Chapter 2

State of the Art

At present, there are relatively few works that investigate the topic "similarity search in XML data". However, over the past years much research has been conducted in related areas:

- The database community has adopted the well-known concept of formal query languages to the requirements of XML.
- A theoretical branch of database research has worked on flexible mapping techniques for trees and graphs, which help to query XML data with complex, heterogeneous structure.
- The information retrieval community has enriched query languages for structured documents with text-retrieval concepts like term weighting and relevance ranking.
- In theoretical computer science and computational biology, researchers have developed similarity measures for labeled trees, and have shown their applications to XML.

We review the state of the art in each of the mentioned research areas, develop a taxonomy of existing approaches, and discuss why none of the reviewed approaches meets the requirements listed in Section 1.2. In the following section, we take a closer look at XML and semistructured data to prepare the groundwork for the subsequent reviews.

2.1 XML and Semistructured Data

XML is a practical subset of the Standard Generalised Markup Language (SGML) defined in the ISO standard 8879:1986 [ISO86]. SGML was developed for maintaining large repositories of structured documentation, but it is not well suited for the Internet-wide exchange

of documents. There are many optional features the sender and receiver of a document must agree on in order to remain interoperable. The syntax of SGML is complex, ambiguous, and includes many rarely used options. XML differs from SGML primarily through simplifying the complex formalism of SGML.

XML documents have a physical and a logical structure. The physical structure is made up of *entities* and entity references, which is a concept similar to macros in programming languages. The logical structure consists of nested *elements*. Each document must have a single root element. An element begins with a start-tag and ends with an end-tag enclosed in angle brackets. Both tags must have the same *name*. All elements and text passages between the start-tag and the end-tag of an element make up its *content*. An element can have certain *attributes* that are listed within its start-tag. An attribute is a pair consisting of a name and a value. Figure 2.1 shows a part of an XML document. The document starts with a preamble that specifies the version of the XML specification and the name of the character set used within the document. The preamble is followed by the start-tag of the root element `catalog`. All other elements are nested within that root element. The elements `cd` and `track` each have one attribute.

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
<catalog>
  ...
  <cd year="2001">
    <performer>Ashkenazy</performer>
    <composer>Rachmaninov</composer>
    <title>Piano concerto no. 1</title>
    <tracks>
      <track length="13:25">
        <title>Vivace</title>
      </track>
      ...
    </tracks>
  </cd>
  ...
</catalog>
```

Figure 2.1: A part of an XML document.

The XML specification [BPSM00] does not predefine the names for elements and attributes. Communities that want to exchange documents must agree on a set of common names, and must specify the relationships between elements with given names. Such an agreement may be formalized as a Document Type Description (DTD), which allows the users to verify that each component of a document occurs in a valid place within the interchanged data stream. However, a Document Type Description (DTD) is not required. Any document that obeys

the XML syntax is a well-formed XML document.

The meaning of a name in a certain context of a document can be encoded in the programs that interpret the document. For some applications, the interpretation of tag names can be left to the users — the XML specification explicitly states that "XML documents should be human-legible and reasonably clear". Descriptive markup is one of the key features of XML that allows the users to take advantage of the structure for displaying, searching, and browsing.

It is particularly the combination of content (the text sequences) and schema (the names and structures of elements and attributes) within the same document that makes XML well suited for the representation of semistructured data, which is "data that does not have a regular and static structure like data found in a relational database but whose schema is dynamic and may contain missing data or types" [HMG97]. Such properties are typical for data found on the Internet, like web pages and mails. XML is considered to be the common format for structured data, semistructured data, and even unstructured data.

XML is a markup language, not a storage format. However, a natural interpretation of an XML document is a rooted tree or, if references between elements are included, a rooted directed graph. The element and attribute names are mapped to inner nodes (or, alternatively, to edges) of the tree or graph, and the text sequences enclosed by the innermost elements are assigned to leaves. We frequently make use of this interpretation in the subsequent reviews.

2.2 Query Languages for XML

The motivation behind the design of query languages for XML is the fact that XML has its own implied data model, which is neither that of relational databases nor that of object-oriented or object-relational databases. This fact has triggered intensive research that has produced many language proposals tailored to the XML data model. Some of them, like Lorel [AQM+97] and YATL [SC98], were originally designed for semistructured data, and have later been adapted to XML. Others, like XML-QL [DFF+98], XQL [RLS98], and YAXQL [Moe00] were developed directly for XML. Fruitful impulses came from other fields of research such as functional programming, which resulted in query languages like XDuce [HP00], logic programming, which had influences on query languages like XPathLog [May01], and graphical user interfaces, which lead to the development of query languages like XML-GL [CCD+99]. The papers [FSW99, BC00] provide comparisons between some of the mentioned languages. Currently, a World Wide Web Consortium (W3C) working group advances a new language standard called

```
FOR $c IN document("media.xml")/catalog/cd
  WHERE $c/@year > 2000 AND $c/composer = "Rachmaninov"
RETURN $c/title
```

(a) XQuery

```
WHERE
  <catalog>
    <cd year=$y>
      <title>$t</title>
      <composer>Rachmaninov</composer>
    </cd>
  </catalog> IN "media.xml",
  $y > 2000
CONSTRUCT
  <title>$t</title>
```

(b) XML-QL

```
MAKE
  title [ $t ]
MATCH "media.xml" WITH
  catalog [ cd [ @year    [ $y ],
                 title    [ $t ],
                 composer [ $c ] ] ]
WHERE
  $y > 2000 AND $c = "Rachmaninov"
```

(c) YATL

Figure 2.2: The query "select the titles from all CDs that appeared after 2000 and contain works by Rachmaninov" expressed in XQuery, XML-QL, and YATL.

XQuery [BCF+02], which is based directly on the language Quilt [CRF00], but also adopts and integrates several aspects from other XML query languages. In the following section, we concentrate on the common characteristics of the various languages; in Section 2.2.2 we take a closer look at language features related to semistructured data.

2.2.1 The Basic Query Components

XML queries typically consist of three parts: *pattern* clauses, *filter* clauses, and *construction* clauses [FSW99]. A pattern clause specifies names and hierarchical relationships of elements and attributes that must occur in the documents. A filter specifies conditions on the selected values. A construction clause creates the resulting document by specifying a skeleton of nested elements and attributes, in which the selected and transformed values and substructures are inserted. Some languages allow pattern and filter clauses within a construction clause.

Figure 2.2 shows a simple query expressed in three different XML query languages. Although the languages provide quite different syntactic constructs to express the same semantics, it is easy to see the common basic structure of all three queries: Pattern clauses in XQuery are specified by XPath [CD99] expressions. In the example query depicted in Figure 2.2(a), the pattern consists of the source selector `document("media.xml")` and the path `/catalog/cd` that starts at the root of the source document. XML-QL uses patterns that follow the syntax of XML. A pattern is introduced by the keyword `WHERE`, and ends with the specification of the source to which the pattern should apply (see Figure 2.2(b)). A YATL pattern starts with the

keyword `MATCH`, followed by the specification of the source. The selection part of the pattern models the structure of XML documents, where hierarchical relationships are indicated by brackets (see Figure 2.2(c)). Filters in XQuery are implemented either as XPath filters or as `WHERE` clauses. Our example shows the latter case. XML-QL groups patterns and filters within `WHERE` blocks, separated by commas. YATL uses separate blocks for patterns (`MATCH`) and filters (`WHERE`). All three languages have explicit construction blocks, introduced by the keywords `RETURN` in XQuery, `CONSTRUCT` in XML-QL, and `MAKE` in YATL.

Besides the selection, filtering, and construction of documents, most of the languages also support operations like inner and outer joins, aggregation, grouping, sorting, and quantification. While all those operations have their counterparts in query languages for relational and object oriented databases, there are also some language elements that cope with a peculiarity of XML: its structural heterogeneity. We take a closer look at those elements in the next section.

2.2.2 Language Aspects Related to Semistructured Data

XML documents may be semistructured, or may contain semistructured parts. Querying semistructured data with a conventional query language is costly and tedious, because the users must be aware of all structural deviations and must formulate queries that explicitly encode them. To allow querying with incomplete knowledge, *regular path expressions* have been proposed for document databases [CACS94, CCM96] and semistructured data [AQM+97]. Today, regular path expressions are part of almost all query languages for XML. The W3C recommendation XPath provides a standardized syntax for path expressions with (restricted) regular features.

Among all XML query languages, Lorel is the one that offers the most sophisticated regular path expressions. Besides sequences like `cd.composer.name`, it supports optional labels (`cd(.composer)?.name`), alternatives (`cd.(composer|performer).name`), repetitions (`cd(.composer)+.name`), skipping of an arbitrary number of labels (`cd.#.name`), and even wildcards for labels (`cd.composer.%name`).

Type coercion is another feature of Lorel that allows to cope with structural heterogeneity. The data model of Lorel considers a collection of XML documents as a graph of objects and values. The objective of type coercion is to force comparisons between objects and/or values, even if the objects or values have different types. For example, in the inequality "4.3" < 5, both sides are coerced to the type *real* in order to perform a type-preserving comparison.

All newer XML query languages use simpler regular path expressions than Lorel, and support either no type coercion or more restricted variants of it. It is particularly interesting that XQuery, the future standard language, uses rigid typing and rather restricted regular path expressions. This observation indicates that XQuery is designed as a language for XML documents with well-defined structures, rather than a user-friendly language for ad-hoc queries to semistructured data.

2.3 Flexible Query Mappings for Trees and Graphs

Regular path expressions are a powerful language primitive to cope with the structural heterogeneity of XML data. In many cases, however, the additional flexibility provided by regular path expressions is not sufficient at all. The users still have to know that structural heterogeneity exists, and that therefore a regular path expression is necessary. They also have to know which expression is appropriate to match all of the variants of a substructure that may exist in the database. *Flexible mappings* require less knowledge about the structure of the data. The users have to specify only "common" cases in the query, and the system is responsible to find matches similar to those demanded by the query. We distinguish between models that valuate the degree of query relaxation and models that do not.

2.3.1 Flexible Query Mappings without Valuation

One of the simplest cases of flexible query mapping is the automatic relaxation of parent-child relationships to ancestor-descendant relationships. An automatic relaxation differs from the application of a "skip" operator that is part of many XML query languages (e.g., the operator "//" in XPath), because skip operators must be applied explicitly, whereas automatic relaxations are performed implicitly.

Kilpeläinen proposes ordered and unordered *tree inclusion* [Kil92] as a means to query a tree-structured database with only partial knowledge of the structure. The users phrase tree-shaped queries that specify the labels and the hierarchical relationships of database nodes they are interested in. The system relaxes all parent-child relationships to ancestor-descendant relationships. Surprisingly, the unordered variant of the tree-inclusion problem has proven to be NP-complete. Meuss [Meu00] shows that the unordered tree-inclusion problem can be solved in polynomial time if the injectivity property of the embedding function is dropped, and if siblings of the query tree are allowed to be mapped to nodes on the same document path.

Kanza et. al. [KNS99] propose a mapping semantics that allows partial query matches. A partial match of a query pattern in a graph-structured database is a subgraph that does not bind all query variables. In order to prevent a proliferation of answers, only maximal answers are accepted in the sense that there are no assignments that bind more variables and satisfy the query conditions. If the query is a tree or Directed Acyclic Graph (DAG), then all maximal answers can be computed in polynomial time with respect to the size of the query, the database, and the result. If the query is a general graph, then the problem is NP-complete. In a later work [KS01], Kanza and Sagiv investigate mappings that allow the permutation of path-connected query nodes. They propose a *semiflexible* mapping semantics, where query paths are mapped to document paths such that every query label has an occurrence in the document, but the occurrences may appear in an other order than specified. A *flexible* semantics does not require that paths are mapped to paths, but only requires that the matches of each pair of edge-connected query nodes must be connected by a path within the data.

An important application of flexible query mappings is the integration of data sources with different — but similar — structures. Domenig and Dittrich [DD01] propose the query language SOQL, which supports a `LIKE` operator whose argument is a path. The query processor rewrites the path in such a way that the resulting paths are similar to the original path. Using the set of rewritten paths, it selects those paths that have matches in at least one source.

2.3.2 Flexible Query Mappings with Valuation

Flexible mappings improve the usefulness of query languages, especially if the structure of the data is heterogeneous, and if the users have incomplete knowledge on the structure. However, the mapping flexibility may lead to a huge number of results. The valuation of the query-result similarity, and the retrieval of the best n results avoid the explosion of the result set.

Damiani and Tanca [DT00] model a collection of XML documents as a *fuzzy labeled graph* [CC92] and a query as (unweighted) labeled graph. To answer a query, the closure of the document graph is computed by inserting shortcut edges between each pair of nodes that are connected by a path of arbitrary length through the original graph. The weight of a shortcut edge is a function of the weights associated to the edges of the original path. The query graph is exactly embedded into the closure graph. To assign a score to an embedding, the normalized sum of the weights of the matching edges is computed. The embedding images are ranked by decreasing scores. To lower the time complexity, the authors propose to compute the closure only once and to store the edges and their weights in a table. Unfortunately, the table can reach a quadratic size with respect to the number of nodes in the original graph.

TreeSearch [SZW01, SWG02] is a very simple but efficient technique to find approximate results for tree-pattern queries. A query tree is split up into paths, and each path is separately searched in the document trees. The score of a document is determined by the number of root-to-leaf query paths that have matches in the document. A consequence of this valuation model is that nodes close to the query root contribute more to the scores of the documents than nodes close to the query leaves. Queries may contain "placeholder" nodes, which match arbitrary document nodes, and "variable length don't cares", which match arbitrary document paths. The authors point out that the proposed algorithm only works correctly if both the query tree and the document trees have no sibling nodes with the same label. Otherwise, the algorithm may return "false positives", which are document trees that include the same leaf-to-root paths as the query tree, but do not have the same shape. More precisely, an inner query-tree node may be mapped to several document-tree nodes such that none of its matches is a common ancestor of the matches of its children.

Amer-Yahia et. al. [ACS02] use *valuated tree-pattern relaxations* as a means to find approximate results for structured queries to XML documents. Lee [Lee02] reinterprets that model in the context of the query-modification framework proposed by Chaudhuri [Cha90]. A query is a single tree pattern whose nodes are labeled with element names. Selection conditions with respect to the content of the documents are not supported. The model allows the relaxation of parent-child relationships between query nodes to ancestor-descendant relationships, the generalization of query labels, and the deletion of query nodes. To this end, an exact weight and a relaxed weight are associated to each node and to each edge in the pattern. The exact weight is used if the node or edge, respectively, has a match in the documents such that the parent-child relationships in the pattern are preserved. The relaxed weight of a node is used if its label does not match, but one of the other specializations of its generalized label matches. If a node must be deleted, then a zero weight is used. The actual weight of an edge depends on the distance (in terms of hierarchy levels) between the matches of the parent node and the child node, and may range between the exact and the relaxed weight associated to the edge. Amer-Yahia et. al. propose an algorithm that finds all results whose total weights are below a certain threshold, and sketch an algorithm that finds the best n results.

2.4 XML and Information Retrieval

Information Retrieval is a science that investigates models and techniques to answer queries by retrieving objects from a database that satisfy the users' information needs. A query may be posed in a natural language, in a formal language, or simply as sequence of terms.

Database objects may be text documents, multimedia files, hypermedia documents, and others. Typically, an object is represented by a set of descriptors. *Vagueness* is one of the key concepts of Information Retrieval (IR), because the mapping of a user's information needs to a query is vague, the mapping of database objects to descriptors is vague, and the relationships between query descriptors and object descriptors are vague [Fuh92]. Due to the importance and the widespread availability of text documents, many researchers have investigated the problem of answering queries to text databases, and many models have been proposed. These include the Boolean model and its derivates [SFW83, Lee94], the Vector Space Model (VSM) [Sal68], models based on probabilistic logic [CLvRC98], possibilistic logic [Lal98], fuzzy sets [Cro94, KBP99], evidential reasoning [CCH92], genetic algorithms [YKR93, PGF00], and neural networks [Kwo89, WH91]. Baeza-Yates and Ribeiro-Neto [BR99] give an excellent survey of models and algorithms used for text retrieval.

In contrast to the large body of work available for text retrieval, few query languages and retrieval models for structured documents (and especially for XML documents) exist. We distinguish between text algebras, focused retrieval, and structured languages that support relevance ranking.

2.4.1 Query Languages and Algebras for Structured Text Databases

The objective of early query languages and algebras for text databases with hierarchical structure was to allow querying for content and structure in a uniform way. Here, we briefly review some important approaches. Baeza-Yates and Navarro [NB96] give a more comprehensive analysis of the expressiveness and the efficiency of several models. Other reviews are provided in [RH90, Loe94, CM95].

PAT expressions [ST93] allow the dynamic definition of text regions using pattern-matching expressions that specify the begin and end of regions. Only the content of the database is indexed. Once the regions are created, they can be manipulated with standard set operations like union and intersection, but also with very specialized operations like "longest repeated string". The algebra cannot handle overlapping regions. If an operation leads to overlapping regions, only their start points are taken. Because some operations return regions and others return points, it is sometimes impossible to statically determine the return type of a nested expression.

Burkowski [Bur92a, Bur92b] proposes an algebra based on lists of disjoint regions. Text words and text regions are indexed the same way, using an extension of inverted lists. The model has

been extended to overlapping regions in later works [CCB95a, CCB95b]. The algebra supports several query operations that ask for the relationships between regions, and the relationships between regions and words. Examples are: the nesting of regions, the containment of a word in a region, the union of regions, and the smallest region covering some other regions. Additional restrictions on the order of regions are imposed by a "followed by" operator.

MacLeod [Mac90, Mac91] proposes a language to define and query structured databases in a uniform way. The language allows the users to query not only the content and hierarchical structure of the database, but also hyperlinks between nodes and attributes assigned to nodes. The answer to a query is a list of references to regions in the database.

Proximal nodes [Nav95, NB97] is another model that integrates both content and hierarchical structure by specifying a data model, two types of indexes, and a query language. For a given text, many independent hierarchical structures may be specified — but only one of them can be used to answer a single query. The query language consists of a pattern-matching sublanguage, operators to select nodes by name, and operators to combine the selected results. Supported combination operations are, e.g., containment, intersection, union, and difference of regions.

With the exception of [Bur92b], where an implementation technique for the computation of term weights in the context of text regions is proposed, the notion of relevance is not considered in any model.

2.4.2 Focused Document Retrieval

Focused document retrieval aims at retrieving the best document components that satisfy a query. The returned components are displayed to the users, and then constitute entry points from where the users can decide to browse through the structure of the documents.

A simple model, introduced in [SKW01], computes the lowest common ancestors of the matches of the query keywords. The subdocuments rooted at those ancestors are retrieved as results.

More sophisticated models shift from a structure-oriented to a content-oriented view: Structured documents are interpreted as trees, where the leaves comprise the "raw" data, and the information content of non-leaf components is derived from the information content of its subcomponents. The matching function proposed in [CK96] locates each component C that implies the query (exhaustivity of C), and that has no component that implies the query (specifity of C). Lalmas [Lal97, LR98] extends this model by incorporating uncertainty in

both the indexing and the retrieval of structured documents using Dempster-Shafer's Theory of Evidence (D-S theory) [Sha76]. The degree of belief that a component is an answer to a query is aggregated from the information content of the component and the information content of its subcomponents via the combination rule of the D-S theory. The notion of uncommitted belief of the D-S theory reflects the fact that indexing itself is a uncertain process.

So far, queries are restricted to content. Lalmas [Lal00] extends the model to a uniform representation of content and structure based on evidential reasoning [RLS92]. Components are represented by descriptors for both content and structure. Descriptors for structural parts may be adapted to a more general context. For example, the descriptor `is_chapter` may be changed to `has_chapter` in the component containing the chapter. A query is a logical sentence consisting of propositions connected by "`and`", "`or`", and "`not`". A query sentence Q retrieves a (composite) component C, if C is described by a sentence S that implies Q with a degree of uncertainty. For example, the query `is_chapter` $\wedge$ `about_wine` selects components whose structure is described by the proposition `is_chapter` and whose content is described by the proposition `about_wine`. Because queries are "flat" logical formulae, nested hierarchical queries cannot be expressed. For example, the query `is_chapter` $\wedge$ `has_title` $\wedge$ `about_wine` locates chapters that have a title and are about wine — but it is not possible to formulate a query that ensures that the title of the chapter is about wine.

Kazai et. al. [KLR01] extend the aggregation-based model for focused retrieval to linear relationships and link relationships between components. Also, the criterion for the retrieval of a document component is now more general: A component is returned if all (most, at least one, etc.) of its related components are relevant to the query. Each component is represented by a collection of weighted index terms, where the weight of a term is a function of the occurrences of the term in the component and in its connected components. The combination function is based on *fuzzy aggregation* [Yag96].

2.4.3 Combining Structured Queries and Relevance Ranking

A relatively new branch of research investigates the combination of structured queries and relevance ranking. This combination is challenging because the model for the calculation of relevance scores must take not only the structure of the documents into account, but also the constraints on structure and content specified by the query.

Arnold-Moore et. al. recognized the necessity of a retrieval language that supports ranking by textual similarity [AFL+95]. They introduced the ELF data model and the SGQL query

language, which provides three ranking operators. The operators compute similarity scores for the results of queries asking for keywords in a structural context. However, the authors do not provide a method to calculate the similarity scores for such keyword-in-context queries, and they point out the open problem of how to combine similarity scores calculated by subqueries of a complex query.

Navarro et. al. [NBVF98] introduce a simple query language and a generic ranking model. A query is a set of subqueries, each annotated with a positive or negative weight. A subquery is a logical formula consisting of presence and inclusion operators. To compute a score for a particular document, the query processor counts the number of occurrences of each subquery within the document. It then multiplies the user-defined weights by these numbers, and adds the products to obtain a score for the document. The authors highlight the simplicity of their ranking model and mention that other models can also be used within their framework.

An adaptation of conventional IR techniques to text-rich XML documents is proposed by Hayashi et. al. [HTK00]. Their system uses a format file, which defines a mapping of elements to search fields. The terms within a search field are weighted by using a variant of the tf*idf formula [SM83]. A query is a Boolean formula consisting of terms and tag-paths, which specify the fields in which the terms are searched. The query engine uses the format file to select the appropriate indexes created for the search fields, evaluates the fields separately, and combines the relevance scores of the parts according to the Boolean query operators. If the query engine encounters an "`or`" operator, it adds the scores of the operands; if it encounters an "`and`" operator, it selects the minimum of the scores. A restriction of this approach is the static mapping of elements to fields, which requires an adaptation of the format file whenever a new document type is indexed.

Myaeng et. al. [MJKZ98] use an inference network with two connected subnets to model both queries and SGML documents. The document subnet represents the hierarchical relationships between elements, and between elements and terms. The query subnet models the relationships (e.g., Boolean operators) between the query terms. The query terms are connected only to those leaves in the document subnet that fulfill the hierarchical relationships of the query, which means that the query instantiates a part of the document subnet. The degree to which an element, at any level, supports the query is computed by combining the probabilities of its included elements, which are propagated along the network.

Wolff et. al. [WFC99, WFC00] propose XPRES, a probabilistic model for the ranking of results for structured queries to structured documents. A query is a set of pairs consisting of a term (a keyword or a phrase) and an associated structural role. The structural role of a term

can be the name of an element, but also a simple path or regular path made up of element names. The structural role specifies the elements in which the term must appear. With this language, the users cannot express that one term should appear in the title, and that another should appear in the body of the *same* chapter of a book. The retrieval function valuates the descriptiveness of a term with respect to the structural role of the term. To this end, the authors modify a retrieval function proposed by Croft [Cro83] such that it measures the frequency and the inverse document frequency of a term within the scope of the elements specified by its structural role. The structural role may also have a weight that can be incorporated in the retrieval function.

XIRQL [FG00, FG01] is a language proposal that incorporates the notion of term weights, data types, vague predicates, semantic relativism, and relevance-oriented search into XQL. Term weights are defined with respect to predefined index elements; they are adapted to broader contexts via augmentation. Terms in index nodes represent probabilistic events, which are assumed to be independent. Retrieval results are Boolean combinations of events based on a probabilistic relational algebra [FR97]. Datatypes capture different kinds of data, e.g., person names, locations, and dates. Vague predicates help to search for similar values of a data type, e.g., phonetically similar person names. The term "semantic relativism" describes the capability to create generalizations. For example, the operator `~person` matches both elements and attributes with the name `person`; the operator `#person` refers to elements and attributes with type *person* or one of its subtypes. Finally, relevance-oriented search omits structural conditions and aims at retrieving the best document components as described for focused retrieval.

Theobald and Weikum [TW00, TW02] propose the XML query language XXL, which adopts several concepts from XML-QL and from other XML query languages. XXL supports a similarity operator that works at the level of values, and also at the level of element and attribute names. For example, the expression `~cd` selects elements with a name similar to the name `cd`. The query engine assigns weights to the matches and combines them according to the different Boolean operators in the query, assuming probabilistic independence. Sizov et. al. [STW01] propose to use the classical tf*idf formula to calculate the term weights used by the XXL query engine, and to use ontologies to determine the weights of element names.

ELIXIR [CK02] is a query language comparable to XXL, except it additionally supports similarity joins on strings. The language uses WHIRL [Coh98, Coh00], which is an extension of relational databases that can perform "soft joins" based on the similarity of textual identifiers.

A retrieval technique that combines structured queries with the similarity measure of the

VSM is proposed in [SM00, SM02]. The key concept of this approach is the use of labeled trees for the modeling of queries, collections, documents, and even terms. A collection of XML documents is mapped to a single data tree; each subtree of the data tree is considered to be a logical document. All logical documents that have (partial) embeddings of the query tree according to the tree-inclusion formalism [Kil92] are results. To allow the relevance valuation of the results, the authors extend the traditional term concept to subtrees of query trees and logical documents: For each structural term that occurs in the query tree or in the data tree, the term frequency and the inverse document frequency are determined, and a term weight is calculated. As in the traditional VSM, the similarity between a query and a result is defined as the closeness between the vector of (structural) query terms and the vector of (structural) document terms.

2.5 Distance Measures for Labeled Trees

Distance measures for labeled trees are of great interest in several domains: In molecular biology and genetics, they help to compare protein structures [SZ90] and phylogenetic trees [DHJ+97]. In programming languages, the comparison of parse trees allows syntactic error recovery and correction [Tai78]. In data warehouses, digital libraries, and authoring systems, different versions of documents are managed. Changes between versions of tree-structured documents can be detected and analyzed using similarity measures for trees [CRGW96, CAM02].

The definition of the classical tree-edit distance was given by Kuo-Chung Tai [Tai79]. He proposed to use three kinds of edit operations: *Changing* a node u means changing the label of u. *Deleting* a node u means making the children of u become the children of the parent of u and then removing u. *Inserting* is the complement of deleting. Each operation has a cost. Edit operations can be composed to sequences; the cost of a sequence is the sum of the costs of the operations. To measure the closeness between two trees T_1 and T_2, edit sequences are applied to T_1 such that the resulting tree is congruent to T_2. The similarity between T_1 and T_2 is defined to be the edit sequence with the lowest cost. The edit operations can be represented by a *mapping*, which is a graphical specification of the edit operations applied to the nodes in the trees. A mapping M consists of node pairs; each pair (u, v) represents two corresponding nodes in T_1 and T_2. If u and v have distinct labels, then (u, v) represents a renaming. Otherwise, it is a null edit. Nodes in T_1 not occurring in M represent deletions. Similarly, nodes in T_2 not occurring in M represent insertions. The cost of M is the sum of all renamings represented by the node pairs in M, plus the sum of deletions of nodes in T_1,

plus the sum of all insertions of nodes into T_2. For each edit sequence S, a mapping with a cost less than or equal to the cost of S exists, and for each mapping an edit sequence with the same cost exists. This implies that the distance between T_1 and T_2 is the cost of the mapping with the lowest cost among all mappings between T_1 and T_2 [AG97].

The edit distance is defined for ordered and unordered trees. A tree is ordered if the left-to-right order of sibling nodes is fixed. The mappings for ordered trees preserve the order of siblings, i.e., for any two nodes u, v in T_1, if u is to the left of v, then the image of u must be to the left of the image of v in T_2. The problem of finding the edit distance between two ordered trees T_1 and T_2 can be solved in $O(|N_1| \cdot |N_2| \cdot depth(T_1) \cdot depth(T_2))$ time, where $|N_1|$ ($|N_2|$) is the number of nodes in T_1 (T_2). An early variant of ordered tree edit was proposed by Selkow [Sel77]. His model restricts the insertions and deletions to the leaves of the trees. The time complexity of the algorithm solving this problem is $O(|N_1| \cdot |N_2|)$.

Unfortunately, the edit-distance problem for unordered trees is MAX SNP-hard — even if both trees are binary —, which means that there is no polynomial time approximation scheme (PTAS) for this problem unless P=NP [ZSS92]. Motivated by the hardness result for the editing distance between unordered trees, some restricted measures have been proposed. The restrictions concern the order of edit sequences, the permitted mappings, or both. The alignment of unordered trees [JWZ94] corresponds to a restricted tree edit distance in which all insertions precede all deletions. Even the computation of this restricted measure is MAX SNP-hard when at least one of the trees is allowed to have an arbitrary degree (the maximum number of children of a node). The main idea of a model for unordered trees introduced in [TT88] and refined in [Zha93] is to restrict the mappings between the trees: Separate subtrees in T_1 must be mapped to separate subtrees in T_2. A further restriction of this model is proposed in [ZWS95]. Here, only nodes with at most two neighbors can be inserted into or deleted from T_1. Both problems can be solved in $O(|N_1| \cdot |N_2| \cdot D)$ time, where D is a factor that depends on the degree of the trees.

2.6 Similarity Search in Tree-Structured Data

Similarity search is concerned with the task to locate objects that are "close" to a query according to a given distance measure. There is a large number of domains where similarity search is of interest: In text databases, users may want to find strings allowing a number of errors; in genome databases they may be interested in protein or DNA sequences that are variations of a given sequence; in multimedia databases the users may want to select images

or audio clips that are similar to a sample; in spatial databases they may want to locate objects that are geometrically close to a query object; etc.

Tree-similarity search is a variant of similarity search where the query and the objects are labeled trees. An object is "close" to the query if the tree distance between the query and the object is below a certain threshold. Obviously, every distance measure reviewed in the previous section can be used for tree-similarity search.

Approximate tree-pattern matching is an important specialization of tree-similarity search. Here, the set of objects is defined as the set of trees included in a single *data tree*. Note that an included tree is any tree that can be constructed as follows: Choose a subtree of the data tree and prune away some of its subtrees. The time complexity of an algorithm that solves a pattern-matching problem cannot be below the complexity of an algorithm for the corresponding distance problem. In particular, tree pattern matching is MAX SNP-hard for the unordered tree-edit distance and the unordered tree-alignment distance.

Very few restricted measures for unordered trees allow approximate tree-pattern matching in polynomial time [Zha93, ZWS95], and only in [Zha93], a pattern-matching algorithm is proposed. This algorithm benefits from restricted mappings which, as discussed in the previous section, require that separate subtrees of the query tree must be mapped to separate subtrees of the data tree.

Shasha et. al. [ZSW94] propose a pattern-matching algorithm with "variable length don't cares" based on the tree-edit distance for ordered trees. A "variable length don't care" is a sequence of virtual nodes that matches an arbitrary path in the data tree. The query language WAQL [HWC+99] uses this algorithm to implement the special operator

$$D \texttt{ has } T \texttt{ with dist } op\ k,$$

which evaluates to true if the document D includes a tree whose distance to T is $op\ k$, where op is a numerical comparison operator like "$<$".

2.7 A Taxonomy of XML Query Languages and Retrieval Models

XML query languages developed by the database community are tailored to *structure-rich* documents, and they focus on *applications* that query, transform, and integrate XML documents. Most languages require that the documents have relatively regular and static structure. This is particularly true for the upcoming standard XQuery, which uses a static type system.

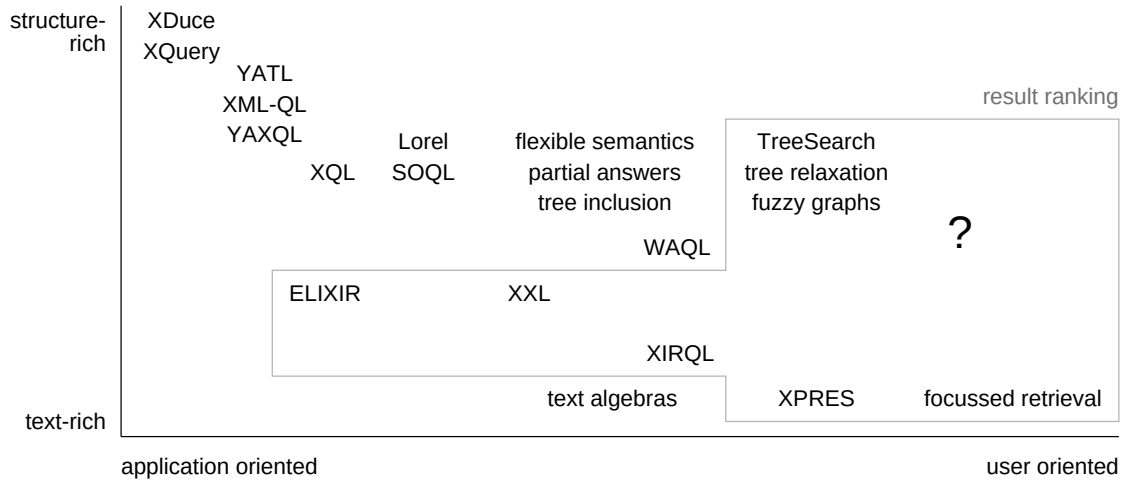

Figure 2.3: A rough taxonomy of XML query languages and retrieval models. The approaches are proposed in the following papers: XDuce [HP00], XQuery [BCF+02], YATL [SC98], XML-QL [DFF+98], YAXQL [Moe00], XQL [RLS98], Lorel [AQM+97], SOQL [DD01], flexible queries [KS01], partial matches [KNS99], tree inclusion [Kil92], TreeSearch [SZW01], tree relaxation [ACS02], fuzzy graphs [DT00], WAQL [HWC+99], ELIXIR [CK02], XXL [TW02], XIRQL [FG01], XPRES [WFC99]. Text algebras and techniques for focused retrieval are proposed in several papers. We review them in Section 2.4.1 and Section 2.4.2, respectively. The question mark indicates the position a new approach should be located at.

The structure of the documents must be fully known to a user who formulates queries. The semantics is rigid, which means that there must be a perfect match between the conditions specified in the query and the structure and content of the documents retrieved.

Query languages and retrieval models developed in the IR community take a *user-oriented* view. They are tailored to *text-rich* documents, where the XML markup structures a large, continuous text. The focus of IR languages is on content; the query structure is interpreted rigidly.

Figure 2.3 shows a rough taxonomy of existing XML query languages and retrieval models. XML query languages developed by the database community are located in the upper-left corner; they are designed for structure-rich documents and for use in applications. Note that older proposals like Lorel have more features related to user-oriented querying than younger developments like XQuery. The lower-right corner represents user-oriented languages or models for text-rich documents. The user-friendliest models are those developed for focused retrieval, where queries are just sequences of keywords.

Table 2.1 on the next page shows a more detailed comparison between query languages and retrieval models that (i) support structured queries and (ii) either have a non-standard map-

query language or retrieval model	content similarity	name similarity	partial matches	edge expansion	permutations	valuation method
ELIXIR [CK02]	√	–	–	–	–	WHIRL [Coh00]
flexible queries [KS01]	–	–	–	–	√	–
fuzzy graphs [DT00]	–	–	–	√	–	total weight
partial answers [KNS99]	–	–	√	–	–	–
tree inclusion [Kil92]	–	–	–	√	–	–
tree relaxation [ACS02]	–	√	√	√	–	total weight
TreeSearch [SZW01]	–	–	√	—	–	matching paths[1]
WAQL [HWC+99]	√	√	√	√	–	–
XIRQL [FG01]	√	√	–	–	–	probabilistic
XPRES [WFC99]	√	–	–	–	–	probabilistic
XXL [TW02]	√	√	–	–	–	probabilistic

Table 2.1: Comparison of selected features of XML query languages and retrieval models.

ping semantics or support a kind of similarity valuation. The table columns list features we think are favorable — or even necessary — to realize similarity search in XML data. The column "content similarity" shows the ability of the language or model to find content similar to the keywords or values specified by the query. Some languages also support the mapping of query labels to similar element or attribute names, which is indicated in the column "name similarity". A language allows "partial matches" if a query may return answers that do not comprise bindings for all element and attribute names occurring in the query. The column "edge expansion" is marked for those languages or models that *automatically* relax query edges to paths. Note that this feature differs from operators like "`a//b`" or "`a.*.b`", where the users must explicitly state that an edge should be relaxed. The column "permutations" lists all languages that allow the reordering of hierarchical relationships of query selectors. Finally, the "valuation method" shows the method used for the similarity valuation.

2.8 What is Missing?

The approaches shown in Figure 2.3, and particularly those listed in Table 2.1 have some of the features that we demanded in our definition of the objectives of this thesis (see Section 1.2). However, none of the reviewed query languages and retrieval models fulfills all, or at least the majority of our demands. In the following, we review and extend the requirements for the new language and discuss why no existing approach meets our requirements.

[1] The score of a document is determined by the number of query paths that have matches in the document.

Syntax. The query language should allow to incorporate the hierarchical *structure* of XML documents into the search process to yield *precise* queries. The formulation of a query should be *simple* even for occasional users and should require only a *partial knowledge* of the structure and content of the documents. Consequently, the language should be located in the far-right area of Figure 2.3. For experienced users, the language should offer extended features that allow to specify the users' information needs more precisely. This includes syntactic primitives to modify the interpretation of a query, but also standard components of query languages like Boolean operators, data types, and comparison predicates for keywords and numerical data.

Semantics. The interpretation of a structured query should be *vague*. A query should retrieve results that exactly match the query, but also answers with a structure and content *similar* to the selection conditions specified by the query. The similarity measure should build on all five concepts of flexible query interpretation listed in Table 2.1, and it should combine them in such a way that a unique score can be assigned to each query result. The scores should be the basis for a *ranking* of the results. Moreover, it should be possible to *adapt* the basic parameters of the similarity measure. Because the new language should be able to handle complex, heterogenous structure (but also text-rich documents), it should be located above the vertical center of Figure 2.3.

Algorithms. The query processor should be able to answer a query in *sublinear time* with respect to the size of the collection. In particular, it should retrieve the *best* n results quickly. The answer time of the system should be low, even if the collection is large, and the structure of the data is very complex and heterogeneous. Fast query processing requires that the similarity measure allows an implementation based on well-known database techniques, which includes the use of index structures and query-execution plans.

The XML query languages reviewed in Section 2.2 clearly do not meet our requirements as we have already pointed out in Section 1.1. The complex syntax of those languages in conjunction with the rigid interpretation of queries indicate that their main field of usage are applications that query and transform XML documents with relatively regular structure.

Flexible query mappings are a means to add non-standard semantics to the selection parts of a query. None of the flexible mapping models without valuation reviewed in Section 2.3.1 meets our requirements, because one of our main requirements is the assignment of scores to the results in order to establish a relevance ranking. The approach of Damiani and Tanca [DT00] supports the ranking of results. However, it also does not meet our requirements, because it

is restricted to a single kind of flexibility, namely the expansion of query edges (by means of weighted shortcuts inserted into the original data graph). The TreeSearch approach [SZW01, SWG02] allows only the deletion of nodes. The valuation model is extremely simple and cannot be adapted to the properties of queries and documents. Moreover, the possible retrieval of "false positives" restricts the usability of the approach. Tree-pattern relaxation [ACS02] is a flexible mapping model that supports name similarity, partial matches, and edge expansion. Despite its capabilities to retrieve results similar to the query, the approach lacks several features demanded in this section: It only supports tree-pattern queries (without Boolean operators) that match the structural parts of XML documents. All content-related features are missing. In particular, no selection predicates for keywords and numerical data exist. Moreover, the ability to find results with similar element or attribute names is limited, because only one generalized label for each query label can be specified. For example, the label cd may be generalized to media. Then, each subtype of media like dvd or book gets the same score. We believe that a name-similarity model should allow to differently valuate the renamings cd → dvd and cd → book.

Although there is a wide range of information retrieval models for text-rich XML documents, we claim that none of them is appropriate for similarity search in XML documents with a complex, heterogeneous structure. All query algebras for text databases interpret the structure in a rigid way, and only one model incorporates term weights to enable the ranking of results. The goal of models for focused retrieval is to find the best document components that satisfy a query. Queries are either unstructured, or they can include simple structural constraints (as discussed in Section 2.4.2). This contrasts with our objective to support queries that reflect the hierarchical structure of XML documents, which means that the query language should fully support nested structural expressions. All approaches that aim at combining structured queries and relevance ranking (reviewed in Section 2.4.3) are designed for documents with large text passages, but with simple, regular structure. None of the models allows for partial structural matches, automatic edge expansion, or permutations; and only XIRQL [FG00, FG01] and XXL [TW00, TW02] consider the similarity of element and attribute names.

The use of a tree distance measure as a basis for the semantics of a query language is promising, because it enriches the language with similarity-search capabilities. Unfortunately, none of the distance measures reviewed in the Sections 2.5 and 2.6 satisfies our requirements. First of all, the distance measures are not tailored to XML data. They are often based on ordered trees — but ordered trees are inappropriate for querying, because the order of the elements, and especially the order of the keywords within the documents may be inconsistent. Even if it were consistent, the order might not be known to the users. Furthermore, the measures rely

on a notion of similarity that is based on neighborhood and not on semantic closeness. The measures are also not designed to support queries with Boolean operators, and other selection predicates than those testing the equality of labels. Tree-similarity measures do not only have an inappropriate semantics, they also require algorithms with very high time complexities. Most similarity measures for unordered trees are NP-hard. Even the restricted variants for unordered trees (and also the similarity measures for ordered trees) require algorithms that touch every node in the query tree *and* every node in the in the data tree at least once. Such algorithms cannot have a sublinear time complexity as demanded, and will have unacceptable answer times for large document collections.