitly comprises the work of pattern-matching, there are no further operations to express it. The language relies on the facilities of its host language, Gödel, based on Maple [8].

There are a number of powerful operations that can be performed to manipulate parsed strings: they can be reparsed by another grammar, some nonterminals can be hidden, etc. With those operators, the job of taking into account all the complex variations that appear in the structure of the dictionary is simplified, although not eliminated. The approach is extremely powerful, and it is shown to be relationally complete.

The problem is efficiency. Being such a dynamic approach, it is hard to implement efficiently. In [5], we can see that the operations are really slow, although this was not a concern for the SOED project (since the results had to be worked on further by human experts).

A formalization of a data manipulation model based on grammars, quite similar to this approach, together with a study of its expressiveness, can be found in [23].

4 A Taxonomy of Models

Our aim in this section is to analyze and discuss the different approaches presented by the novel models, dividing this analysis in three parts: structuring power, query language and efficiency. We are going to pay special attention to the way in which decisions about expressiveness affect the efficiency that can be achieved.

Figure 1 presents a graphical version of the expressiveness analysis. The main desirable features are presented, and each model is represented as a set of the features it reasonably supports. Recall that we only consider part of Lists of References.

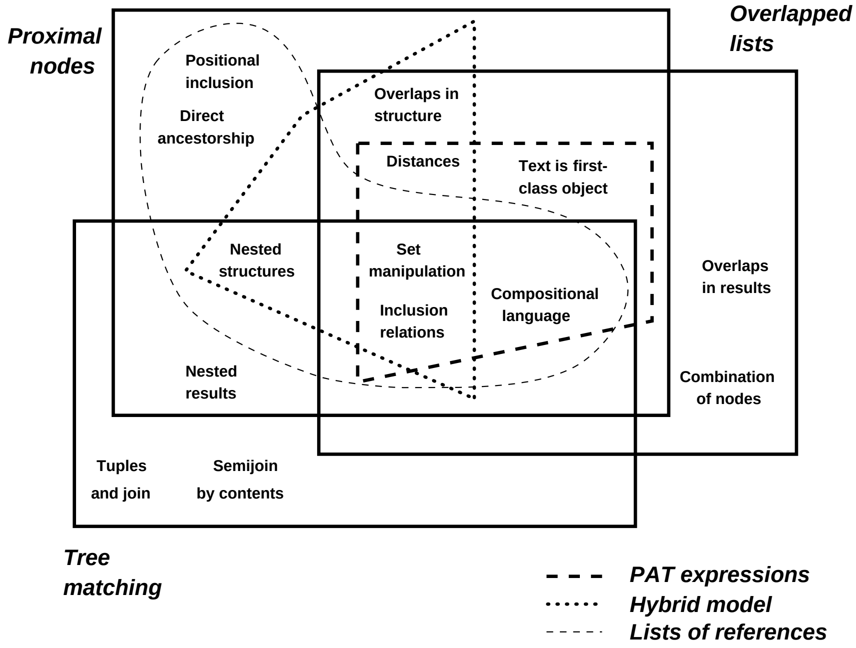

Figure 1: A graphical representation of the expressiveness analysis.

4.1 Structuring Power

Since there is no consensus on how to structure a textual database, there is a wide variety of structuring mechanisms. We point out a number of relevant aspects here. Table 2 summarizes this analysis.

4.1.1 Type of Structure

This refers to the form of the structure allowed. For example, structure can be seen as a graph (e.g. hypertext), as a tree, as a sequence of blocks (e.g. traditional IR), etc. The richer the structure, the more difficult to implement efficiently.

Most of the new models use a hierarchic structure, where ancestorship is dictated by the inclusion relation between structural elements. This is reasonable, since the problem of designing a powerful and efficient query language is hard even under this restriction. Moreover, a hierarchical structure is enough for many applications. Even when the structure must be a network, it is normally not required to query on paths, but they are used in a navigational way. Because of this, a good first step is to satisfactorily solve the problem under the hierarchical hypothesis, and then integrate this solution into a model that allows to specify graphs for navigation (e.g. Lists of References).

Flat: In this case, no structural element can contain another. This is the case of the Hybrid Model, where the inclusion is not forbidden, but it is not possible to query it except under special conditions. The structural elements are intended to limit pattern-matching queries.

Hierarchical: This is the most general case, although many models deviate from this schema. For example, PAT Expressions does not allow recursive structures. Some of them allow many independent hierarchies on the same text (e.g. Proximal Nodes). Overlapped Lists allows overlaps in the structure.

Network: There are no truly network models among those studied. Only Lists of References allows expressing nonhierarchical relationships, but this is to be used in the navigational portion of the language.

4.1.2 Implicit or Explicit Structure

Some models rely on parsing or on markup to determine the structure of the text. That is, there is no a separate hierarchy information, but the text itself encodes the structure.

Each alternative has good and bad points. On one hand, models relying on markup are somewhat dependent on the encoding, since structures that cannot be represented by the provided markup mechanism are forbidden (e.g. it is difficult to have recursive structures). On the other hand, the markup can easily express types of structures that are not allowed in many models using an explicit structure, e.g. overlapping.

Another important consideration relates to querying: while implicit structure models can be very simple and

Model	Type of structure	Implicit or explicit	Static or dynamic	Bound to	Answers
Hybrid Model	Flat	Implicit	Static	Text	Flat
PAT Expressions	Hierarchy (not recursive)	Implicit	Dynamic	Text	Flat
Overlapped Lists	Hierarchy (with overlaps)	Implicit	Dynamic	Text	Overlapped
Lists of References	Hierarchy (and network)	Explicit (single)	Static	Structure	Flat (and of the same type)
Proximal Nodes	Hierarchy	Explicit (multiple)	Static	Intermediate	Nested
Tree Matching	Hierarchy	Explicit (single)	Static	Structure	Nested
p-strings	Hierarchy	Explicit (multiple)	Dynamic	Intermediate	Nested

Table 2: Analysis of structuring power.

efficient by translating structural queries into pattern-matching expressions (thus unifying the implementation), explicit structure models can pose queries about direct ancestorship (i.e. nodes that are direct parents or children of others), and can discriminate ancestorship even when the nodes own exactly the same text area (this happens frequently in parsing, when for example an isolated number is also a constant, a factor, an additive expression, etc.).

Finally, implicit structure databases tend to be easier to reindex.

Implicit: PAT Expressions and Overlapped Lists rely on (SGML-like) markup. Overlapped Lists takes full advantage of the flexibility allowed. The Hybrid Model also uses markup to identify components.

Explicit: The remaining models use explicit structuring. Among them we can classify those allowing just one or many hierarchies (since implicit structuring naturally allows to see the text in many ways):

Single: Tree Matching and Lists of References work with a single hierarchy.

Multiple: Proximal Nodes and *p-strings* allow many independent hierarchies, although only Proximal Nodes relates the different views in a single query.

4.1.3 Static or Dynamic Structures

Some models work reasonably well only under the assumption that the structure is more or less static, while others are more oriented to manipulation of dynamic structures. Generally, implicit structure models allow dynamic structures, and explicit structure models need static ones. The exception is *p-strings*, which is precisely oriented to manipulate structures, allowing to dynamically specify grammars to reparse the database.

Dynamic structure models allow easy reindexing, and querying is normally fast too, while static structure models cannot reindex very efficiently. The advantage of static structuring lies in its related feature: explicit structuring. Of course the alternative of explicit and dynamic structures cannot be efficiently implemented.

Static: Hybrid Model, Lists of References, Proximal Nodes and Tree Matching.

Dynamic: Overlapped Lists, PAT Expressions and *p-strings*.

4.1.4 Link Between Content and Structure

This is a very important point, central to integrating both types of queries. Some models are biased towards one type of query, disregarding in part the other. We classify models attending to how bounded they are to text or to structure.

Being strongly text-bound allows an efficient implementation, since all queries can be translated into a text matching language, which can be efficiently solved. The problem is that many structural manipulations are forbidden (e.g. direct ancestorship). On the other hand, being bound to structure does not involve any advantage to efficiency, but restricts the number of text manipulation features available to the language.

Strongly text-bound: These models normally put the structure into the same text, that is, they use implicit structure. All operations are translated into text operations. These models are PAT Expressions, Hybrid Model and Overlapped Lists.

Strongly structure-bound: These models see the structure as mostly separated from the text. The text is used just to restrict matches in structure, but it is not possible, for example, to retrieve text. These are Tree Matching and Lists of References.

Intermediate: These models pay the same attention to text and to structure, i.e. although the structure has separate identity and it is possible to manipulate it independently from the text, the text is also a first-class object: it can be retrieved and operated the same way as structure nodes. These models are Proximal Nodes and *p-strings*. In Proximal Nodes, the text forms a separate hierarchy to which any text matching result is supposed to belong, and thereafter it can be operated as any true structural node.

4.1.5 Structure of Answers

The structure allowed by the models for answers to queries is not necessarily the same as that of the database. Some models allow just a "flat" set of answers, others nested, and yet others overlapped answers.

Flat answers are clearly the less expressive ones. The other two are formally not comparable, although each application may find one of them more suitable to its needs. On the other hand, all these schemas can be efficiently implemented, although having both nested and overlapped answers seems not suitable for an efficient implementation.

Having a different structure for the database and for the results of operations is not desirable, since it limits the compositionality (i.e. the ability to compose operations freely) and makes the semantics of the model more complex. This is incidentally the case of the models that return only flat answers.

Flat: PAT Expressions, Hybrid Model and Lists of References, although have a richer structure, allow only flat answers. In Lists of References all nodes retrieved must also be of the same type.

Overlapped: Overlapped Lists allow overlapping answers, coherently with its database structure.

Nested: Proximal Nodes and Tree Matching, coherently with their database structure, allow nested answers. Since *p-strings* retrieves trees, one could say that it returns "nested answers", although it is in fact a unique answer.

4.2 Query Language

The other expressiveness aspect under study is the query language offered by the models. Although they must be influenced by the structuring model, the subject constitutes a wholly separate issue.

We analyze different aspects of a query language: text matching, set manipulation, inclusion relationships, and distances. Being a data manipulation language, *p-strings* is be left aside from this analysis.

We summarize the main points of this subsection in Table 3.

4.2.1 Text Matching

This aspect refers to what queries can be posed in the model that refer purely to text pattern-matching. This is normally a wholly independent issue, and we left it aside from this study. We should say, however, that the most powerful text pattern-matching languages are those of PAT Expressions and the Hybrid Model. Some models, such as Proximal Nodes, do not define the text matching sublanguage, being independent of it.

An important issue related to text matching has also been mentioned when discussing structuring power, i.e. how is the link between text and structure. This is a question related to querying too. The support (or the lack of it) for seeing the text as a first-class entity influences also what the query language allows to do with text, although it does not determine it.

We pay attention to what can be done with text regarding each aspect we cover in this section.

4.2.2 Set Manipulation

All models are set-oriented. That means that answers are sets of entities. The models with non-nested answers impose a linear ordering in the sets they manipulate. The area of set manipulation refers to what can be done in each model to combine two sets of results, and under what restrictions. Most models can make set union, intersection and difference of results, although each one has its own special features.

The Hybrid Model supports two kinds of set operations. Since all answers include a set of documents and a set of matches inside those documents, there are for the set of documents and others for the set of matches. This stands for full text support in set manipulation. Another interesting feature is that it supports the "complement" operation on sets.

PAT Expressions allows union, intersection and difference of results, and also a negated version for each selection operator. Since the model has two types of answers (match points and regions) the set operations must be defined to operate in all cases. This forces a very complicated conversion semantics (e.g. to avoid a union of regions resulting in overlapped answers).

Overlapped Lists solves this problem elegantly by al-

Model	Set manipulation	Inclusion relationships	Distances
Hybrid Model	Separate for text and documents. Complement	Restricted to fields	Only in matches
PAT Expressions	Yes. Also negation of operations	Including n and included	Yes, distance-bound
Overlapped Lists	Union and combination	Including and included, plus negations	Combination and "n words"
Lists of References	Yes, but only for nodes of the same type	Including n and included. Restricted direct	None
Proximal Nodes	Yes (same hierarchy). A different set for nodes and for text	Including n and included. Direct and positional inclusion	Both distance-bound and minimal Inside a given node
Tree Matching	Yes, via logical connectives	Tree patterns + variables	None

Table 3: Analysis of query languages.

lowing overlapped answers. However, since nesting is not allowed, the minimal segment is selected whenever a nesting results. Another interesting feature is a variant of union consisting on "combining" nodes instead of taking both of them. The model does not specify any intersection nor difference operator.

Lists of References provides the three set operations on results, but the operands must be all from the same type (i.e. it is not possible to join chapters and figures). This forces answers to be of only one type. Only structural nodes can be operated, not text.

Proximal Nodes also provides the three set operations on results. Since the results cannot overlap, only nodes of the same hierarchy can be operated. This works for all the normal hierarchies, but not for text, since it is a special hierarchy in which overlaps are possible. To operate text, a language is proposed that deals only with text results, and interprets all set operations as if the areas were sets of points. That is, text segments lose their identity and are seen as sets of points that are operated and converted into new segments.

Finally, Tree Matching does not provide set operations explicitly, but through the logic language it is embedded into. The "and" and "or" operations make the same effect as intersection and union of results (since the set of results is generated one answer at a time, ala Prolog). Negation is not provided. Only structural nodes can be operated, not text.

4.2.3 Inclusion Relationships

Inclusion relationships refers to selecting nodes which are included or include others. Although all models provide some kind of selection by inclusion, they greatly differ in the details.

The Hybrid Model has very restrictive inclusion operations. It only allows to restrict a text matching query to occur inside a given field type, and to retrieve all fields (of a given type) containing a given text pattern. This is the only usage of fields, and because of this we say that the model is flat. Observe also that is not fully compositional, since there are no expressions returning fields (the second type of query mentioned retrieves start points of fields, not fields).

PAT Expressions allows to select areas which include at least n areas of another set, and areas included in some area of another set. Recall that there are also negated versions of these operators (i.e. areas not included or not including).

Overlapped Lists is quite the same, but the "including n" feature is not present. It is replaced by using combination operators (one of them already discussed).

Lists of References has complex inclusion operators, allowing to select nodes including n elements of another set, or included in a node of another set. It can also express direct ancestorship (although with some complex restrictions). Finally, the only support provided for text is to select nodes that include a given text pattern.

Proximal Nodes allows many kinds of inclusions. The simplest one selects nodes containing at least n nodes of another set, or contained in some node of another set. Those sets can be from different hierarchies. It also allows "positional inclusion", i.e. to select the i-th element from those included in an element of another set (e.g. the third paragraph of all chapters that have a figure). The sets can also be from different hierarchies. If two sets are from the same hierarchy, a third type of inclusion makes sense: direct ancestorship. It can select nodes that are parents in the structure of at least n nodes of another set, and nodes that are children of a node of another set. i-th children can also be selected.

Tree Matching has all its power in inclusion expressions. It does not use operators such as "included in" or "including", but uses "tree patterns" that must be embedded into the tree of the database structure to find the answers. The answers are the roots of the matches, but logical variables can be used to extract other elements of the embedding. This is also the place where text is included in the model: it is possible to select nodes containing a given text pattern. Two variants of the model are presented, differing in whether they respect or not the ordering among siblings of the query.

4.2.4 Distances

This aspect refers to what restrictions can be expressed regarding the distance between elements in the text. Some models are quite poor in this aspect, which is very important in practice.

The Hybrid Model has distance operations, although they are part of the pattern-matching sublanguage. No distance restrictions can be expressed outside it.

PAT Expressions allows to combine results by selecting elements from a set which are near (or only short before) some element of another set. The distance is measured from start point to start point, which is not the ideal.

Overlapped Lists has another combination operation, "followed by", which combine nodes of one set with nodes of another set that follow them. This allows to select something like "this followed by this followed by this" (not possible in other models). Another feature is the operation "n words", with returns all (overlapping) segments formed by n consecutive words. This is used to force results to be of a given size, e.g. "word followed by word within 5 words" forces the two words to be at distance five.

Proximal Nodes has two different after/before operations, that can be used to relate any two sets of nodes, even from different hierarchies. A first one selects elements from a set which are after/before some element of another set, at a distance of at most k. A third argument (another set of nodes) means that both elements that form the after/before pair must be inside the same minimal node of the third argument. This allows to say, for example, "I want all figures preceded by an emphasized text at a distance of 10, being both in the same section". The other form does not impose a maximum distance k, but selects the nearest candidate node for each element of the second set.

Lists of References and Tree Matching do not support distance operations.

4.3 Query Time Complexity

In order to make a fair comparison between models, not only their expressiveness must be taken into account, but also how efficiently can the different features be implemented.

Since it is not easy to know precise details of the efficiency of a model, we use the order of the involved algorithms. We do not consider indexing times, since it can always be done in reasonable time and querying is more frequent.

From the description of the implementation of the different models, we classify them according to querying times. We measure the efficiency of a query as a function of n, the total size of intermediate results, except otherwise specified. Observe that n is normally much less than the size of the whole database (this is so because most models manage to avoid traversing the whole database to search their candidate results).

- $O(n)$: The Hybrid Model, PAT Expressions and Overlapped Lists are susceptible of a linear implementation, since all results can be put in sequential order and all operations can be implemented by traversing both sequences.

- Almost $O(n)$: Proximal Nodes is linear in most operations. This is achieved by allowing only operators that relate proximal nodes. The few operations that are not linear in theory are linear in most practical situations.

- $O(n \log n)$: The description of the implementation of Lists of References suggests this behavior. There are in fact more costly operations, but they correspond to the more powerful part of the model, that we are not considering here.

- $O(n^2)$: It is not easy to predict the behavior of *p-strings*, but given that there are operations returning $O(n^2)$ results, this is the minimum possible complexity. On the other hand, the times pointed out in [5] show that the operations are very costly.

- Non-polynomial: Tree Matching is even more costly. Although for some good cases it is linear in the size of the whole database, it is not linear in the size of intermediate results. For bad cases, the problems are shown to be NP-complete or NP-hard. The good case is ordered tree inclusion with no logical variables, which is too restrictive.

It is interesting to observe that object-oriented models applied to this problem lead probably to non-polynomial costs, since operations are translated into searching paths in a graph.

5 Conclusions

We began this work by arguing that classical text databases do not allow to mix contents and structure in queries, and that that feature could improve their retrieval capabilities. We deeply analyzed a set of novel models that address that problem, focusing specifically on the modeling and language aspects of the integration of both types of queries. We also paid special attention to the expressiveness versus efficiency tradeoffs.

We have pointed out a number of important issues related to the expressiveness of this kind of models, regarding structuring and querying the database. We showed a number of design decisions that must be made to build a model of this kind, and gave an idea of the cost of each alternative in terms of efficiency.

No model is the best for all applications, especially because the more expressive the model, the less efficient can it be. Each application has its own set of requirements, and should select the most efficient model supporting them.

We have not discussed how this focused subject is embedded into a full database model, including important topics such as indexing, and necessary features such as those present in relational databases, traditional document retrieval, etc. Specifically, we have not considered how to manage tuples and relevance ranking in these models. Further problems arise regarding a sound implementation, with all the required security, concurrency, fault-tolerance, etc.

In [38] it is argued that is better to put a layer integrating a traditional database system with a textual one, than trying to design a language comprising all the features. For example, in [14] it is shown that structure-related queries are handled better by a query engine that knows about the semantics of hierarchies than by a general-purpose object-oriented database language.

Another important issue that we did not cover is the perspective of the user. When we incorporate operators and evaluate the cost of implementing them, we are implicitly assuming that they are useful for the user of the system. This in fact deserves a deeper study, to avoid including theoretically interesting features that are of no use. Moreover, not only it is important to know what the users want to express, but to devise user-friendly languages to implement on top of the algebras we covered here. Some interesting attempts are [27, 30]. On the other hand, a formal and complete study on expressiveness is required. See for example [23, 15].

Acknowledgments

We want to thank José Blakeley for his careful reading and useful suggestions to improve this paper.

References

[1] Ars Innovandi, Santiago, Chile. *Search City 1.1. Text Retrieval for Windows Power Users*, 1992.

[2] R. Baeza-Yates. An hybrid query model for full text retrieval systems. Technical Report DCC-1994-2, Dept. of Computer Science, Univ. of Chile, 1994.

[3] D. Berg, G. Gonnet, and F. Tompa. The new Oxford English Dictionary project at the University of Waterloo. In *Computational Issues in Lexicology and Linguistics, Special Issue in Honour of Bernard Quemada*, 1991.

[4] E. Bertino, F. Rabitti, and S. Gibbs. Query processing in a multimedia document system. *ACM TOIS*, 6(1):1–41, January 1988.

[5] G. Blake, T. Bray, and F. Tompa. Shortening the OED: Experience with a grammar-defined database. *ACM TIS*, 10(3):213–232, July 1992.

[6] F. Burkowski. An algebra for hierarchically organized text-dominated databases. *Information Processing & Management*, 28(3):333–348, 1992.

[7] F. Burkowski. Retrieval activities in a database consisting of heterogeneous collections of structured text. In *Proc. ACM SIGIR'92*, pages 112–125, 1992.

[8] B. Char, K. Geddes, G. Gonnet, M. Monagan, and S. Watt. *Maple Reference Manual, 5th Edition*. Waterloo, 1988.

[9] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. From structured documents to novel query facilities. In *Proc. ACM SIGMOD'94*, pages 313–324, 1994.

[10] C. Clarke, G. Cormack, and F. Burkowski. An algebra for structured text search and a framework for its implementation. *The Computer Journal*, 1995.

[11] C. Clarke, G. Cormack, and F. Burkowski. Schema-independent retrieval from heterogeneous structured text. In *Procs. of the 4th Annual Symposium on Document Analysis and Information Retrieval*, April 1995.

[12] J. Conklin. Hypertext: An introduction and survey. *IEEE Computer*, 20(9):17–41, September 1987.

[13] M. Consens and A. Mendelzon. Hy$^+$: A hygraph-based query and visualization system. In *Proc. ACM SIGMOD'93*, pages 511–516, 1993. Video presentation summary.

[14] M. Consens and T. Milo. Optimizing queries on files. In *Proc. ACM SIGMOD'94*, pages 301–312, 1994.

[15] M. Consens and T. Milo. Algebras for querying text regions. In *Proc. PODS'95*, 1995. California.

[16] C. Date. *An Introduction to Database Systems.* Addison-Wesley, Reading, Massachusetts, 6th edition, 1995.

[17] S. Deerwester, K. Waclena, and M. LaMar. A textual object management system. In *Proc. ACM SIGIR'92*, pages 126–139, 1992.

[18] B. Desai, P. Goyal, and S. Sadri. A data model for use with formatted and textual data. *Journal of ASIS*, 37(3):158–165, 1986.

[19] H. Fawcett. *PAT 3.3 User's Guide.* UW Centre for the New OED and Text Research, Univ. of Waterloo, 1989.

[20] W. Frakes and R. Baeza-Yates, editors. *Information Retrieval: Data Structures and Algorithms.* Prentice-Hall, Englewood Cliffs, New Jersey 07632, 1992.

[21] G. Gonnet. Examples of PAT applied to the Oxford English Dictionary. Technical Report OED-87-02, UW Centre for the New OED and Text Research, Univ. of Waterloo, 1987.

[22] G. Gonnet and F. Tompa. Mind Your Grammar: a new approach to modelling text. In *Proc. VLDB'87*, pages 339–346, 1987.

[23] M. Gyssens, J. Paredaens, and D. Van Gucht. A grammar-based approach towards unifying hierarchical data models. In *Proc. ACM SIGMOD'89*, pages 263–272, 1989.

[24] R. Hull and R. King. Semantic database modelling: Survey, applications and research issues. *ACM Computing Surveys*, 19(3):201–260, 1987.

[25] International Standards Organization. *Information Processing — Text and Office Systems — Standard Generalized Markup Language (SGML)*, 1986. ISO 8879-1986.

[26] P. Kilpeläinen. Tree matching problems with applications to structured text dat abases. Technical Report A-1992-6, Dept. of Computer Science, Univ. of Helsinki, November 1992.

[27] P. Kilpeläinen and H. Mannila. Retrieval from hierarchical texts by partial patterns. In *Proc. ACM SIGIR'93*, pages 214–222, 1993.

[28] P. Kilpeläinen and H. Mannila. Ordered and unordered tree inclusion. *SIAM Journal on Computing*, 24(2):340–356, April 1995.

[29] W. Kim and F. Lochovski, editors. *Object-Oriented Concepts, Databases and Applications.* Addison-Wesley, Reading, Massachusetts, 1989.

[30] E. Kuikka and A. Salminen. Two-dimensional filters for structured text. Dept. of Computer Science, Univ. of Waterloo (submitted for publications), 1995.

[31] A. Loeffen. Text databases: A survey of text models and systems. *ACM SIGMOD Conference. ACM SIGMOD RECORD*, 23(1):97–106, March 1994.

[32] I. MacLeod. Storage and retrieval of structured documents. *Information Processing & Management*, 26(2):197–208, 1990.

[33] I. MacLeod. A query language for retrieving information from hierarchic text structures. *The Computer Journal*, 34(3):254–264, 1991.

[34] U. Manber and G. Myers. Suffix arrays: A new method for on-line string searches. In *Proc. ACM-SIAM'90*, pages 319–327, 1990.

[35] G. Navarro. A language for queries on structure and contents of textual databases. Master's thesis, Dept. of Computer Science, Univ. of Chile, April 1995.

[36] G. Navarro and R. Baeza-Yates. Expressive power of a new model for structured text databases. In *Proc. PANEL'95*, pages 1151–1162, 1995.

[37] G. Navarro and R. Baeza-Yates. A language for queries on structure and contents of textual databases. In *Proc. ACM SIGIR'95*, pages 93–101, 1995.

[38] R. Sacks-Davis, T. Arnold-Moore, and J. Zobel. Database systems for structured documents. In *Proc. ADTI'94*, pages 272–283, 1994.

[39] A. Salminen and F. Tompa. PAT expressions: an algebra for text search. In *COMPLEX'92*, pages 309–332, 1992.

[40] G. Salton and M. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, New York, 1983.

[41] G. Salton and M. McGill. *Automatic text processing.* Addison-Wesley, Reading, Massachusetts, 1989.

[42] M. Stonebraker, H. Stettner, N. Lynn, J. Kalash, and A. Guttman. Document processing in a relational database system. *ACM TOIS*, 1(2):143–158, April 1983.

[43] J. Tague, A. Salminen, and C. McClellan. Complete formal model for information retrieval systems. In *Proc. ACM SIGIR'91*, pages 14–20, 1991.